A Good Fishman Knows All the Angles: A Critical Evaluation of Google's Phishing Page Classifier

Changqing Miao
School of Information, Renmin University of China
Beijing, China
miaochangqing@ruc.edu.cn

Jianan Feng
School of Information, Renmin University of China
Beijing, China
jiananfeng@ruc.edu.cn

Wei You
School of Information, Renmin University of China
Beijing, China
youwei@ruc.edu.cn

Wenchang Shi
School of Information, Renmin University of China
Beijing, China
wenchang@ruc.edu.cn

Jianjun Huang*
School of Information, Renmin University of China
Beijing, China
hjj@ruc.edu.cn

Bin Liang*
School of Information, Renmin University of China
Beijing, China
liangb@ruc.edu.cn

ABSTRACT

Phishing is one of the most popular cyberspace attacks. Phishing detection has been integrated into mainstream browsers to provide online protection. The phishing detector of Google Chrome reports millions of phishing attacks per week. However, it has been proven to be vulnerable to evasion attacks. Currently, Google has upgraded Chrome/Chromium's phishing detector, introducing a CNN-based image classifier. The robustness of the new-generation detector is unclear. If it can be bypassed, its billions of users will be exposed to sophisticated attackers. This paper presents a critical evaluation of Google's phishing detector by targeted evasion testing, and investigates corresponding defensive techniques. First, we propose a three-stage evasion method against the phishing image classifier. The experiments show that it can be completely bypassed with adversarial phishing pages generated using the proposed method. Meanwhile, the phishing pages still preserve their visual utility. Second, we introduce two defense techniques to enhance the phishing detection model. The results show that even using lightweight defense methods can significantly improve the model robustness. Our research reveals that Google's new-generation phishing classifier is very vulnerable to targeted evasion attacks. A sophisticated phishers can know how to fool the classifier. Billions of Chrome users are being exposed to potential phishing attacks. To improve its robustness, necessary security enhancements should be introduced.

CCS CONCEPTS

• **Security and privacy** → **Browser security**; *Malware and its mitigation*; **Intrusion/anomaly detection and malware mitigation**; • **Computing methodologies** → *Machine learning*.

*Corresponding authors.

CCS '23, November 26–30, 2023, Copenhagen, Denmark.

ACM ISBN 979-8-4007-0050-7/23/11…$15.00
https://doi.org/10.1145/3576915.3623199

KEYWORDS

Phishing attacks, Image classifiers, Google Chrome/Chromium, Evasion

ACM Reference Format:
Changqing Miao, Jianan Feng, Wei You, Wenchang Shi, Jianjun Huang, and Bin Liang. 2023. A Good Fishman Knows All the Angles: A Critical Evaluation of Google's Phishing Page Classifier. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security (CCS '23), November 26–30, 2023, Copenhagen, Denmark.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3576915.3623199

1 INTRODUCTION

Phishing attacks pose a significant security threat in cyberspace. In Q3 2022, APWG [12] reported a record-breaking 1,270,883 phishing attacks, marking the worst quarter for phishing incidents observed by the organization. A substantial portion of these attacks involved phishing webpages, which deceive users by imitating legitimate webpages.

Many machine-learning-based classifiers [3, 5, 6, 11, 22, 55, 56, 60–62] have been developed to detect phishing webpage attacks. They leverage various features, such as URL, DOM tree, visual elements, and meta-information, to predict whether the given webpage is a phishing one.

It is natural to integrate the phishing classifier within the web browser. The browser can directly access page content, and can conveniently extract all desirable features. More importantly, the integrated classifier can identify the phishing page and block it before being seen by users. In other words, it can provide an online protection for end users.

For instance, Google Chrome, the most widely used browser with five billion users, has equipped a phishing detection mechanism for years. The old version mechanism consists of two components, a URL blacklist, and a logistic regression classifier. They had been proven to be vulnerable to evasion attacks [35]. Currently, the phishing detection mechanism is upgraded. An new CNN-based image classifier is introduced to enhance the phishing detection. A webpage must pass through all the three detection components before being displayed to users. Otherwise, a phishing block page, as illustrated in Figure 1, will be presented to warn the page viewer.

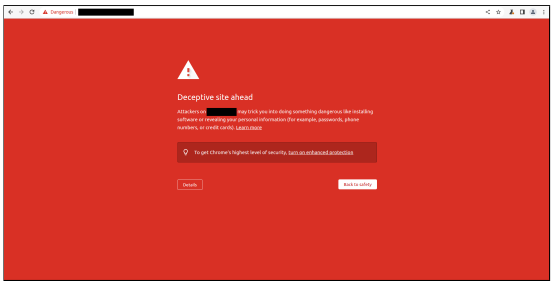


Figure 1: The block page for a phishing webpage.

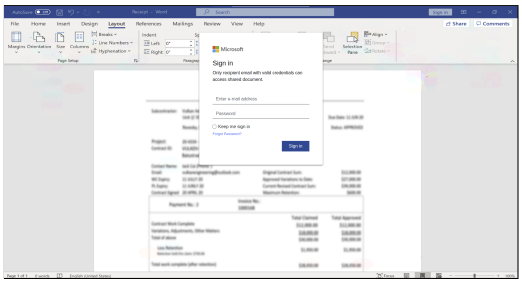


Figure 2: An example of perturbed phishing page.

The new-generation detection mechanism have proven to be effective, reporting over 3,500,000 web browsing warnings per week in July 2022 [25].

However, the robustness of the new-generation mechanism is still unclear. An important question arises naturally as *whether it can also be bypassed like its old version*, as done in [35]. For this purpose, we present a critical evaluation of Google's phishing page detector via launching targeted evasion test and investigating potential defense methods in this paper.

As shown in Figure 3, a phishing page must circumvent all the two classifiers to bypass the new-generation detector. Surprisingly, the evasion technique presented in [35] is still effective to bypass the current logistic regression classifier. To this end, in this study, we focus on evading the newly introduced image classifier.

To fool the logistic regression classifier, the attacker can dress up the target page by altering the URLs, DOM elements, or terms of it, and conceal the modification with various tricks. As a result, the render result of the modified page is completely identical with that of the original one.

However, to fool the image classifier, the visual appearance of the target page will be perturbed to generate a different model input, leading to misclassification. To ensure the accuracy of the critical evaluation, we need to ensure that the proposed evasion method is practicable. The primary challenge lies in preserving the visual utility of the page. First, a phishing page should closely resemble the impersonated page as much as possible, to successfully deceive end users. Second, the introduced perturbation should be imperceptible, and able to mislead the classifier.

Our approach. We propose a three-stage evasion method against Google's phishing image classifier, consisting of adversarial screenshot generation, inverse downsampling and webpage construction. First, the candidate pixels are identified based on their saliency and classification contribution, and perturbed using an iterative optimization process to generate an adversarial screenshot. However, the Chrome/Chromium classifier is a simplified TF-Lite model, which is designed for improved deployability but lacks the capability to perform iterative optimization. To enable the optimization for adversarial sample generation, we reconstruct a standard TensorFlow model from the TF-Lite model by introducing a dequantization layer and a quantization layer before and after each original layer. Second, we inversely downsample the adversarial screenshot to obtain a full-size version. Downsampling the input image is a common preprocessing mechanism in industrial models. A practical attack needs to reflect the perturbations within the downsampled image in the original webpage. We specifically design a programming-based method to perform upsampling while preserving the attack information and visual utility. Finally, a dressed up phishing page is constructed base on the full-size adversarial screenshot. The visual utility-preserving of the target page is ensured by setting a pixel optimization scope and introducing specific elements in the programming objective function.

Our experiments show that the proposed evasion method can effectively bypass the new-generation phishing image classifier. We collected 135 real-world phishing pages that can be spotted by the image classifier. However, all of them can bypass the detection after being perturbed with our method. As the same time, their visual utility is preserved well. An example of the perturbed page is shown in Figure 2. We can see that it is difficult for ordinary users to detect the introduced perturbation.

Furthermore, we enforce two simple defense approaches to improve the classifier robustness. We find that most of the phishing pages can be re-detected after introducing lightweight techniques.

We can draw the conclusion that Google's new-generation phishing detection is still vulnerable. Billions of users are being exposed to potential phishing attacks. Fortunately, the classifier can be significantly enhanced by introducing necessary defense mechanisms.

This paper makes the following main contributions:

- Google's new-generation phishing classifier is proven to be vulnerable. It can be completely evaded with sophisticated adversarial samples. Numerous Chrome/Chromium users still face serious phishing threat. The dataset of our work can be found in the Github repository[1].
- We propose a three-stage evasion method to dress up phishing pages, making them effectively bypass Google's phishing image classifier.
- The proposed method presents an adversarial sample generation technique for the deployment-oriented simplified model, and offers a utility-preserving inverse downsampling technique to bridge the downsampling gap arising from model input preprocessing, enriching the arsenal of attacking industrial models.
- We apply two potential defense methods, and prove that introducing lightweight techniques can significantly enhance its performance for phishing detection.

[1]https://github.com/GoodPhishman/A-Good-Fishman-Knows-All-the-Angles

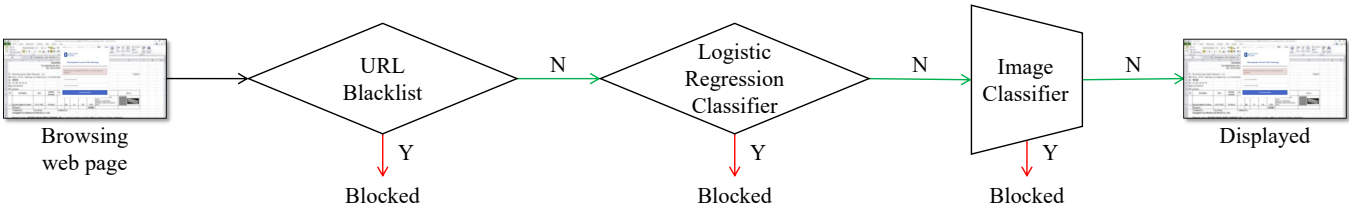


Figure 3: Google's current phishing detection mechanism.

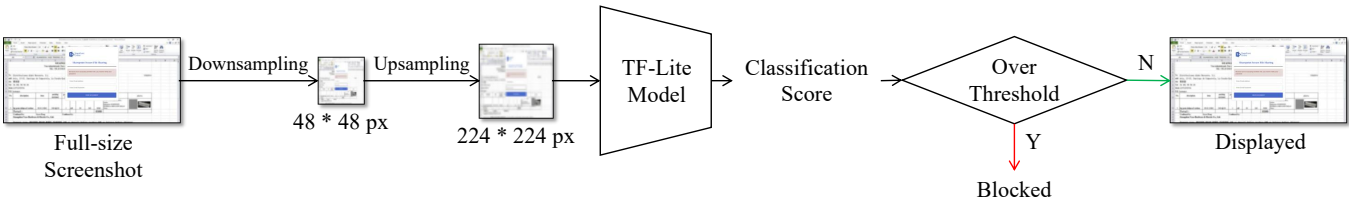


Figure 4: Google's newly-added phishing image classification stage.

2 BACKGROUND

In this section, we demystify the details of Chrome's phishing webpage detection mechanism by analyzing its open-source Chromium. The detection consists of URL blacklisting, logistic regression classification, and image classification, as shown in Figure 3.

URL Blacklisting is triggered when the user requests a webpage with a URL. Before the browser sends a request, the requested URL must first pass the blacklist check. Chrome/Chromium hashes the URL and searches in the maintained URL blacklist. If the hashed URL appears in the blacklist, the request is canceled, and a block page is shown; otherwise, the request proceeds to the target host.

Logistic Regression Classification begins after the webpage request has been sent, the response has been loaded, and the rendering of the webpage has started. In this stage, the detector extracts three types of features from the webpage, including URL features (e.g., the protocol, domain, and port of the target host's URL), DOM features (e.g., with or without <form> elements, number of <script> elements, fraction of page links that use HTTPS), and term features (i.e., the words used in the webpage). After extracting these features, the detector matches them with pre-defined rules, and adds the weight of any matched features to the total score. If the total score is over 0.5, the webpage is reported to Google for further inspection and will be blocked if confirmed as a phishing page.

Image Classification is the newly-added stage. Its workflow is shown in Figure 4. In this stage, Chrome/Chromium takes a screenshot of the rendered webpage, downsamples it to 48*48 px, then upsamples it to 224*224 px, and feeds it into a convolutional neural network (CNN) for image classification. The CNN is deployed as a *TF-Lite* model and consists of 55 hidden layers. The output of the CNN is a 19-dimensional vector. The 19 dimensions are composed of one benign score and 18 phishing type scores, each with a predefined threshold. When a particular dimension in the phishing dimensions exceeds its threshold, the current page will be recognized as a potential phishing page. There are not any public information about phishing webpage categories. Besides, the labels for each output dimension in the TF-Lite model within Chromium have been hashed. The above factors limit our ability to understand their exact meaning. However, based on the observation on the model output for the collected phishing webpages, we have reasons to infer that the 18 phishing dimensions may correspond to some common phishing target websites respectively, like Amazon, Facebook, Microsoft and Netflix.

3 METHODOLOGY

We present our evasion method, whose goal is to bypass Google's phishing image classifier while preserving the visual utility of the phishing webpage. ***It is important to note that specific sensitive details of our method have been intentionally omitted to prevent them from being used for malicious purposes.***

Figure 5 presents the workflow of our evasion method. We first extract the image classifier model (❶) used for phishing detection in Chrome/Chromium. The extracted model is reconstructed (❷), making it more suitable for evasion. The reconstructed model is used to generate adversarial screenshot (❸), which takes the downsampled original screenshot of a phishing webpage as input and generates a downsampled adversarial screenshot. The downsampled adversarial screenshot is further inversely downsampled (❹) to generate a full-size adversarial screenshot. Finally, a fully-functional webpage that can bypass the phishing detector is constructed (❺), according to the full-size adversarial screenshot.

3.1 Model Extraction

Model extraction involves two parts: locating model uses in source code and extracting the model data. However, with tens of millions of lines of code, it is nearly impossible to analyze the source code purely by manual effort. Thus, we developed a method for locating the uses of the phishing image classifier, which leverages dynamic debugging.

Locating Model Use. We first heuristically search the entire source code base of the open-source edition Chromium to identify functions that contain related keywords (e.g., "phishing", "classifier",

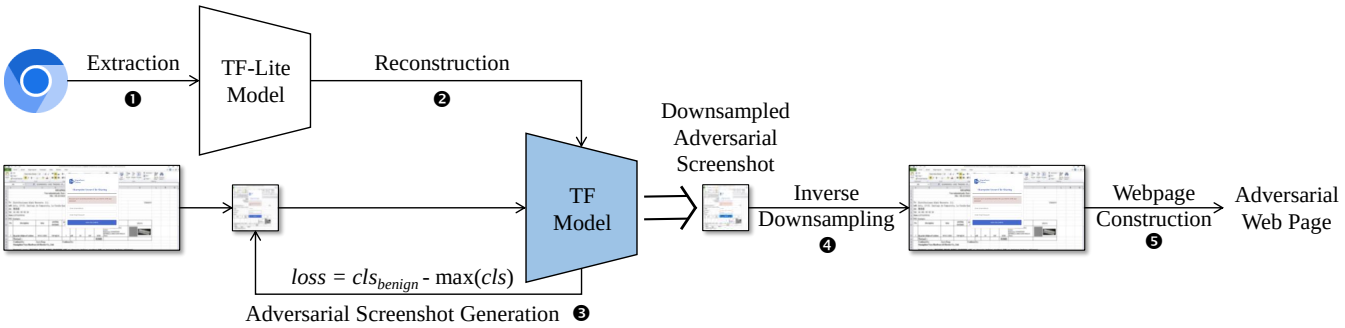


Figure 5: The workflow of our method.

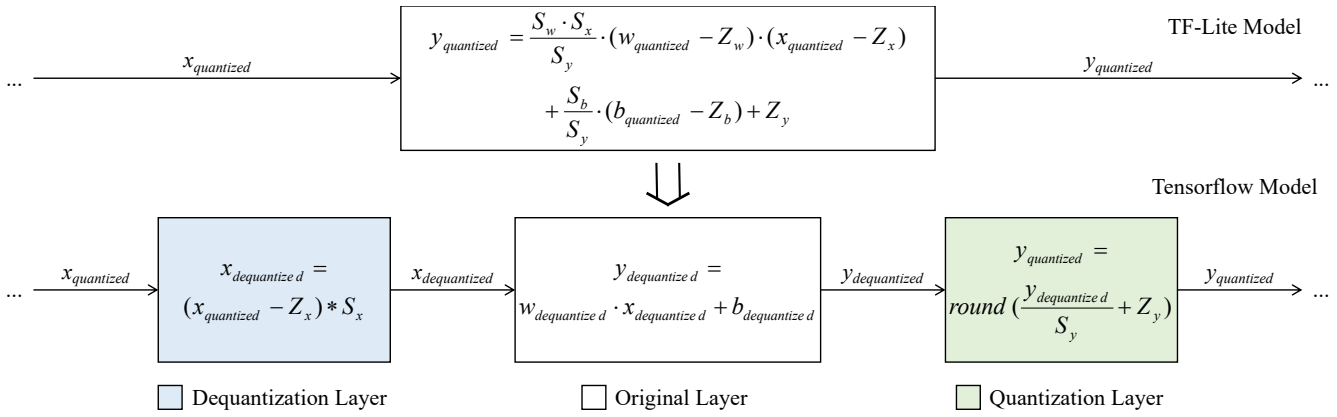


Figure 6: The method for adding quantization and dequantization layers.

"classification", etc.), and add these functions to a candidate list. Then, we recompile and run Chromium in a debug mode and set breakpoints on the candidate functions. We then visit some phishing webpages and collect the execution traces. After that, we analyze the execution traces to identify the functions that are invoked in each triggering of the block pages.

Using the above method, we locate the function that contains the operation of loading the model, i.e., *CreateClassifier()*. It takes as input a byte stream and outputs a *tflite::task::vision::ImageClassifier* object. The byte stream originates from a file in the format of a *TF-Lite model*, and the constructed classifier is used to generate the classification score of the webpage. Therefore, we can confirm that the byte stream is the model that we want to extract.

Extracting Model Data. After locating the function that loads the phishing image classifier model data, there are two ways to extract the model: reading from the disk file or dumping from the runtime memory. We choose the latter as the model file saved on the disk might not be identical to the model that performs the classification. There might be modifications after loading, and Google might update the model from the internet. We resort to debugger to dump the model data from the memory, whose corresponding variable in source code is always in the nearest position to code that performs classification.

The extracted model has a size of 3.05 MB. Using the official API [2], we can obtain the metadata of the model and acquire the knowledge about the network as mentioned in Section 2. The model takes a 224*224*3 image as input and outputs a 19-dimensional vector. Note that, the TF-Lite model is optimized with full integer quantization, meaning that all tensors in this model have been quantized into integers, including the input and output tensors.

3.2 Model Reconstruction

We have obtained a TF-Lite model suitable for classification but not for evasion. TF-Lite models are lightweight models optimized to save storage and computing resources, enabling their use on mobile, embedded, and edge devices. To this end, TF-Lite models discard most functions irrelevant to classification.

However, the most common approach for adversarial attacks against image classifiers is based on the gradient from the output layer to the input layer [16, 24, 38, 44]. Unfortunately, as a model designed for low-resource devices, TF-Lite does not support gradient calculations, making it impossible to implement gradient-based evasion attacks. To address this problem, we convert the TF-Lite model into a *TensorFlow* model with full gradient access.

The most influential difference between TF-Lite and TensorFlow models in terms of evasion is *quantization*. In the TF-Lite model, tensors are in the form of unsigned 8-bit integers, and the neural network calculations are specially designed for integers. In contrast, in TensorFlow, tensors and calculations are performed using signed

32-bit floating-point numbers. The quantization of TF-Lite reduces the model size by 75%, decreases the computational resources required for inference calculations by reducing necessary memory bandwidth, and provides better adaptability for devices that cannot efficiently perform floating-point operations. The precision difference between the two kinds of models could lead to differences in classification results, which can in turn cause evasion failure.

To address the difference, we add a *Quantization* layer and a *Dequantization* layer for each hidden layer in the model. Figure 6 illustrates the layers. A dequantization layer processes quantized 8-bit integers and passes dequantized 32-bit floating-point numbers to a standard TensorFlow hidden layer. The hidden layer performs calculations on floating-point numbers and passes the result to a quantization layer, which converts a 32-bit floats to an 8-bit integer. Quantization and dequantization are shown in Equations 1 and 2, respectively. R represents the real value calculated in the convolutional and other hidden layers. S is the quantization scale to zoom the data into a specific range, and Z is the quantized value Q corresponding to the real value 0. In the output layer, the output result is an integer between $[0, 255]$. We divide it by 256 to obtain its corresponding softmax result between $[0, 1]$.

$$Q = round(\frac{R}{S} + Z) \tag{1}$$

$$R = (Q - Z) * S \tag{2}$$

As an example, if a tensor dimension with a value of $255(Q)$ is passed into the dequantization layer before the convolutional layer, and the scale and zero point values are $0.007874(S)$ and $128(Z)$, respectively, then the dequantized value for this dimension is:

$$R = (255 - 128) * 0.007874 = 0.999998$$

If an output value of a convolutional layer is $2.3653743(R)$, and this value is passed into the quantization layer, where the scale and zero point values are $0.405654(S)$ and $131(Z)$, then the quantized value for this value is:

$$Q = \frac{2.3653743}{0.405654} + 131 = 136.831$$

The value is rounded to the nearest integer, i.e., 137.

It is also worth noting that, the *upsampling* procedure in Figure 4 is implemented in the reconstructed model. A resize layer is added into the model before the first dequantization layer. The resize layer uses bilinear interpolation to upsample a 48*48 px image to a 224*224 px image, the same as done in Chromium.

A natural concern is that the reconstructed model may not reproduce the results of the TF-Lite model with sufficient precision. However, we conduct a pilot experiment to demonstrate that our reconstruction has adequate precision. Our experiment uses 20 webpages: 10 ordinary non-phishing pages and 10 phishing pages. We downsample the 20 full-size screenshots to 48*48 px images and then feed them into the TF-Lite model (after a bilinear interpolation) and the reconstructed TensorFlow model, respectively. The classification results are then compared. The difference between the results for two models is not significant and does not influence the phishing detection outcome. In fact, both TF-Lite model and our TensorFlow model can correctly classify all the 20 webpages. The pilot study demonstrates that our reconstruction has acceptable classification performance.

Algorithm 1: Adversarial Screenshot Generation

Input: Original downsampled screenshot x
Output: Adversarial downsampled screenshot x' or failure

$iter\_count, last\_expand\_count \leftarrow 0$;
$saliency, score \leftarrow$ get_score(x) {
  $sal \leftarrow$ calculate_saliency(x);
  $cont\_by\_channel \leftarrow$ calculate_contribution(x);
  $contri \leftarrow \{\sum_{c=r,g,b} cont\_by\_channel[i][c] \mid \forall i \in x\}$;
  $sal, contrib \leftarrow$ normalize($sal, contrib$);
  $sco \leftarrow \{contrib[i] - sal[i] \mid \forall i \in x\}$;
  return sal, sco; };
$current\_mods \leftarrow$ get_topk($x, score, k * |score > 0|$) ;
$upper, lower \leftarrow$ get_inital_limits($H, saliency$);
$relaxed \leftarrow$False; $M \leftarrow \emptyset$;
while $iter\_count \leq max\_iter$ **do**
  if $relaxed$ **then**
    $upper, lower \leftarrow$ relax_limits($upper, lower$) ;
    $relaxed \leftarrow$ False;
  end
  $cls \leftarrow$ model(x); $loss \leftarrow cls_{benign} - \max(cls)$;
  $D(g_x) \leftarrow$ calculate_degree($loss, model, x$);
  $x \leftarrow$ modify($x, D(g_x), current\_mods$);
  $x \leftarrow$ clip($x, upper, lower$);
  if model(x) $< thresh$ **then**
    return current x as x';
  end
  $iter\_count$++; $last\_expand\_count$++;
  if count_mods(x) $< clip\_ratio \times |current\_mods|$ **then**
    $last\_expand\_count \leftarrow 0$;
    $\_, score \leftarrow$ get_score(x);
    $M \leftarrow M \cup current\_mods$;
    $current\_mods \leftarrow$ get_topk($x - M, score, k * |score > 0|$);
    if $|M| > 100$ **then**
      $relaxed \leftarrow$True;
      $M \leftarrow \emptyset$;
    end
  end
  if $last\_expand\_count > 100$ **then**
    $last\_expand\_count \leftarrow 0$;
    $current\_mods \leftarrow$ get_topk($x, score, |current\_mods| + 10$);
  end
end
return failure message;

3.3 Adversarial Screenshot Generation

Adversarial screenshot generation performs iterations on the reconstructed TensorFlow model and generates a 48*48 px adversarial

downsampled screenshot using original downsampled screenshot. The primary goal of this step is to bypass the phishing image classifier while preserving visual utility as much as possible. There are three basic principles for preserving visual utility:

- Modified pixels should have low visual saliency.
- The number of modified pixels should be minimized.
- The degree of modification per pixel should be minimized.

To achieve the goal, we develop an algorithm to get modification candidates and modify pixels. The algorithm is shown in Algorithm 1.

3.3.1 Getting Modification Candidates. We first screen out the candidate pixels for modification. The pixels should have relatively high contribution to classification but relatively low saliency. Therefore, we compute the saliency of each pixel to reduce the likelihood of modifying high-saliency pixels. Besides, we calculate the per-pixel contribution to classification and modify only the ones with high contribution, and thus we can achieve the bypass goal with fewer modified pixels and smaller per-pixel changes. We compose the saliency and contribution to a kind of per-pixel score, based on which we select the candidates. It involves two major steps.

First, we calculate the score for each pixel using an inner function *get_score()* at line 2 in Algorithm 1, the details of which span from line 3 to line 8. The function also returns the saliency for subsequent use. We use a pretrained webpage saliency calculation model [43] to calculate each pixel's saliency (line 3), and the well-known Guided Backpropagation method [50] to calculate the classification contribution for each pixel channel (line 4). The absolute values in all channels are summed up to calculate the pixel's contribution (line 5). After obtaining the saliency and contribution maps, we normalize them to the range of $[0, 1]$ (line 6), and calculate the final score by subtracting the saliency from the contribution (line 7). As a result, a pixel with a high score should have relatively low saliency but high contribution to classification. Prioritizing the high-score pixels for modification would satisfy the first two principles.

Then, we sort the pixels by their scores and select the top k pixels as modification candidates (line 9), denoted as *current_mods*. The value of k is related to the number of pixels with positive scores. If a pixel has a positive score, it means that the contribution loss is greater than the saliency loss by modifying the pixel. In other words, the pixel is worth modifying. In practice, users can select a proper k to limit the number of modified pixels, in case the number becomes too large.

3.3.2 Modifying Pixels. We then calculate the limits of pixel modifications (i.e., the maximum gap between current and modified pixel values), modifying pixel values of the top k pixels, and use the upper and lower limits to constrain the modification to satisfy the last principle for preserving visual utility. The task may take multiple rounds of modifications and involves lines 10-39 in Algorithm 1.

Calculating Pixel Modification Limits. First, we calculate the limits for pixel modifications (line 10). The initial upper and lower limits for each pixel are empirically calculated using Equation 3.

$$\begin{aligned} &upper = \min(i+1, 255),\ lower = \max(i-2, 0); \\ &\quad \textbf{if } i \in H; \\ &upper = \min(i+2, 255),\ lower = \max(i-4, 0); \\ &\quad \textbf{if } i \notin H \textbf{ and } saliency[i] > 0.5; \\ &upper = \min(i+3, 255),\ lower = \max(i-6, 0); \\ &\quad \textbf{if } i \notin H \textbf{ and } saliency[i] \leq 0.5; \end{aligned} \tag{3}$$

where H is the pixel set chosen by attackers, representing the pixels that users are considered to look at from the attacker's point of view, e.g., the pixels in the login button.

According to H and the saliency, we divide the pixels into three categories at different restriction levels. The most restrictive category limits pixels that users has a high probability of seeing. The relatively restrictive category represents salient pixels users may glance at, such as a company logo. The least restrictive category contains pixels that most people will ignore. As we know, the larger a pixel's value is, the brighter and more salient it becomes. Therefore, we set stricter upper limits than lower limits, i.e., the scope of increasing pixel values is narrower than decreasing.

Lines 13-16 relax the upper and lower limits if the *relaxed* flag is True. In this study, a relaxation looses the modification limit. If the initial limit is $i \pm n$, then after m times of relaxation, the limit becomes $i \pm mn$. The relaxation happens when the iteration reaches a dead end. If the number of modified pixels expands beyond a certain amount (in our case, 100), the limit is relaxed (lines 30-33). The relaxation is performed to prevent situations where simply expanding the number of modified pixels cannot achieve a successful bypass.

Calculating Gradient and Modification Degree. We then classify the sample and construct the loss for gradient calculation (line 17). The goal of the loss is to decrease the classification scores for phishing classes until they are lower than the thresholds, above which the browser detects a phishing page. However, incorporating all phishing classes in the loss function makes it too complex for gradient calculation, making the bypass less likely to succeed. As a result, we simplify the process by using the maximum class score instead of the combination of the 18 phishing class scores, shown in Equation 4, in which *cls_benign* denotes the benign score and $\max(cls)$ indicates the maximum score among the 18 phishing classes.

$$loss = cls_{benign} - \max(cls) \tag{4}$$

There is an issue with this simplification, as the maximum class score could change during the iteration, causing the scores of the two phishing categories to rise alternately. However, the class with the maximum score has never changed before the bypass succeeded, so we can consider this simplification valid for our method.

Once the loss is acquired, we can calculate the degree of modification based on the gradient of the loss to the input (line 18). We want the output to move in the direction that the loss increases, so that we can modify the input in the positive direction of the gradient. Therefore, we use the gradient to calculate the degree of modification as shown in Equation 5.

$$D(g_x) = \frac{sign(g_x) * k_g * |g_x|}{\max(\max(g_{sign}), 0) + \epsilon}, \quad where \quad g_{sign} = \{|g_k| \quad | \quad sign(g_k) = sign(g_x)\} \tag{5}$$

In Equation 5, g_x is the pixel's gradient value, k_g is a predefined positive coefficient, and ϵ is a small positive constant to ensure the denominator is not equal to 0. With this formula, we can map the positive gradient to the interval $(0, k_g]$, map the negative gradient to the interval $[-k_g, 0)$, and leave 0 in the gradient unchanged. By adjusting k_g, we can control the step size, ensuring that the iteration will not miss the optimal point because of a too large step or get stuck in a local optimum because of a too small step.

Modifying and Clipping Pixels. After calculating the modification degree, the modification is rounded to an integer and added to the screenshot (line 19). Then, the pixel values are clipped within the upper and lower limits (line 20). After that, a series of checks are performed to complete the iteration or expand the candidates.

First, if the modified sample can bypass the phishing image classifier, it is outputted as the adversarial downsampled screenshot (lines 21-23).

Second, if the modified sample cannot bypass the phishing image classifier, we check whether most of modifications in current iteration have been clipped by the upper and lower limits (line 25). This check compares the sample before modification in this iteration (line 19) and the sample after clipping (line 20). Function *count_mods()* counts the number of modified pixels. If the number is smaller than a certain ratio of the candidate pixels in this iteration, it means that most of the candidates have reached the limit, causing further modifications to be clipped by the boundary. In this case, we recalculate the score (line 27), select candidates from pixels that have never been modified (i.e., $x - M$ at line 29), and relax current upper and lower limits if number of modified pixels have exceeded 100 since the initialization or last limit relaxation (lines 30-33).

Third, if there have been over 100 iterations of modifications for current candidates, we believe there is a high risk that the modification with current candidates has got stuck in a local optimum. We address the issue by expanding the candidates by a relatively small amount (ten in our practice, line 37).

The overall number of iterations is constrained by *max_iter* at line 12. Exceeding the iteration boundary means that there may not exist an adversarial screenshot for the original downsampled screenshot, and thus the algorithm returns a failure message (line 40).

3.4 Inverse Downsampling

Inverse downsampling accepts a constructed adversarial downsampled screenshot as input and outputs a full-size phishing webpage screenshot capable of bypassing phishing detection. Efficiency is the primary goal while the visual utility is to be preserved.

A straightforward approach to implementing the evasion is to generate the full-size adversarial screenshot directly based on the classification score. However, this end-to-end implementation encounters the combinatorial explosion problem. The task of constructing a full-size adversarial screenshot can be divided into two parts: constructing an adversarial downsampled screenshot, and inversely downsampling it to obtain the full-sized adversarial screenshot. Constructing an adversarial downsampled screenshot by modifying 100 pixels has a search space of $O(256^{100*3})$, and inverse downsampling into a 1920*1080 screenshot has a search space of $O(256^{100*23*40*3})$, then the end-to-end adversarial screenshot generation has a search space of $O(256^{100*3} * 256^{100*23*40*3})$, resulting in an extreme increment of time consumption for bypassing.

To address the problem, we separate the large iteration into two stages: selecting the local optimum downsized screenshot and applying it to the inverse downsampling process. By doing so, we can reduce the time complexity to $O(256^{100*3} + 256^{100*23*40*3})$, which shows a significant decrement from the end-to-end approach.

Note that, since the output of the previous step, i.e., a 48*48px image, is taken as input, we can only achieve the local optimum in visual utility within the constraints of the 48*48px downsampled image. We cannot achieve the global optimum in the entire search space of the full-size screenshot. To address this issue, we introduce several techniques to implement fine-tuned visual utility preservation. Although we still cannot achieve the global optimum, the visual utility of the webpage remains at an acceptable level.

In our design, we convert the inverse downsampling problem into an integer linear programming problem and leverage the *GUROBI* optimizer [1] to solve the problem. Any feasible solution to this problem represents a viable bypass of the phishing image classifier. To accomplish this, we treat the pixels in the full-size original image, which can influence the altered pixels in the downsampled adversarial screenshot, as variables. We constrain the integer linear programming problem to ensure that the full-sized screenshot, after inverse downsampling, can be downsampled to the output of the adversarial screenshot generation. We incorporate indicators to assess visual utility in the objective function. By minimizing the value of the objective function, we can preserve the visual utility in the inverse downsampling result as much as possible.

The transformed integer linear programming problem can be expressed as Equation 6. $C_1^{\mathbf{T}}$ and $C_2^{\mathbf{T}}$ represent weights for different modification distances. $K^{\mathbf{T}}$ denotes the downsampling kernel. $P_{fullsized}$ is the pixel value in the full-size original screenshot, and $P_{downsized}$ is the pixel value in the downsampled adversarial screenshot. M, N and U, V are two pairs of variable vectors designed to measure the absolute modification distance. Since absolute value operations are not permitted in linear programming problems, we use a pair of variables to represent the *distance* between a variable and a constant. The details will be discussed as follows.

$$\begin{aligned} \min \quad & Z = C_1^{\mathbf{T}} \cdot (U + V) + C_2^{\mathbf{T}} \cdot (M + N) \\ s.t. \quad & 0 \le (U - V) + P_{fullsized} \le 255 \\ & 0 \le U, V, M, N \le 255, \quad U, V, M, N : Integer \\ & (U - V) + P_{origin} = (M - N) + P_{blurred} \\ & P_{downsized} = K^{\mathbf{T}} \cdot (U - V + P_{fullsized}) \\ & \delta \le \textstyle\sum_c (U_c + V_c) \quad c = r, g, b \end{aligned} \tag{6}$$

Variable choice. Figure 7 shows an example of choosing proper pixels as variables. To optimize the visual utility of the modified full-size image, we employ a trick to change the shape of the modified zone from a rectangle (the shape of the downsampling kernel) to the inscribed ellipse of the original rectangular range. For those pixels

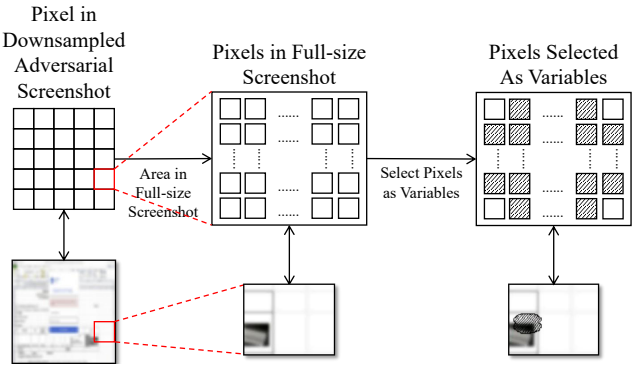


Figure 7: A schematic diagram about how we choose variable that in an ellipse shape with real examples.

not within the ellipse but inside the original rectangular range, we treat their values as constants. By implementing this trick, we can make the shape of the modified zone resemble a dripping stain on the monitor.

Objective function. The objective function of the integer linear programming problem comprises two parts. One limits the distance to the original pixel value, and the other limits the distance to the pixel values around its positions, as illustrated in Figure 8.

Suppose that X is a variable and P_{origin} is the corresponding pixel value in the full-size original screenshot. We use the combination of U, V (as shown in Equation 7) to denote the distance between the variable and its original pixel (see Figure 8(a)). The distance, with a coefficient $C_1^{\mathbf{T}}$, is minimized as a partial objective. To measure the distance to pixels around, we blur the original image, i.e., averaging the pixels around as the pixel value as shown in Figure 8(b). The blurred pixel has a value $P_{blurred}$ and then the combination of M, N in Equation 7 can represent the distance between the variable and the surrounding pixels. The distance, accompanying with a coefficient $C_2^{\mathbf{T}}$, becomes the other part of the objective function.

$$\begin{aligned} U &= \frac{|X-P_{origin}|+(X-P_{origin})}{2} \\ V &= \frac{|X-P_{origin}|-(X-P_{origin})}{2} \\ M &= \frac{|X-P_{blurred}|+(X-P_{blurred})}{2} \\ N &= \frac{|X-P_{blurred}|-(X-P_{blurred})}{2} \end{aligned} \tag{7}$$

We take the full-size blurred image into consideration for the following reason. According to our observations, the visual saliency for a modified pixel is related not only to its distance to the original pixel value but also to its distance to the surrounding pixels.

Note that, for a single channel of a pixel, $C_1^{\mathbf{T}}$ is greater than $C_2^{\mathbf{T}}$. However, the ratio between $C_1^{\mathbf{T}}$ and $C_2^{\mathbf{T}}$ depends on the value of the channel. In our design, if a channel value is larger than 128, then the ratio is smaller than the situation where the value of the channel is smaller than 128. This method is based on the observation that darker colors (with channel values smaller than 128) can be hidden more easily within their surroundings than brighter colors (with values larger than 128).

Constraint. The constraint of the integer linear programming problem consists of three parts.

First, the variable value, denoted as $U - V + P_{fullsized}$ in Equation 6, should be legal. As a result, U, V, M, N are all integers within $[0, 255]$. Additionally, the variable should remain consistent during the problem solving, i.e., $(U - V) + P_{origin} = (M - N) + P_{blurred}$.

Second, we must ensure that the constructed full-size screenshot can be downsampled to the input, i.e., the adversarial downsampled image. As such, we constrain that the value of a variable multiplying the downsample kernel $K^{\mathbf{T}}$ should be equal to the value of the corresponding input pixel.

Third, we employ a trick to enhance the visual utility of the constructed full-size screenshot, i.e., the sum of all channel distances to the original pixel must be larger than a predefined δ. According to our observation, a small change in the downsampled pixel can be achieved through a drastic alteration of a small number of pixels and negligible alteration of the other related pixels in the full-size screenshot, which compromises visual utility, as illustrated in the top part of Figure 9. To unsure all pixels in the full-size screenshot related to a changed pixel in the downsampled screenshot are evenly altered as far as possible to preserve visual utility, we constrain that each pixel should have at least a minimum change, as demonstrated in the bottom part of Figure 9. By this means, drastic pixel changes will be forbidden, because the overall change is constrained by the validity of the related modified pixel in the downsampled screenshot.

In conclusion, the proposed inverse downsampling generates a full-size adversarial screenshot within a reasonable amount of time, while simultaneously preserving the visual utility of the adversarial screenshot.

3.5 Webpage Construction

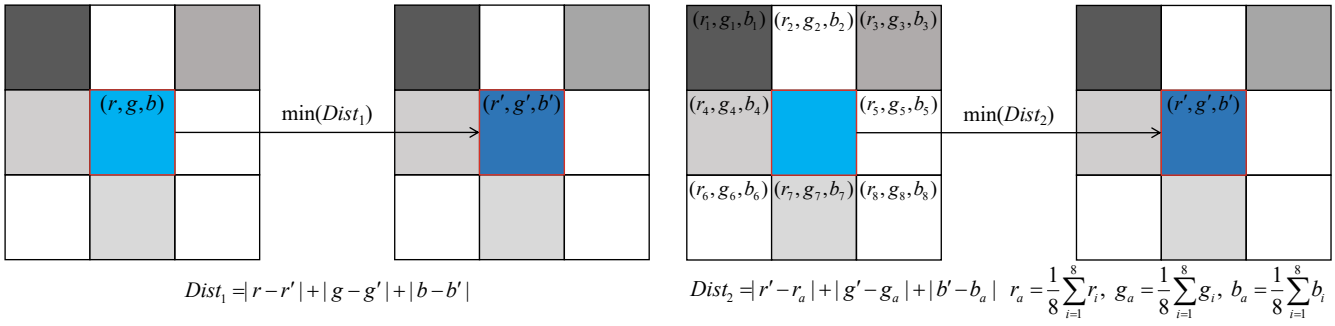
This step takes full-size adversarial screenshot as input and outputs a fully-functional webpage that can bypass Google's phishing detector. The key is to introduce the perturbations in a way that does not affect the normal function of the target webpage.

There are many leverageable web front-end tricks to launch an effective attack. In this study, we utilize the widely-used CSS technique to construct the phishing webpage straightforwardly.

We convert the full-size adversarial screenshot into an SVG image with the same resolution as the original webpage. Note that, only the modified pixels are kept, while the others are removed and left transparent in the SVG image. Besides, if the adversarial perturbation involves input fields in the webpage, the corresponding areas are cut out of the full-size SVG and saved alone as small PNG images. The left half of Figure 10 shows the conversion.

The constructed phishing webpage consists of three layers and the CSS property "z-index" is utilized to place the layers. The bigger the z-index, the upper the layer. An illustration of the webpage construction is shown in the right half of Figure 10.

The top layer contains the perturbed input fields, such as the input box for user ID in Figure 10. It has the biggest z-index to ensure the user input will not be blocked. We adjust the box's CSS property "background" and set the corresponding small PNG image as its background. Furthermore, we adjust the small PNG's position-related CSS properties like "top" and "left" to incorporate it with the input field seamlessly.



(a) Distance with pixel in corresponding position

(b) Distance with pixels around corresponding position

Figure 8: The two kinds of distance we use to construct the objective function.

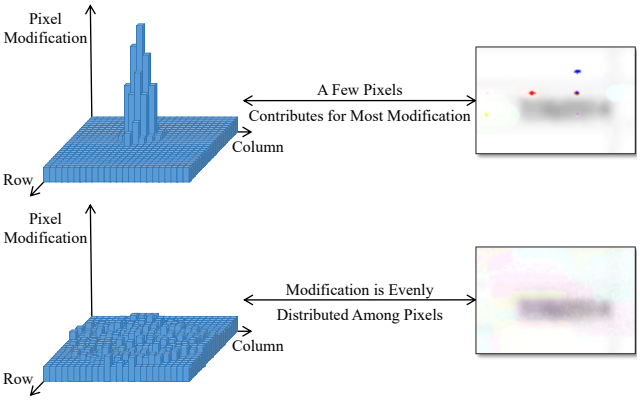


Figure 9: The constraint design to evenly spread the contribution.

The middle layer contains the full-size SVG image. We set it as the background of a *div* element, and align the *div* element with the whole page. At the same time, we set the "pointer-event" property of the *div* element to *none*, so that any click operations can go through the image and reach the bottom layer, where all elements reside except the input fields.

Consequently, the functionality of the webpage remains unchanged. The three-layer structure ensures that user inputs are correctly displayed without being obscured and the clicks can reach the underlying target elements. In the meantime, all the perturbations have been introduced in the form of small background images of input fields at the top layer and full-size background image at the middle layer.

For some webpages, we must also bypass the logistic regression classifier simultaneously. To address this, we apply the method presented in [35], inserting invisible *Good Terms* into the webpage. The terms can effectively decrease the classification score to a safe range under the threshold of phishing detection. In all of our 135 samples, only 20 requires to bypass the logistic regression classifier.

It should be pointed out that, the primary objective of this study is to demonstrate the feasibility of attacking the phishing webpage image classifier. In practice, attackers could employ more sophisticated techniques to realize the attacks.

4 EVASION EVALUATION

We investigate the following research questions to evaluate the proposed evasion method.

RQ1: Can our approach effectively bypass Google's phishing webpage image classifier? (Section 4.1)

RQ2: Can our approach preserve the visual utility of the target webpages? (Section 4.2)

RQ3: How much time is consumed to generate adversarial image? (Section 4.3)

4.1 Effectiveness

We did an end-to-end testing of our attack method. To prevent the leakage of attack samples, we set up a local web server to host the attack samples. However, Chrome/Chromium does not perform classification detection on pages with local addresses. Therefore, we disabled Chromium's local address detection in the source code and took it as the evaluation target.

To evaluate the evasion ability of our approach, we collect phishing webpage samples from *PhishTank* [53] and *OpenPhish* [39]. In total, we have collected 50 and 85 samples, respectively, from the two sources. All the obtained pages are identified as phishing webpages by Chromium's phishing image classifier. Although the number is limited, the samples target login pages of different popular web sites, including Paypal, Outlook, Netflix, etc.

Following the implementation of our bypass method, we found that each of the 135 pages can be dressed up to successfully bypass Google's phishing image classifier detection. Additional details and results can be found in Table 1, in which the *ID* columns indicate the sample ID in PhishTank or sequential ID in OpenPhish, the *Category* columns show the attack targets, and the *Bypass* columns show the evasion result against the image classifier.

4.2 Visual Utility Preservation

In this study, we leverage two methods to evaluate the effectiveness of visual utility preservation.

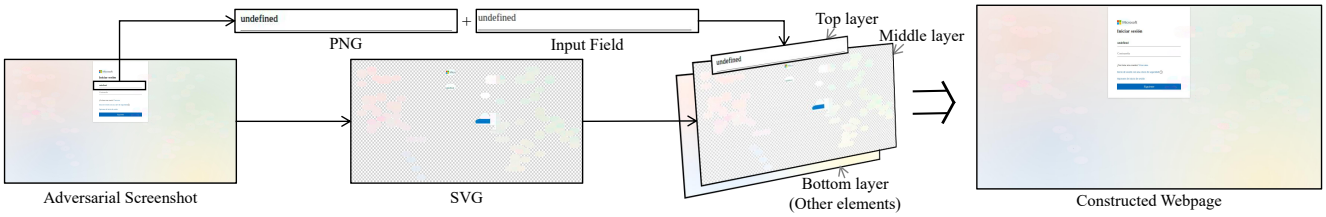


Figure 10: Webpage Construction.

Table 1: Experiment results (time unit: minutes)

ID	Category	Bypass	AUC	Time	ID	Category	Bypass	AUC	Time	ID	Category	Bypass	AUC	Time
8120157	Outlook	✓	0.996	5.0	7945622	Shared Document	✓	0.995	16.5	op041	Shared Document	✓	0.963	6.1
8101033	Paypal	✓	0.996	2.4	7973561	Microsoft	✓	0.85	6.5	op042	Netflix	✓	0.994	11.4
8100905	Microsoft	✓	0.958	23.9	7969583	Shared Document	✓	0.994	4.3	op043	Microsoft	✓	0.959	3.7
8082292	Microsoft	✓	0.732	2.1	7958598	Netflix	✓	0.995	8.5	op044	Microsoft	✓	0.904	5.0
8082216	Facebook	✓	0.989	13.8	7897335	Shared Document	✓	0.988	4.1	op045	Microsoft	✓	0.998	1.8
8082078	Netflix	✓	0.998	19.8	op001	Microsoft	✓	0.946	3.0	op046	Microsoft	✓	0.998	1.9
8080988	Microsoft	✓	0.953	4.6	op002	Microsoft	✓	0.973	3.3	op047	Facebook	✓	0.997	0.4
8080214	Microsoft	✓	0.905	14.6	op003	Microsoft	✓	0.966	3.5	op048	Netflix	✓	0.993	53.8
8078574	Paypal	✓	0.996	2.9	op004	Microsoft	✓	0.977	14.5	op049	Outlook	✓	0.946	6.9
8052357	Outlook	✓	0.949	6.0	op005	Microsoft	✓	0.964	11.6	op050	Facebook	✓	0.998	7.5
8052355	Shared Document	✓	0.987	33.7	op006	Shared Document	✓	0.993	3.1	op051	Facebook	✓	0.998	7.4
8052333	Microsoft	✓	0.963	4.7	op007	Microsoft	✓	0.974	22.0	op052	Microsoft	✓	0.968	5.6
8042311	Microsoft	✓	0.964	3.9	op008	Microsoft	✓	0.9	87.8	op053	Outlook	✓	0.963	7.0
8042296	Netflix	✓	0.997	44.1	op009	Microsoft	✓	0.95	8.6	op054	Microsoft	✓	0.879	9.0
8042295	Netflix	✓	0.996	21.8	op010	Microsoft	✓	0.923	44.7	op055	Outlook	✓	0.963	6.0
8042290	Microsoft	✓	0.809	44.3	op011	Microsoft	✓	0.933	7.7	op056	Netflix	✓	0.998	11.3
8042289	Microsoft	✓	0.843	8.1	op012	Microsoft	✓	0.915	3.3	op057	Microsoft	✓	0.952	2.4
8040774	Paypal	✓	0.984	8.1	op013	Microsoft	✓	0.977	6.1	op058	Facebook	✓	0.997	5.5
8040771	Outlook	✓	0.952	6.2	op014	Facebook	✓	0.992	9.5	op059	Microsoft	✓	0.966	7.0
8040761	Shared Document	✓	0.987	32.0	op015	Microsoft	✓	0.958	1.4	op060	Microsoft	✓	0.944	9.5
8040752	Microsoft	✓	0.954	5.8	op016	Microsoft	✓	0.933	4.5	op061	Netflix	✓	0.998	12.4
8040698	Facebook	✓	0.987	12.0	op017	Microsoft	✓	0.939	6.8	op062	Microsoft	✓	0.950	11.2
8039424	Microsoft	✓	0.862	12.4	op018	Microsoft	✓	0.955	6.4	op063	Microsoft	✓	0.975	2.2
8039340	Microsoft	✓	0.956	7.0	op019	Microsoft	✓	0.956	5.5	op064	Shared Document	✓	0.998	3.7
7998977	Microsoft	✓	0.957	2.4	op020	Microsoft	✓	0.734	5.9	op065	Amazon	✓	0.993	14.7
7989793	Shared Document	✓	0.983	8.4	op021	Microsoft	✓	0.949	6.1	op066	Facebook	✓	0.997	6.2
7989787	Netflix	✓	0.997	46.8	op022	Microsoft	✓	0.997	6.0	op067	Microsoft	✓	0.965	17.5
7989781	Microsoft	✓	0.922	4.2	op023	Netflix	✓	0.992	20.4	op068	Netflix	✓	0.997	11.7
7989779	Microsoft	✓	0.939	5.7	op024	Microsoft	✓	0.954	5.6	op069	Microsoft	✓	0.965	7.7
7989777	Microsoft	✓	0.826	7.5	op025	Netflix	✓	0.992	50.6	op070	Amazon	✓	0.983	6.9
7984500	Microsoft	✓	0.955	19.8	op026	Outlook	✓	0.972	7.2	op071	Amazon	✓	0.993	8.5
7984484	Microsoft	✓	0.974	4.2	op027	Microsoft	✓	0.97	3.7	op072	Microsoft	✓	0.848	5.2
7983855	Outlook	✓	0.959	6.3	op028	Microsoft	✓	0.973	3.7	op073	Facebook	✓	0.998	8.2
7983589	Netflix	✓	0.997	7.9	op029	Microsoft	✓	0.965	3.2	op074	Microsoft	✓	0.960	6.4
7983585	Shared Document	✓	0.982	71.1	op030	Outlook	✓	0.965	6.7	op075	Microsoft	✓	0.972	0.9
7983549	Microsoft	✓	0.918	3.5	op031	Microsoft	✓	0.96	3.3	op076	Facebook	✓	0.987	7.5
7983503	Microsoft	✓	0.929	3.7	op032	Shared Document	✓	0.975	13.1	op077	Outlook	✓	0.983	6.8
7983245	Microsoft	✓	0.979	16.8	op033	Facebook	✓	0.995	6.1	op078	Facebook	✓	0.987	8.5
7982619	Shared Document	✓	0.992	3.3	op034	Microsoft	✓	0.926	4.1	op079	Facebook	✓	0.973	4.0
7981210	Microsoft	✓	0.899	3.3	op035	Facebook	✓	0.991	4.7	op080	Microsoft	✓	0.964	14.3
7949021	Microsoft	✓	0.903	7.5	op036	Facebook	✓	0.998	6.2	op081	Microsoft	✓	0.971	3.4
7949019	Microsoft	✓	0.968	7.0	op037	Microsoft	✓	0.947	5.3	op082	Outlook	✓	0.970	9.8
7949016	Microsoft	✓	0.997	6.4	op038	Netflix	✓	0.998	12.4	op083	Microsoft	✓	0.985	4.9
7948992	Shared Document	✓	0.984	8.2	op039	Microsoft	✓	0.907	23.3	op084	Outlook	✓	0.964	5.1
7948941	Microsoft	✓	0.850	6.6	op040	Microsoft	✓	0.971	3.6	op085	Outlook	✓	0.963	5.2
Average													**0.959**	**10.7**

Quantitative analysis. We perform a quantitative analysis to ascertain whether perturbations possess the capability to obviously redirect the attention of users interacting with a given webpage. The underlying premise is that if there is a marked shift in attentional focus, users would be more inclined to perceive pronounced anomalies within the webpage. Consequently, this would indicate a failure in preserving the visual utility. If no such shift is observed, it is construed that the visual utility remains intact.

Webpage saliency, as described in [45], serves as a tool to pinpoint potential areas capturing users' attention. To holistically assess the changes in page-wide saliency, the widely-acknowledged metric of AUC (Area Under the ROC Curve) [28, 45] is employed. A higher AUC score is indicative of reduced changes in saliency. An AUC score equating to one denotes the absence of saliency shifts.

The computation of the AUC score necessitates two distinct saliency maps. As delineated in webpage saliency prediction [45], the duo comprises a ground truth map and a target map. In our study, the ground truth is represented by the saliency map of the original page, while the adversarial page's saliency map forms the target version. Our approach to calculating AUC aligns with the methodology proposed in [45]. First, the ground truth map is transformed into a discrete map m_g, with all values exceeding zero set to one; zeroes remain unaffected. Subsequently, for each unit position in m_g with a value of one, we extract the corresponding value v in the target map m_t. Utilizing v as a binary classifier, values in m_t exceeding v are set to one and the others are set to zero, yielding a classified map m_c. We then compute both the *True Positive Rate (TPR)* and *False Positive Rate (FPR)* based on m_g and m_c. Finally, the ROC curve, using TPRs and FPRs for all binary classifiers, is plotted, and the AUC is calculated.

A threshold is needed to be established for the AUC score. Only scores exceeding the threshold would signify the absence of overt saliency shifts on the given webpage. Relying on data from [45], a consistent AUC score of approximately 0.7 was attained across diverse conditions, aligning the target saliency map closely with the ground truth. Therefore, we adopt 0.7 as our evaluative threshold for whole-page saliency alterations.

Our experiment yielded AUC scores for all evaluation targets, as tabulated in the *AUC* columns of Table 1. The cumulative average stands at 0.959. Remarkably, out of 135 samples, 124 (91.9%) showcase AUC scores exceeding 0.9, with a mere two falling below 0.8 but above the threshold. This overarching data underscores only a little saliency deviation is caused by our adversarial perturbations, implying a minimal likelihood of users' attention shift and thereby retaining the visual utility.

User study. We also conduct a single-blind user study involving 23 computer science students from our university. They have no prior knowledge about this project. Each participant is presented with a set of 100 phishing webpages: 50 of these are adversarial pages, and the remaining 50 are their corresponding original versions. Each page is displayed for five seconds, the same as done in [45]. The pages are randomly presented, ensuring that participants are unaware of which pages have been perturbed. Participants are tasked with identifying any abnormal or incongruent regions on the displayed page. If a page is deemed to contain such perturbed regions, participants are then instructed to specify these areas. For the original samples, any misidentification as an adversarial page is counted as a false positive. For the adversarial samples, our evaluation criteria is somewhat lenient. If a participant correctly identifies any of the perturbed regions without flagging non-perturbed ones, we consider it a correct identification.

We amassed 1,700 valid feedback entries, as some participants did not look through all pages. Data from Table 2 reveals that 66.0% of the adversarial samples and 33.1% of the original ones are misidentified by the participants. In total, around half of the samples (840) are inaccurately recognized. Moreover, none of the samples achieve unanimous correct identification across all participants. The findings suggest that differentiating between the adversarial and original samples is challenging for the participants. The study demonstrates that the visual utility of adversarial samples is well preserved. It is worth noting that, given the heightened sensitivity of computer science students to such perturbations, we postulate that general users may exhibit even lower identification accuracy.

Table 2: User study result

	Original	Adversarial
Total (1,700)	858	842
Correctly Identified (860)	574 (66.9%)	286 (34.0%)
Incorrectly Identified (840)	284 (33.1%)	556 (66.0%)

We must also notice that, repeated exposure during the testing phase might have further heightened the participants' awareness of the perturbations. This might have refined their skills, especially when numerous pages were presented consecutively. However, in real-world scenarios, users are likely to encounter adversarial phishing attacks sporadically. Without the guidance provided in our study, they might not focus on or detect subtle changes, meaning that their attention might not be drawn to the perturbations. Given these considerations, we posit that our study offers a conservative yet reliable assessment.

4.3 Time Cost

The reconstructed Tensorflow model was built on Python 3.7.11 and Tensorflow 2.7.0. Adversarial screenshot generation and inverse downsampling were conducted on Windows 10 21H2, Intel(R) Core(TM) i7-10870H CPU @ 2.20GHz without the use of a GPU.

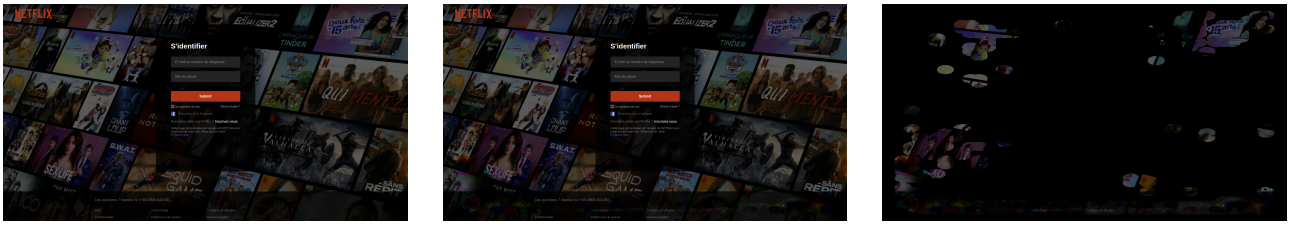
As shown in the *Time* columns in Table 1, generating one full-size adversarial screenshot, including adversarial screenshot generation and inverse downsampling, takes an average of 10.7 minutes. Most of the samples (92.6%) take no more than half one hour.

It is noteworthy that, in our study, adversarial screenshot generation is relatively fast, with an average time consumption of just 88.4 seconds. Inverse downsampling requires much more time, encompassing the majority (over 86%) of the total time consumption as we work on solving the integer linear programming problem. Despite the occasionally extreme time consumption, we consider the cost is acceptable, as the adversarial image generation is a one-time off-line effort.

4.4 Case study

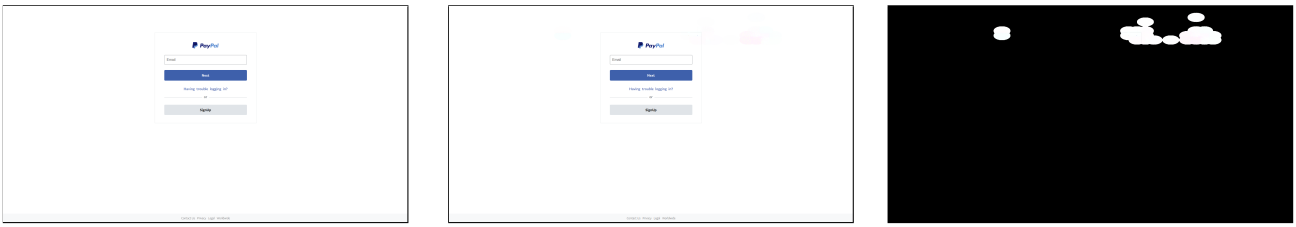
We use two cases to demonstrate the effectiveness of our approach in both evasion and utility preservation. The first shows an example with complex background and the second shows a case with very simple background.

Webpage with complex background. Some webpages use complex backgrounds, such as pictures and dense texts, which makes it easy to hide modified pixels. For example, Figure 11 shows a case that imitates the login page of Netflix. In such a case, many pixels have been modified as shown in Figure 11(c). However, the high information density in the background makes it harder for human beings



(a) Original screenshot (b) Adversarial screenshot (c) Illustration of modified pixels

Figure 11: An example phishing page with complex background.



(a) Original screenshot (b) Adversarial screenshot (c) Illustration of modified pixels

Figure 12: An example phishing page with simple plain white background and the adversarial screenshot.

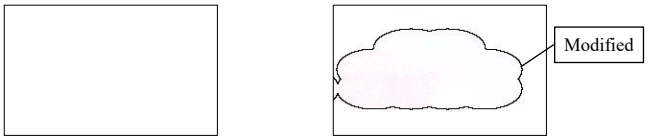


Figure 13: A detailed illustration of a modified area in Figure 12.

to identify the information from the adversarial modification.notice the changes from Figure 11(a) to Figure 11(b). Especially when Figure 11(b) alone is shown to users, it is impossible for them to identify the changed pixels that are intended to bypass the phishing page image classifier. Even if there are a few salient pixels in the adversarial screenshot, they are harder to spot or may be mistaken for dead pixels on LCD screen due to their small number.

Webpage with simple background. Some websites use simple backgrounds, such as plain white or solid color, which makes it difficult to preserve visual utility while succeeding the bypass. Figure 12(a) imitates the login page of PayPal. Such a webpage with simple background, e.g., plain white, makes it hard to preserve visual utility, since simple background has very low saliency. Any modification with salient pixels can drawn unwanted attention and web viewers would be aware of the abnormal visual effect. We successfully fit the modifications to the background, with each pixel only modified a small amount. Without a special notification, web viewers may consider the negligible modifications in Figure 12(b) as dust on the screen. The modified pixels are illustrated in Figure 12(c). Figure 13 further shows that significant changes in salient pixels are avoided, taking as an example a modified area in the original and corresponding adversarial screenshot.

5 DEFENSE

As presented above, the phishing image classifier is demonstrated to be vulnerable. Improving the robustness of the model thus becomes an urgent issue. In this section, we present two lightweight defense methods, adversarial training and noise filtering, to enhance the classifier. Note that, developing a comprehensive defense mechanism is beyond the scope of this paper, and our main aim is to explore the feasibility of defending potential evasion attacks.

5.1 Adversarial Training

Adversarial training is widely-used in the field of deep learning defense. Adversarial training takes adversarial samples as the training set to train the model, making the model more robust to the same kind of adversarial samples. Therefore, we choose to use the generated adversarial samples to conduct adversarial training on the phishing webpage classifier model in Chrome/Chromium, making it more robust against adversarial samples.

In a preliminary experiment, we randomly split the 135 adversarial samples into two parts: a training set with 50 samples and a testing set with the others. We have checked that adversarial screenshots for each type of phishing web page are included in the training set. After a quick training, the model is fortunately able to detect all the 85 phishing webpages in our testing set.

The adversarial training performs well in our experiment with only 135 samples. This is a good sign that, with enough adversarial samples, the robustness of the phishing image classifier could be further improved using adversarial training. Therefore, the phishing

image classifier should not use fixed weights for a long time. Instead, it should periodically update its weights through adversarial training.

5.2 Noise Filtering

Our method of constructing adversarial samples is to add tiny perturbations to original images. The perturbations can be seen as a kind of noise that is introduced into the images. Therefore, it is natural to use a noise filtering technique as a defense method against our bypass.

In our experiment, we use a 5*5 median filter and a 9*9 Gaussian filter to filter the introduced adversarial noise. The effect is remarkable. After the filter, 92 among the 135 samples (68.1%) can be detected by the phishing image classifier.

The noise filtering is a simple, direct yet powerful way to raise the bar, making the phishing image classifier robust to this kind of bypass with adversarial noise. In the practice, the filtering operation can be implemented by the convolutional layer with different kind of kernels, so that the source code does not need to be modified.

5.3 Summary

The two defense methods we proposed perform well in defending our bypass. Preliminary experiment shows that even the lightweight defense method could make a significant improvement in the robustness of phishing detection. However, the detection and defense of the phishing webpage is an open problem. Facing the latest defense, the attacker could always come up with new attack methods that are not able to defend with current defenses. Therefore, the defenders should always update their defense method to adapt to the latest attacks.

6 RELATED WORK

Since our work focuses on bypassing a phishing image classifier, we explore the domain of attacks against other image classification neural networks, the machine learning based approaches for detecting phishing webpages, and the attacks against machine learning based methods for detecting phishing webpages.

Attacks against Image Classification Neural Networks. Attacks against image classification neural networks primarily fall into two categories: perturbation-based attacks and generation-based attacks. Perturbation-based attacks deceive the image classifier by introducing perturbations to benign images. A notable example is [24], which alters each pixel channel by one in the direction of gradient descent. Other works, such as [17, 21, 33, 34, 42, 52, 54, 63], develop various techniques to control modification degrees and modification boundaries but do not regulate the number of modified pixels. Adversarial patch attack [16, 26, 32, 36, 38, 41, 51, 59] is another type of perturbation-based attack. It limits the number of modified pixels but does not constrain the per-pixel modification.

Generation-based attacks [15, 31, 49, 57] employ Generative Adversarial Networks (GANs) to create adversarial samples. A GAN consists of a generator and a discriminator. The former attempts to produce adversarial inputs close to benign samples, and the latter tries to differentiate the generated samples from benign ones.

Machine Learning Based Approaches for Detecting Phishing Webpages. Many methods have been developed for detecting potential phishing webpages using machine learning techniques. The most commonly used feature for detection is the URL. Many works [5, 6, 8, 14, 22, 23, 30, 48, 55, 56, 58, 60–62] utilize the URL as one of their machine learning feature sources. Other feature sources include HTML DOM tree, terms on the webpage, meta-information of the website, and so on.

Google proposed its first version of a phishing detector using a logistic regression classifier in 2010 [56]. Yi et al. [62] employed Deep Belief Networks (DBNs) and features from both the webpage and interaction flow for webpage classification. PDGAN [6] used the discriminator in a GAN to determine if a website is phishy. Apruzzese et al. [11] introduced the concept of Protective Operation Chain (POC), preventing attackers from evading phishing detectors by cracking features in the detector and manipulating a small number of features. Xiao et al. [58] applied Natural Language Processing (NLP) techniques to analyze the meaning of URLs. Recurrent Neural Networks (RNNs) are also used to implement phishing website detection[14, 23]. Somesha et al. [48] incorporated third-party information like WHOIS, Alexa, and search engines as features. Huh et al. [29] utilized search results returned from popular search engines such as Google, Bing and Yahoo as features for phishing detection. Some approaches [3, 18, 19, 37] relied on visual features to identify phishing websites. Abuadbba et al. [4] analyzed techniques used to bypass ML-based phishing detection and proposes a more resilient model based on logistic regression.

Attacks Against Machine Learning Based Approaches to Detecting Phishing Websites. While many approaches focus on detecting phishing websites using machine learning techniques, the number of works attempting to bypass these detections is relatively small. Liang et al. [35] cracked GPPF's features in Chrome and used the cracked features to evade Chrome's early version of phishing detection [56]. Shirazi et al. [46] demonstrated that even with limited knowledge of the phishing detector, attackers can still generate adversarial samples by manipulating features in the URL and webpage content. Researchers also employed Generative Adversarial Networks to create adversarial URLs for phishing webpages [7, 9, 13]. Apruzzese et al. [11] used a degree to measure the attacker's knowledge of the phishing detector, evading 13 phishing detectors in a gray box scenario. They also introduced "evasion-space" to measure the actual cost of attacks [10]. MaskDGA [47] proposed a black-box attack technique to evade character-level classifiers capable of detecting algorithmically generated domain names. Panum et al. [40] perturbed visual features of phishing websites, such as HSL (hue, saturation, light), to bypass phishing detection. Gressel et al. [27] evaded phishing detection by evaluating feature importance and perturbing them towards the benign class.

7 DISCUSSION

There are many other classifiers deployed in industry, some of which are close-source. In theory, all classifiers face varying degrees of evasion threats. However, bypassing different classifiers may involve specific technical challenges, and designing a universal bypass method is nearly impossible. Moreover, for close-source classifiers, sophisticated probe testing methods need to be designed to acquire exploitable knowledge. We plan to further explore the

security aspects of some important commercial classifiers in future research efforts.

There may exist many potential evasion variants for adversarial training in Section 5.1. Focusing on the hard-to-change parts would be an insightful idea that could bring more thorough defense. One potential approach is encoding screenshots of the hard-to-change parts of pages of interest (e.g., the login page) as retrieval vectors and recording them. When users browse webpages, it is possible to quickly determine whether the current page is similar to a particular page of interest using CBIR (Content-Based Image Retrieval) [20] techniques. Based on this, potential phishing pages can then be detected by comparing the related URLs. This can provide a defense mechanism for individual users and can be implemented as a browser extension. Besides, developing a one-time solution is infeasible, due to the ongoing evolution of attacks. Developers should closely monitor state-of-the-art attacks and implement targeted defenses accordingly.

Applications employing phishing webpage detection methods similar to Chrome/Chromium's may also face the same issue, especially the browsers utilizing the Chromium kernel. Future research could focus on evaluating the robustness of these applications.

8 CONCLUSION

We presented a critical evaluation of the new-generation Google's phishing page detector with billions of users in this paper. We found that Google upgrade its detector by introducing a phishing image classifier in Chrome/Chromium alongside the existing system. To evaluate its robustness, we conducted a targeted evasion testing and investigated corresponding defense techniques. We proposed a three-stage evasion method, which can evade the phishing image classifier with 100% success rate, and maintain high visual utility. The time cost of generating an evasion page is acceptable for an off-line process. We also proposed two lightweight defense techniques for the phishing image classifier, significantly enhanced its detection ability. Our work shows that the new-generation of Google's phishing detector is still vulnerable to targeted evasion attacks, putting billions of users at risk of phishing. Fortunately, we also demonstrated that the robustness of the phishing detector can be greatly improved by employing necessary security enhancements. We believe that all commercial machine learning systems with massive users should be subjected to critical evaluation, to reveal their weakness and strengthen them as far as possible.

ACKNOWLEDGMENTS

The authors would like to thank the reviewers for their constructive comments. The authors would also like to thank the students in Turing Class 2021 in RUC for their contribution to the user study. The work is supported in part by National Natural Science Foundation of China (NSFC) under grants 62272465, 62002361, 62272464 and U1836209, and the Fundamental Research Funds for the Central Universities and the Research Funds of Renmin University of China under grant 22XNKJ29.

REFERENCES

[1] 2023. GUROBI OPTIMIZATION. https://www.gurobi.com/.
[2] 2023. Module: tf.lite. https://www.tensorflow.org/api_docs/python/tf/lite.
[3] Sahar Abdelnabi, Katharina Krombholz, and Mario Fritz. 2020. VisualPhishNet: Zero-Day Phishing Website Detection by Visual Similarity. In *CCS '20: 2020 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, USA*. ACM, 1681–1698. https://doi.org/10.1145/3372297.3417233
[4] Alsharif Abuadbba, Shuo Wang, Mahathir Almashor, Muhammad Ejaz Ahmed, Raj Gaire, Seyit Camtepe, and Surya Nepal. 2022. Towards Web Phishing Detection Limitations and Mitigation. *CoRR* abs/2204.00985 (2022). https://doi.org/10.48550/arXiv.2204.00985 arXiv:2204.00985
[5] Moses Adebowale Akanbi, Khin T. Lwin, and M. Alamgir Hossain. 2019. Deep Learning with Convolutional Neural Network and Long Short-Term Memory for Phishing Detection. In *13th International Conference on Software, Knowledge, Information Management and Applications, SKIMA 2019, Island of Ulkulhas, Maldives*. IEEE, 1–8. https://doi.org/10.1109/SKIMA47702.2019.8982427
[6] Saad A. Al-Ahmadi, Afrah Alotaibi, and Omar Alsaleh. 2022. PDGAN: Phishing Detection With Generative Adversarial Networks. *IEEE Access* 10 (2022), 42459–42468. https://doi.org/10.1109/ACCESS.2022.3168235
[7] Ahmed Aleroud and George Karabatis. 2020. Bypassing Detection of URL-based Phishing Attacks Using Generative Adversarial Deep Neural Networks. In *Proceedings of the 6th International Workshop on Security and Privacy Analytics, New Orleans, LA, USA*. ACM, 53–60. https://doi.org/10.1145/3375708.3380315
[8] Amani Alswailem, Bashayr Alabdullah, Norah Alrumayh, and Aram Alsedrani. 2019. Detecting phishing websites using machine learning. In *2nd International Conference on Computer Applications & Information Security (ICCAIS)*. IEEE, 1–6.
[9] Hyrum S. Anderson, Jonathan Woodbridge, and Bobby Filar. 2016. DeepDGA: Adversarially-Tuned Domain Generation and Detection. In *Proceedings of the 2016 ACM Workshop on Artificial Intelligence and Security, AISec@CCS 2016, Vienna, Austria*. ACM, 13–21. https://doi.org/10.1145/2996758.2996767
[10] Giovanni Apruzzese, Mauro Conti, and Ying Yuan. 2022. SpacePhish: The Evasion-space of Adversarial Attacks against Phishing Website Detectors using Machine Learning. In *Annual Computer Security Applications Conference, ACSAC 2022, Austin, TX, USA*. ACM, 171–185. https://doi.org/10.1145/3564625.3567980
[11] Giovanni Apruzzese and V. S. Subrahmanian. 2022. Mitigating Adversarial Gray-Box Attacks Against Phishing Detectors. *CoRR* abs/2212.05380 (2022). https://doi.org/10.48550/arXiv.2212.05380 arXiv:2212.05380
[12] APWG. 2022. *Phishing Activity Trends Report, 3rd Quarter 2022*. Technical Report. https://apwg.org/.
[13] Trinh Nguyen Bac, Phan The Duy, and Van-Hau Pham. 2021. Pwdgan: Generating adversarial malicious url examples for deceiving black-box phishing website detector using gans. In *2021 IEEE International Conference on Machine Learning and Applied Network Technologies (ICMLANT)*. IEEE, 1–4.
[14] Alejandro Correa Bahnsen, Eduardo Contreras Bohorquez, Sergio Villegas, Javier Vargas, and Fabio A. González. 2017. Classifying phishing URLs using recurrent neural networks. In *2017 APWG Symposium on Electronic Crime Research, eCrime 2017, Phoenix, AZ, USA*. IEEE, 1–8. https://doi.org/10.1109/ECRIME.2017.7945048
[15] Tao Bai, Jun Zhao, Jinlin Zhu, Shoudong Han, Jiefeng Chen, Bo Li, and Alex C. Kot. 2021. AI-GAN: Attack-Inspired Generation of Adversarial Examples. In *2021 IEEE International Conference on Image Processing, ICIP 2021, Anchorage, AK, USA*. IEEE, 2543–2547. https://doi.org/10.1109/ICIP42928.2021.9506278
[16] Tom B. Brown, Dandelion Mané, Aurko Roy, Martín Abadi, and Justin Gilmer. 2017. Adversarial Patch. *CoRR* abs/1712.09665 (2017). http://arxiv.org/abs/1712.09665
[17] Nicholas Carlini and David A. Wagner. 2017. Towards Evaluating the Robustness of Neural Networks. In *2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA*. IEEE Computer Society, 39–57. https://doi.org/10.1109/SP.2017.49
[18] Teh-Chung Chen, Scott Dick, and James Miller. 2010. Detecting visually similar Web pages: Application to phishing detection. *ACM Trans. Internet Techn.* 10, 2 (2010), 5:1–5:38. https://doi.org/10.1145/1754393.1754394
[19] Kang-Leng Chiew, Ee Hung Chang, San-Nah Sze, and Wei King Tiong. 2015. Utilisation of website logo for phishing detection. *Comput. Secur.* 54 (2015), 16–26. https://doi.org/10.1016/j.cose.2015.07.006
[20] Ritendra Datta, Jia Li, and James Ze Wang. 2005. Content-based image retrieval: approaches and trends of the new age. In *Proceedings of the 7th ACM SIGMM International Workshop on Multimedia Information Retrieval, MIR 2005, Singapore*. ACM, 253–262. https://doi.org/10.1145/1101826.1101866
[21] Yinpeng Dong, Fangzhou Liao, Tianyu Pang, Hang Su, Jun Zhu, Xiaolin Hu, and Jianguo Li. 2018. Boosting Adversarial Attacks With Momentum. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA*. CVF/IEEE CS, 9185–9193. https://doi.org/10.1109/CVPR.2018.00957
[22] Muna Elsadig, Ashraf Osman Ibrahim, Shakila Basheer, Manal Abdullah Alohali, Sara Alshunaifi, Haya Alqahtani, Nihal Alharbi, and Wamda Nagmeldin. 2022. Intelligent Deep Machine Learning Cyber Phishing URL Detection Based on BERT Features Extraction. *Electronics* 11, 22 (2022), 3647.
[23] Tao Feng and Chuan Yue. 2020. Visualizing and Interpreting RNN Models in URL-based Phishing Detection. In *Proceedings of the 25th ACM Symposium on Access Control Models and Technologies, SACMAT 2020, Barcelona, Spain*. ACM, 13–24. https://doi.org/10.1145/3381991.3395602
[24] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. 2015. Explaining and Harnessing Adversarial Examples. In *3rd International Conference on Learning*

Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings. http://arxiv.org/abs/1412.6572

[25] Google. 2023. Safe-browsing in Chrome. https://transparencyreport.google.com/safe-browsing/overview

[26] Divya Gopinath, Mengshi Zhang, Kaiyuan Wang, Ismet Burak Kadron, Corina S. Pasareanu, and Sarfraz Khurshid. 2019. Symbolic Execution for Importance Analysis and Adversarial Generation in Neural Networks. In *30th IEEE International Symposium on Software Reliability Engineering, ISSRE 2019, Berlin, Germany*. IEEE, 313–322. https://doi.org/10.1109/ISSRE.2019.00039

[27] Gilad Gressel, Niranjan Hegde, Archana Sreekumar, and Michael C. Darling. 2021. Feature Importance Guided Attack: A Model Agnostic Adversarial Attack. *CoRR* abs/2106.14815 (2021). arXiv:2106.14815 https://arxiv.org/abs/2106.14815

[28] J A Hanley and B J McNeil. 1982. The meaning and use of the area under a receiver operating characteristic (ROC) curve. *Radiology* 143, 1 (1982), 29–36. https://doi.org/10.1148/radiology.143.1.7063747

[29] Jun Ho Huh and Hyoungshick Kim. 2011. Phishing Detection with Popular Search Engines: Simple and Effective. In *Foundations and Practice of Security - 4th Canada-France MITACS Workshop, FPS 2011, Paris, France, Revised Selected Papers (Lecture Notes in Computer Science, Vol. 6888)*. Springer, 194–207. https://doi.org/10.1007/978-3-642-27901-0_15

[30] Ankit Kumar Jain and BB Gupta. 2018. PHISH-SAFE: URL features-based phishing detection system using machine learning. In *Cyber Security: Proceedings of CSI 2015*. Springer, 467–474.

[31] Surgan Jandial, Puneet Mangla, Sakshi Varshney, and Vineeth Balasubramanian. 2019. AdvGAN++: Harnessing Latent Layers for Adversary Generation. In *2019 International Conference on Computer Vision Workshops, ICCV Workshops 2019, Seoul, Korea*. IEEE, 2045–2048. https://doi.org/10.1109/ICCVW.2019.00257

[32] Danny Karmon, Daniel Zoran, and Yoav Goldberg. 2018. LaVAN: Localized and Visible Adversarial Noise. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden (Proceedings of Machine Learning Research, Vol. 80)*. PMLR, 2512–2520. http://proceedings.mlr.press/v80/karmon18a.html

[33] Alexey Kurakin, Ian J. Goodfellow, and Samy Bengio. 2017. Adversarial examples in the physical world. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Workshop Track Proceedings*. OpenReview.net. https://openreview.net/forum?id=HJGU3Rodl

[34] Alexey Kurakin, Ian J. Goodfellow, and Samy Bengio. 2017. Adversarial Machine Learning at Scale. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, Conference Track Proceedings*. OpenReview.net. https://openreview.net/forum?id=BJm4T4Kgx

[35] Bin Liang, Miaoqiang Su, Wei You, Wenchang Shi, and Gang Yang. 2016. Cracking Classifiers for Evasion: A Case Study on the Google's Phishing Pages Filter. In *Proceedings of the 25th International Conference on World Wide Web, WWW 2016, Montreal, Canada*. ACM, 345–356. https://doi.org/10.1145/2872427.2883060

[36] Aishan Liu, Xianglong Liu, Jiaxin Fan, Yuqing Ma, Anlan Zhang, Huiyuan Xie, and Dacheng Tao. 2019. Perceptual-Sensitive GAN for Generating Adversarial Patches. In *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, Honolulu, Hawaii, USA*. AAAI Press, 1028–1035. https://doi.org/10.1609/aaai.v33i01.33011028

[37] Eric Medvet, Engin Kirda, and Christopher Kruegel. 2008. Visual-similarity-based phishing detection. In *4th International ICST Conference on Security and Privacy in Communication Networks, SECURECOMM 2008, Istanbul, Turkey*. ACM, 22. https://doi.org/10.1145/1460877.1460905

[38] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. 2016. DeepFool: A Simple and Accurate Method to Fool Deep Neural Networks. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*. IEEE Computer Society, 2574–2582. https://doi.org/10.1109/CVPR.2016.282

[39] OpenPhish. 2023. OpenPhish - Phishing Intelligence. https://openphish.com/

[40] Thomas Kobber Panum, Kaspar Hageman, René Rydhof Hansen, and Jens Myrup Pedersen. 2020. Towards Adversarial Phishing Detection. In *13th USENIX Workshop on Cyber Security Experimentation and Test, CSET 2020*. USENIX Association. https://www.usenix.org/conference/cset20/presentation/panum

[41] Nicolas Papernot, Patrick D. McDaniel, Somesh Jha, Matt Fredrikson, Z. Berkay Celik, and Ananthram Swami. 2016. The Limitations of Deep Learning in Adversarial Settings. In *IEEE European Symposium on Security and Privacy, EuroS&P 2016, Saarbrücken, Germany*. IEEE, 372–387. https://doi.org/10.1109/EuroSP.2016.36

[42] Andras Rozsa, Ethan M. Rudd, and Terrance E. Boult. 2016. Adversarial Diversity and Hard Positive Generation. In *2016 IEEE Conference on Computer Vision and Pattern Recognition Workshops, CVPR Workshops 2016, Las Vegas, NV, USA*. IEEE Computer Society, 410–417. https://doi.org/10.1109/CVPRW.2016.58

[43] Wei Shan, Guangling Sun, Xiaofei Zhou, and Zhi Liu. 2017. Two-Stage Transfer Learning of End-to-End Convolutional Neural Networks for Webpage Saliency Prediction. In *Intelligence Science and Big Data Engineering - 7th International Conference, IScIDE 2017, Dalian, China, September 22-23, 2017, Proceedings (Lecture Notes in Computer Science, Vol. 10559)*. Springer, 316–324. https://doi.org/10.1007/978-3-319-67777-4_27

[44] Mahmood Sharif, Sruti Bhagavatula, Lujo Bauer, and Michael K. Reiter. 2016. Accessorize to a Crime: Real and Stealthy Attacks on State-of-the-Art Face Recognition. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*. ACM, 1528–1540. https://doi.org/10.1145/2976749.2978392

[45] Chengyao Shen and Qi Zhao. 2014. Webpage saliency. In *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, Proceedings, Part VII 13*. Springer, 33–46.

[46] Hossein Shirazi, Bruhadeshwar Bezawada, Indrakshi Ray, and Charles Anderson. 2019. Adversarial Sampling Attacks Against Phishing Detection. In *Data and Applications Security and Privacy XXXIII - 33rd Annual IFIP WG 11.3 Conference, DBSec 2019, Charleston, SC, USA, Proceedings (Lecture Notes in Computer Science, Vol. 11559)*. Springer, 83–101. https://doi.org/10.1007/978-3-030-22479-0_5

[47] Lior Sidi, Asaf Nadler, and Asaf Shabtai. 2019. MaskDGA: A Black-box Evasion Technique Against DGA Classifiers and Adversarial Defenses. *CoRR* abs/1902.08909 (2019). arXiv:1902.08909 http://arxiv.org/abs/1902.08909

[48] M Somesha, Alwyn Roshan Pais, Routhu Srinivasa Rao, and Vikram Singh Rathour. 2020. Efficient deep learning techniques for the detection of phishing websites. *Sādhanā* 45 (2020), 1–18.

[49] Yang Song, Rui Shu, Nate Kushman, and Stefano Ermon. 2018. Constructing Unrestricted Adversarial Examples with Generative Models. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, Montréal, Canada*. 8322–8333. https://proceedings.neurips.cc/paper/2018/hash/8cea559c47e4fbdb73b23e0223d04e79-Abstract.html

[50] Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin A. Riedmiller. 2015. Striving for Simplicity: The All Convolutional Net. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Workshop Track Proceedings*. http://arxiv.org/abs/1412.6806

[51] Jiawei Su, Danilo Vasconcellos Vargas, and Kouichi Sakurai. 2019. One Pixel Attack for Fooling Deep Neural Networks. *IEEE Trans. Evol. Comput.* 23, 5 (2019), 828–841. https://doi.org/10.1109/TEVC.2019.2890858

[52] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian J. Goodfellow, and Rob Fergus. 2014. Intriguing properties of neural networks. In *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, Conference Track Proceedings*. http://arxiv.org/abs/1312.6199

[53] Cisco Talos Intelligence Group (Talos). 2023. PhishTank | Join the fight against phishing. https://phishtank.org/

[54] Florian Tramèr, Alexey Kurakin, Nicolas Papernot, Ian J. Goodfellow, Dan Boneh, and Patrick D. McDaniel. 2018. Ensemble Adversarial Training: Attacks and Defenses. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, Conference Track Proceedings*. OpenReview.net. https://openreview.net/forum?id=rkZvSe-RZ

[55] Bo Wei, Rebeen Ali Hamad, Longzhi Yang, Xuan He, Hao Wang, Bin Gao, and Wai Lok Woo. 2019. A Deep-Learning-Driven Light-Weight Phishing Detection Sensor. *Sensors* 19, 19 (2019), 4258. https://doi.org/10.3390/s19194258

[56] Colin Whittaker, Brian Ryner, and Marria Nazif. 2010. Large-Scale Automatic Classification of Phishing Pages. In *Proceedings of the Network and Distributed System Security Symposium, NDSS 2010, San Diego, California, USA*. The Internet Society. https://www.ndss-symposium.org/ndss2010/large-scale-automatic-classification-phishing-pages

[57] Chaowei Xiao, Bo Li, Jun-Yan Zhu, Warren He, Mingyan Liu, and Dawn Song. 2018. Generating Adversarial Examples with Adversarial Networks. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence, IJCAI 2018, Stockholm, Sweden*. 3905–3911. https://doi.org/10.24963/ijcai.2018/543

[58] Xi Xiao, Dianyan Zhang, Guangwu Hu, Yong Jiang, and Shutao Xia. 2020. CNN-MHSA: A Convolutional Neural Network and multi-head self-attention combined approach for detecting phishing websites. *Neural Networks* 125 (2020), 303–312. https://doi.org/10.1016/j.neunet.2020.02.013

[59] Chenglin Yang, Adam Kortylewski, Cihang Xie, Yinzhi Cao, and Alan L. Yuille. 2020. PatchAttack: A Black-Box Texture-Based Attack with Reinforcement Learning. In *Computer Vision - ECCV 2020 - 16th European Conference, Glasgow, UK, Proceedings, Part XXVI (Lecture Notes in Computer Science, Vol. 12371)*. Springer, 681–698. https://doi.org/10.1007/978-3-030-58574-7_41

[60] Peng Yang, Guangzhen Zhao, and Peng Zeng. 2019. Phishing Website Detection Based on Multidimensional Features Driven by Deep Learning. *IEEE Access* 7 (2019), 15196–15209. https://doi.org/10.1109/ACCESS.2019.2892066

[61] Suleiman Y. Yerima and Mohammed K. Alzaylaee. 2020. High Accuracy Phishing Detection Based on Convolutional Neural Networks. *CoRR* abs/2004.03960 (2020). https://arxiv.org/abs/2004.03960

[62] Ping Yi, Yuxiang Guan, Futai Zou, Yao Yao, Wei Wang, and Ting Zhu. 2018. Web Phishing Detection Using a Deep Learning Framework. *Wirel. Commun. Mob. Comput.* 2018 (2018), 4678746:1–4678746:9. https://doi.org/10.1155/2018/4678746

[63] Tianhang Zheng, Changyou Chen, and Kui Ren. 2019. Distributionally Adversarial Attack. In *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, Honolulu, Hawaii, USA*. AAAI Press, 2253–2260. https://doi.org/10.1609/aaai.v33i01.33012253