Raze to the Ground: Query-Efficient Adversarial HTML Attacks on Machine-Learning Phishing Webpage Detectors

Biagio Montaruli
biagio.montaruli@sap.com
SAP Security Research & EURECOM
Mougins, France

Luca Demetrio
luca.demetrio@unige.it
University of Genova & Pluribus One
Genova, Italy

Maura Pintor
maura.pintor@unica.it
University of Cagliari & Pluribus One
Cagliari, Italy

Luca Compagna
luca.compagna@sap.com
SAP Security Research
Mougins, France

Davide Balzarotti
davide.balzarotti@eurecom.fr
EURECOM
Biot, France

Battista Biggio
battista.biggio@unica.it
University of Cagliari & Pluribus One
Cagliari, Italy

ABSTRACT

Machine-learning phishing webpage detectors (ML-PWD) have been shown to suffer from adversarial manipulations of the HTML code of the input webpage. Nevertheless, the attacks recently proposed have demonstrated limited effectiveness due to their lack of optimizing the usage of the adopted manipulations, and they focus solely on specific elements of the HTML code. In this work, we overcome these limitations by first designing a novel set of fine-grained manipulations which allow to modify the HTML code of the input phishing webpage without compromising its maliciousness and visual appearance, i.e., the manipulations are functionality- and rendering-preserving by design. We then select which manipulations should be applied to bypass the target detector by a query-efficient black-box optimization algorithm. Our experiments show that our attacks are able to *raze to the ground* the performance of current state-of-the-art ML-PWD using just 30 queries, thus overcoming the weaker attacks developed in previous work, and enabling a much fairer robustness evaluation of ML-PWD.

CCS CONCEPTS

• **Security and privacy → Phishing**; • **Computing methodologies → Machine learning**.

KEYWORDS

machine learning, phishing, adversarial attacks

1 INTRODUCTION

Over the past years, we witnessed a significant increase in the number of phishing attacks [28, 41, 45], thereby emphasizing that this remains a significant form of cybercrime. Among all the different types of phishing, this work focuses on the detection of phishing *webpages*, which are typically created by an attacker to steal sensitive information such as login credentials [5]. To counter this open problem, in addition to the use of blocklists [35, 40] that have been demonstrated easy to bypass by *adaptive* attackers [47], novel approaches based on machine-learning [16, 24, 26, 34, 42, 46, 47] have been proposed in recent years to enhance the detection capabilities of phishing detection systems. However, phishing webpage detectors based on machine-learning (i.e., ML-PWD, using the same acronym of Apruzzese *et al.* [5]) have been shown to be vulnerable to adversarial attacks [2, 4, 5, 8, 9, 16, 23, 30], both in the *problem space*, which is the input space of HTML pages, and the *feature space*, which is the space where webpages are represented as feature vectors [5, 7]. In problem-space attacks, the attacker directly manipulates the URL [8, 9], the HTML code [29, 30] or the visual representation [2] of the phishing webpage with physically realizable manipulations [7], while feature-space attacks only manipulate the abstract feature representation of input samples. To this extent, *SpacePhish* [5] represents one of the most recent and comprehensive studies about adversarial attacks against ML-PWD both in the problem and feature spaces. Indeed, its authors provide a well-validated benchmark of state-of-the-art ML-PWD.

However, their work is characterized by two major limitations. First, investigating how the adversarial robustness changes if the attacker is able to optimize the adversarial attacks by querying the target ML-PWD is an important open point of their work. Second, they use a limited set of *cheap* adversarial manipulations i.e., manipulations that do not require any knowledge about the structure of phishing webpages such as the insertion of internal links and URL-shortening [5, 6], which result in weak or, in some cases, even useless attacks. Indeed, the reported results show that, for some evaluated ML-PWD, such *cheap* attacks (indicated as WA^r in their paper) cause the manipulated phishing webpage to appear even *more malicious* to the ML-PWD.

To address the aforementioned limitations, we propose a novel methodology for generating optimized and query-efficient HTML adversarial attacks (see Figure 1). Specifically, we first perform a thorough security analysis of the HTML features used in *SpacePhish*
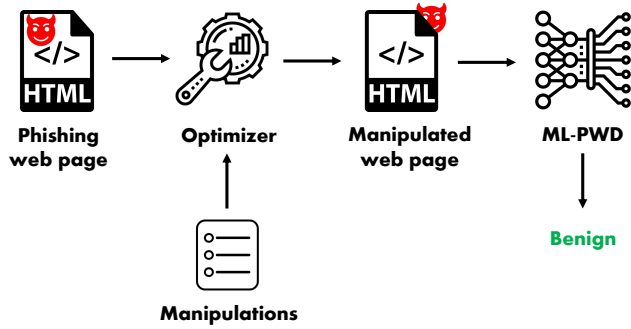
Figure 1: Overview of our work: we propose a novel set of adversarial manipulations that are functionality- and rendering-preserving by design, and a query-efficient black-box optimizer to generate HTML adversarial attacks that are able to *raze to the ground* state-of-the-art machine-learning phishing webpage detectors (ML-PWD).

(Sect. 2), which are widely adopted in the literature [24, 26, 32, 42], to understand how they can be evaded. Then, in addition to the HTML manipulations proposed in *SpacePhish*, we design a novel set of 14 manipulations that maintain the original functionality [17] and rendering [21] while manipulating the HTML code of phishing webpages (Sect. 4.1). On the basis of these manipulations, we formulate a query-efficient black-box optimization algorithm (Sect. 4.2) that generates optimized adversarial phishing webpages in the problem space.

Finally, we validate our approach through an extensive experimental analysis (Sect. 5), showing that our novel adversarial attacks are able to completely evade the ML-PWD evaluated in *SpacePhish* using just 30 queries. To foster reproducible results, we share the source code of our work[1].

To summarize, we provide the following three contributions:

- We conduct a comprehensive security analysis of the HTML features used in *SpacePhish* and, on top of it, we devise a novel set of adversarial manipulations that are functionality- and rendering-preserving by design, with the goal to evade all the analyzed features.
- We propose a black-box optimizer inspired by mutation-based fuzzing [50], which allows to craft optimized HTML adversarial attacks using the proposed manipulations;
- We empirically show that our methodology allows to *raze to the ground* the detection capabilities of current state-of-the-art ML-PWD using very few queries.

We conclude the paper by discussing the open points of our work, along with promising future research directions (Sect. 6).

2 BACKGROUND

In this section, we first give an overview of the basic structure of webpages and then we describe the HTML features adopted in *SpacePhish* [5].

---
[1]https://github.com/advmlphish/raze_to_the_ground_aisec23

2.1 Webpage Structure

Webpages are generally described using the HTML language [1]. They have a basic structure that consists of a tree hierarchy represented by the HTML Document Object Model (DOM) tree [25], which is made of multiple HTML elements corresponding to the DOM nodes. Each HTML element is represented through (i) a single tag or a pair of (start and end) tags and (ii) some content that includes text or other nested HTML elements. Moreover, HTML elements can have attributes consisting of name-value pairs to provide additional information about the element. Although the HTML specification includes many types of elements, a typical webpage (see Listing 1) includes the head (lines 3-8) and the body (lines 9-17) represented by the **<head>** and **<body>** element, respectively. The head is used to set the webpage title through the **<title>** element (line 4) and optionally to define the visual appearance of some embedded HTML elements through the **<style>** element (lines 5-7). The body, instead, includes the main content of the webpage, i.e., all the HTML elements that are generally displayed by a web browser. For instance, the example webpage includes a login form (lines 11-16), defined via a **<form>** element, which consists of two **<input>** elements used to collect the username (line 13) and password (line 15) from the user.

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4  <title>Website title</title>
5  <style>
6    h1 {color: red;}
7  </style>
8  </head>
9  <body>
10   <h1>Welcome to the website</h1>
11   <form action="login.php", method="get">
12     <label for="pwd">Enter your username: </label>
13     <input type="text" name="username" required>
14     <label for="pwd">Enter your password: </label>
15     <input type="password" name="pass" required>
16   </form>
17 </body>
18 </html>
```

Listing 1: Example of a webpage.

2.2 HTML Feature Analysis

In the following we analyze in details the features used in *SpacePhish* to better understand how they work and thus, how to evade them. This is an missing point in *SpacePhish*. Indeed, its authors only provide a brief description of some of them (5 out 22) in the related supplementary document [6], and do not carefully analyze how they can bypassed using problem-space manipulations. We also remark that such features are also widely used in other papers [24, 26, 32, 42], and some of them also in competitions about machine learning security such as the Machine Learning Security Evasion Competition (MLSEC)[2] [6, 21].

HTML_freqDom. This feature analyzes the number of internal (n_int) and external (n_ext) HTML elements in the webpage. An

---
[2]https://www.robustintelligence.com/blog-posts/ml-security-evasion-competition-2022

element is internal if includes a link that shares the same domain as the webpage URL; otherwise, it is external. Then, if n_ext is 0 or n_int ≥ n_ext, this feature is set to -1 (the webpage is likely benign); else, it is set to +1 (the webpage is likely phishing). This feature analyzes the following types of HTML elements: anchors (<**a**>), images (<**img**>), links (<**link**>) and videos (<**video**>).

HTML_objectRatio. This feature represents the ratio between the number of external HTML elements, n_ext, and the total one, n_tot = n_ext + n_int, where n_int represents the number of internal HTML elements. The ratio is compared against two thresholds: the suspicious (0.15) and phishing (0.30) thresholds. If the ratio is lower than the suspicious threshold, the value of the feature is -1 (the webpage is likely benign). Otherwise, if the ratio is in between the two thresholds, the webpage is assumed suspicious and the feature is set to 0. Finally, if the ratio is greater than the phishing threshold, the feature is set to +1 (the webpage is likely phishing). The HTML elements considered by this feature are the same as HTML_freqDom.

HTML_metaScripts. This feature is similar to HTML_objectRatio, but it applies to script (<**script**>), meta (<**meta**>) and link (<**link**>) elements. This feature adopts two different values for the thresholds. Specifically, the suspicious and phishing thresholds are set to 0.52 and 0.61, respectively. Moreover, if the ratio is greater than 0.61, the feature is set to +1 (the webpage is likely phishing); if the ratio is less than 0.52, the feature is -1 (the webpage is likely benign); otherwise, it is set to 0 (the webpage is assumed suspicious).

HTML_commPage. This feature analyzes the number of internal (n_int) and external (n_ext) elements, and is initialized using the following formula:

$$\text{HTML\_commPage} = \frac{max(\text{n\_ext}, \text{n\_int})}{\text{n\_ext} + \text{n\_int}}$$

This feature takes into account the same HTML elements analyzed by both HTML_objectRatio and HTML_metaScripts.

HTML_commPageFoot. This feature works as HTML_commPage except that it focuses on the HTML elements included in the footer (<**footer**>) rather than the body of the webpage.

HTML_SFH. This feature computes the ratio of suspicious forms as the number of suspicious forms divided by the total number of forms. The ratio is compared against two thresholds: susp_thr, which is set to 0.5 and is used to decide if a webpage is suspicious, and phish_thr, which is set to 0.75 and allows to determine whether a webpage is phishing. According to its implementation, a form is considered suspicious if one of the following conditions is satisfied: it includes an external link (specified through the `action` attribute), the `action` attribute is set to "about:blank" (i.e., it points to a new blank webpage) or when it is set to an empty string (i.e., <**form** action="">). In particular, if the ratio is lower than the suspicious threshold, the feature is set to -1 (the webpage is likely benign). Else, if the ratio is greater than the phishing threshold, then the feature is initialized to +1 (the webpage is likely phishing). Otherwise, i.e., the ratio is between the two thresholds, this feature is set to 0 (the webpage is considered suspicious).

HTML_popUP. This feature checks whether the webpage displays a pop-up window that prompts the user for some input, such as credentials in case of phishing webpages. A pop-up window can be commonly introduced by using the `prompt()` or `window.open()`

Javascript (JS) functions. Specifically, this feature looks for the names of such functions and if finds the former, it is set to 1 (the webpage is likely phishing); while it is set to 0 (the webpage is likely suspicious) if finds the latter. Otherwise, its value is -1 (the webpage is likely benign).

HTML_rightClick. This feature inspects the source code of the webpage to determine if a context menu has been disabled, which is the equivalent of disabling the mouse right-click. In particular, it checks the following patterns to disable a context menu: if the `preventDefault()` method of the HTML DOM is present in the webpage or if there is at least one HTML element with the `oncontextmenu` attribute set to "return false". Hence, this feature is set to +1 (the webpage is likely phishing) if it finds at least one disabled context menu, and to -1 (the webpage is likely benign) otherwise.

HTML_domCopyright. This feature analyzes if the webpage contains a copyright notice with the copyright symbol (©). If not, the webpage is considered suspicious and its value is set to 0. Otherwise, if the copyright notice contains the website domain name, the feature is set to -1 (the webpage is likely benign). Else (i.e., no webpage domain in the copyright notice) it is set to +1 (the webpage is likely phishing).

HTML_nullLnkWeb. This feature computes the frequency of suspicious anchors contained in a website as the number of suspicious anchors divided by the total number of anchors. An anchor is considered suspicious if it contains one of the following useless links: "#", "#content", "#skip" and "JavaScript ::void(0)"; or if it is an internal link.

HTML_nullLnkFooter. This feature works in the same way as HTML_nullLnkWeb, but it computes the frequency of suspicious anchors included in the footer rather than the body.

HTML_brokenLnk. This feature computes the ratio of external elements with broken links (i.e., links that point to an unreachable website) against the total number of external ones included in the webpage. This feature analyzes the same HTML elements considered by both HTML_objectRatio and HTML_metaScripts.

HTML_loginForm. This feature is set to +1 (the webpage is likely phishing) if the webpage contains one or more forms with a useless internal link or an external one; otherwise, it is set to -1 (the webpage is likely benign). An internal link is useless if it is equal to one of the following: "" (empty string), "#", "#nothing", "#null", "#void", "#doesnotexist", "#whatever", "javascript", "javascript::;", "javascript::void(0)", "javascript::void(0);".

HTML_hiddenDiv. This feature checks if there are content division elements, a.k.a. div (<**div**>), which are hidden by setting the `style` attribute to "visibility:hidden" or "display:none". If so, this feature is set to +1 (the webpage is likely phishing), else to -1 (the webpage is likely benign).

HTML_hiddenButton. This feature is set to +1 (the webpage is likely phishing) if there is at least one button (<**button**>) element disabled by setting the `style` attribute to "disabled". Otherwise, the webpage is considered benign and this feature is set to -1.

HTML_hiddenInput. This feature is set to +1 (the webpage is likely phishing) if there is at least one input element that is disabled (i.e., <**input** disabled>) or hidden (i.e., <**input** type="hidden">). Otherwise, this feature is set to -1 (the webpage is likely benign).

HTML_URLBrand. This feature analyzes the title (`<title>`) of the webpage to check whether it contains the website's domain name. If so, the webpage is considered benign and this feature is set to -1. Otherwise, it is initialized to +1 (the webpage is likely phishing). Moreover, if the title is not found, this feature is set to 0 (the webpage is suspicious).

HTML_iFrame. This feature targets inline frame elements, a.k.a. iframe (`<iframe>`), usually used to embed a webpage within another one, by checking patterns commonly used for hiding an iframe such as `<iframe style="display: none">` and `<iframe style="visibility: hidden">`. If any of these patterns are found, the feature is set to +1 (the webpage is likely phishing), else to -1 (the webpage is likely benign).

HTML_favicon. This feature checks if the *favicon* (i.e., an icon associated with a particular website) is loaded from an external source. If so, it is set to +1 (the webpage is likely phishing), while it is set to -1 (the webpage is likely benign) if the favicon is internal. Moreover, if no favicon is included in the webpage, it is considered suspicious and this feature is set to 0. To check the presence of the favicon, this feature looks for link elements including either `rel="shortcut icon"` or `rel="icon"` attributes.

HTML_statBar. This feature inspects the webpage to check whether it changes the text of the status bar at the bottom of the browser window by looking for the presence of `window.status` in the HTML code. If so, this feature is set to +1 (the webpage is likely phishing); else the value of the feature is -1 (the webpage is likely benign).

HTML_css. This feature checks whether the webpage uses an external CSS style sheet, i.e., if the style sheet is imported from an external web location using a link element as in the following example: `<link rel="stylesheet" href="mystyle.css">`. If so, this feature is set to +1 (the webpage is likely phishing), else to -1 (the webpage is likely benign).

HTML_anchors. This feature computes the ratio of suspicious anchors included in the webpage and compares it against two thresholds: suspicious (0.32) and phishing (0.505). Then, if there are no anchors in the webpage or the ratio is lower than the suspicious threshold, then this feature is set to -1 (the webpage is likely benign). Else, if the ratio is higher than the phishing threshold, it is set to +1 (the webpage is likely phishing). Otherwise, i.e., if the ratio is between the two thresholds, its value is 0 (the webpage is considered suspicious). An anchor is assumed suspicious if contains an external link or if it includes an internal link belonging to the same list of patterns checked by the HTML_nullLnkWeb feature, i.e., "#", "#skip", "#content" and "JavaScript ::void(0)".

3 THREAT MODEL

In this section, we first formalize the threat model used in our work, and then we compare it to the one proposed in *SpacePhish* [5].

3.1 Formalization

We describe the threat model according to the following four criteria widely used in the adversarial machine learning literature [12].
Goal. The goal of the adversary consists in causing an integrity violation by evading a target machine-learning phishing detector at test time through adversarial phishing webpages generated in the problem space. In other words, the adversary aims to manipulate the

HTML code of these webpages using functionality- and rendering-preserving manipulations so that they are classified as benign.

Knowledge. In our threat model, we assume a *black-box* scenario [12, 20]. Specifically, the machine-learning algorithm, its features, the parameters as well as the data, and the objective function used during the training phase are unknown to the attacker. Regarding the feature set, although it is generally assumed that the attacker does not know the exact features used by the machine learning algorithm [12], it is possible to obtain information about the most widely used features in the state-of-the-art by analyzing the description of many solutions that are publicly available in the literature (e.g., [5, 42]). Based on this idea, we have carefully analyzed the most common HTML features adopted in the literature and defined ad-hoc adversarial manipulations to evade all of them. In this way, the attacker can use all the defined manipulations with the aim to evade as many features as possible.

Capability. In our threat model, we assume that the attacker can use the ML-PWD as an *oracle* by querying it and collecting its output confidence score, representing the probability of classifying the input webpages as phishing.

Strategy. The adversarial phishing webpages can be generated by solving the following optimization problem:

$$\underset{t \in \mathcal{T}}{\text{minimize}} \quad f(h(z, t)), \tag{1}$$

which amounts to find the sequence of manipulations $t = [t_0, \ldots, t_K]$ that, when applied to the given phishing webpage z, generate an adversarial phishing webpage, $z^\star = h(z, t)$, that minimizes the confidence score $f(z^\star)$ returned by the target machine-learning model denoted with f. For simplicity, in our formulation we assume that the machine-learning model includes a feature extraction step before classification, i.e., f takes the raw webpage directly as input, but internally performs a preliminary step to map the input webpage to a feature vector. Moreover, $h : \mathcal{Z} \times \mathcal{T} \rightarrow \mathcal{Z}$ is a function that applies a sequence of functionality- and rendering-preserving manipulations t to the HTML code of the phishing webpage z, and outputs a valid webpage with the same behavior and rendering as the input one, but with a different HTML code. Under the given black-box setting, and considering that the feature extraction step performed by f may not be differentiable, the above optimization problem cannot be solved using classical gradient-based approaches. For this reason, in this work we adopt a *black-box (a.k.a. gradient-free)* optimization algorithm that is described in detail in subsection 4.2.

3.2 Comparison with *SpacePhish*

Our threat model differs from that proposed by Apruzzese *et al.* [5], as we assume the possibility of querying the ML-PWD. Recall indeed that this is a valid assumption adopted in many papers [10, 14, 29, 30, 37], especially when considering Machine-Learning-as-a-Service (MLaaS) scenarios, in which the attacker can interact with the target machine-learning model by sending queries to it and observing its predictions [36, 37]. For instance, those available through VirusTotal can be easily queried through dedicated APIs provided by the VirusTotal platform [15, 39]. In this work, we want to extend the threat model of *SpacePhish* in order to thoroughly evaluate the adversarial robustness of state-of-the-art ML-PWD when the attacker can optimize the adversarial attacks. Finally, it is

worth noting that, even if the output of the ML-PWD is not available, the attacker can still optimize the adversarial attacks by using a so-called surrogate model [17, 20, 36, 52]. However, this approach is out of the scope of our work.

4 OPTIMIZED HTML ADVERSARIAL ATTACKS

In this section, we describe our methodology for generating optimized and query-efficient HTML adversarial attacks. Specifically, we first present our novel set of 14 functionality- and rendering-preserving adversarial manipulations designed to evade the HTML features described in subsection 2.2. Then, we describe our black-box optimizer that uses the proposed manipulations in order to optimize the generation of adversarial phishing webpages.

4.1 Adversarial Manipulations

Each manipulation consists in a function that takes in input a phishing webpage, modifies its HTML code, and returns the new valid webpage with the same functionality and rendering as the input. In the following we will describe the details of our manipulations, including the HTML features they aim to evade, as well as how they preserve the original rendering and functionality.

InjectIntElem. This manipulation aims to inject a given number of internal HTML elements into the body of the webpage. It has been proposed in *SpacePhish* to implement the WA^r and $\widehat{WA^r}$ attacks with the aim to evade the HTML_objectRatio feature [6]. The former, WA^r, assumes no knowledge about the target phishing detectors and injects 50 hidden anchors with internal links. On the other hand, the latter, $\widehat{WA^r}$, assumes an attacker who knows how the HTML_objectRatio feature works including its thresholds, hence this manipulation injects as many links as needed to meet the suspicious threshold (0.15) so that the sample is considered benign by this feature. In our case, we also assume that the attacker does not know the internal thresholds used by the HTML_objectRatio feature. Therefore, in order to evade that feature, we design a black-box algorithm (see subsection 4.2) that iteratively applies this manipulation in order to inject a fixed number of internal elements, until the confidence score returned by the target phishing detector decreases, thus meaning that the feature has been evaded. In our implementation, we inject the same type of HTML elements as in *SpacePhish*, i.e., anchors, but the number of injected internal elements is set to 10 in order to have a finer level of granularity. Using this manipulation, we are able to evade other HTML features that depend on anchor elements with internal links, i.e., HTML_freqDom, HTML_commPage, HTML_nullLnkWeb. Regarding the HTML_nullLnkWeb feature, this manipulation only targets internal anchors included in the body or the footer. On the other hand, to bypass this feature when it searches for patterns that represent useless internal links we have created another manipulation, *UpdateIntAnchors*, which is described in the following.

Finally, since this manipulation injects some HTML elements, we must ensure that they are properly hidden in order to preserve the original rendering. To this end, there are several approaches that can be adopted by the attacker (see Listing 2):

(1) Using the `hidden` attribute (line 10). Inserting this attribute into an HTML element tells the browser to not render the

Manipulation	Evaded feature(s)	Type
InjectIntElem★	HTML_freqDom, HTML_objectRatio, HTML_commPage, HTML_nullLnkWeb (int. links)	MR
InjectIntElemFoot★	HTML_commPageFoot, HTML_nullLnkFooter (int. links)	MR
InjectIntLinkElem	HTML_metaScripts	MR
InjectExtElem	HTML_freqDom, HTML_objectRatio, HTML_metaScripts, HTML_commPage	MR
InjectExtElemFoot	HTML_commPageFoot	MR
UpdateForm	HTML_SFH (int. links), HTML_loginForm (int. links)	SR
ObfuscateExtLinks	HTML_SHF (ext. links), HTML_brokenLnk, HTML_anchors (ext. links), HTML_css, HTML_favicon (ext. links), HTML_loginForm (ext. links)	SR
ObfuscateJS	HTML_statBar, HTML_rightClick, HTML_popUP	SR
InjectFakeCopyright	HTML_domCopyright	SR
UpdateIntAnchors	HTML_anchors (int. links), HTML_nullLnkWeb (useless links), HTML_nullLnkFooter (useless links)	SR
UpdateHiddenDivs	HTML_hiddenDiv	SR
UpdateHiddenButtons	HTML_hiddenButton	SR
UpdateHiddenInputs	HTML_hiddenInput	SR
UpdateTitle	HTML_URLBrand	SR
UpdateIFrames	HTML_iFrame	SR
InjectFakeFavicon	HTML_favicon (no favicon included)	SR

Table 1: Adversarial manipulations used in this work along with the corresponding evaded features and their type, defined according to the way they can be applied by the black-box optimizer (see subsection 4.2), i.e., single-round (SR) or multi-round (MR). The manipulations marked with ★ have been originally proposed by Apruzzese *et al.* [5].

content of the element. This is the default approach adopted by this manipulation.

(2) Modifying the style of the element. It is possible to hide an HTML element setting the `style` attribute to `"display:none"` (line 11). This is the approach used in *SpacePhish* [5, 6].

(3) Similarly to (2), but using the `<style>` HTML element (lines 5-7) instead of the `style` attribute.

(4) Using `<noscript>` (lines 13-15) and add inside it the HTML elements to be hidden. It is worth noting that this only works if JS is enabled on the victim's web browser.

InjectIntElemFoot. This manipulation behaves similarly to *InjectIntElem* but injects the internal elements into the footer of the webpage to evade the HTML_commPageFoot and HTML_nullLnkFooter features.

InjectIntLinkElem. This manipulation works exactly as *InjectIntElem* but injects 10 hidden HTML elements of type `<link>` instead of `<a>` in order to evade the HTML_metaScript feature since it depends on `<link>` elements.

InjectExtElem. This manipulation behaves similarly to *InjectIntElem* but injects external HTML elements, i.e., elements with external links, instead of internal ones. Specifically, it injects 10 `<link>` elements that are also hidden as for *InjectIntElem* (i.e., by adding the `hidden` attribute) to preserve the original rendering. The injected external links are randomly extracted from a list of some well-known websites selected from the Alexa Top Million ranking[3] in order to appear benign. This manipulation evades multiple features that depend on external elements, which are HTML_freqDom, HTML_objectRatio, HTML_commPage, and HTML_metaScript.

InjectExtElemFoot. This manipulation works similarly as *InjectExtElem*, but the external elements are inserted into the footer of the webpage with the goal to evade the HTML_commPageFoot feature.

UpdateForm. This manipulation has been designed to evade the HTML_SFH and HTML_loginForm features when a form in the webpage includes an internal link matching one of the patterns searched by the two features, which represent useless internal links generally used by attackers such as "#". Specifically, this manipulation replaces the original internal link, specified with `action` attribute, with another random one that does not trigger the target features, such as "#!" or "#none". The original rendering is not affected because this manipulation updates a property of forms that does not affect the visual appearance of the webpage.

ObfuscateExtLinks. This manipulation aims to obfuscate the external links in a webpage in order to evade multiple HTML features, i.e., HTML_SHF, HTML_loginForm, HTML_css, HTML_anchors, HTML_brokenLnk and HTML_favicon. Specifically, this manipulation executes the following steps:

(1) Substitute the external link with a random internal one that is not detected as suspicious by the HTML features ("#!" as for HTML_SHF);

(2) Create a new script element (`<script>`) that updates the value of the "`action`" attribute to the original external link when the page is loaded;

(3) Add the new script element into the `<head>` of the webpage.

To better explain the obfuscation approach, let's consider a practical example that shows how to evade the HTML_SHF feature.

For instance, let's examine the simple webpage shown in Listing 3. It includes a form (lines 7-10) with a malicious external link (line 7) for stealing the victim's credentials, which is detected by the HTML_SHF feature. Listing 4 shows a new webpage in which the malicious link has been obfuscated using the script in lines 5-9. In particular, the original link assigned to `action` is updated with a random internal one ("#!"), but its original value is restored (line 7) when the page is loaded. This new adversarial phishing webpage has the same rendering as the original one but it is no longer detected by the HTML_SHF. Furthermore, this manipulation can be applied to obfuscate the external links included in any HTML elements, thus we use it to bypass multiple features as described in the following. Regarding the HTML_anchors feature, we use this manipulation to obfuscate the external links embedded in anchor elements, thus reducing the suspicious anchor rate computed by this feature. In this way, the attacker is still able to insert hidden anchors with malicious external links but without being detected by the HTML_anchors feature. This manipulation can also evade the HTML_brokenLnk feature by replacing all broken links (if any) with internal ones, hence resulting in a benign behavior for this feature. The same applies to HTML_loginForm, HTML_css and HTML_favicon, which can be evaded using this manipulation by obfuscating the external links analyzed by such features. Finally, It is worth noting that, although this manipulation modifies external links, it is independent of *InjectExtElem* and *InjectExtElemFoot* because they target different features. At the same time, the external links injected by these manipulations do not affect the features targeted by *ObfuscateExtLinks*.

ObfuscateJS. This manipulation aims to obfuscate the Javascript (JS) code inside the webpage inserted in `<script>` elements in order to evade the HTML_popUP, HTML_rightClick and HTML_statBar features. To achieve so, several techniques have been proposed in the literature [11, 49]. In this work, however, we use a different approach inspired to [21] for obfuscating the entire HTML code in a webpage, which is described in the following. For instance, let us consider the webpage in Listing 5, which includes a script element to open a malicious webpage. Because of the use of `window.open()` DOM method, the webpage is considered malicious by the HTML_popUP feature. To bypass such feature, this manipulation operates as follows:

(1) Extracts the JS code from the original script and encodes it into Base64 [27].

(2) Replaces the content of the original script with new JS code that creates a new script to hold the original JS code (line 5), decodes the original obfuscated JS code (line 6), and insert the new script into the webpage to be executed (line 8).

It is worth noting that this approach can be also used to obfuscate the patterns searched by the other target features. Moreover, the original rendering is preserved.

InjectFakeCopyright. This manipulation is used to evade the HTML_domCopyright by injecting a new hidden paragraph containing the copyright symbol followed by the "Copyright" string and the domain name of the website. For instance, assuming that the domain name of the webpage to manipulate is `mydomain`, the injected element is: `<p hidden>© Copyright mydomain </p>`. Since the injected paragraph is hidden, the original rendering is preserved.

UpdateIntAnchors. This manipulation is designed to evade the HTML_statBar, HTML_nullLnkWeb and HTML_nullLnkFooter features by replacing every useless internal link with another one that is not checked by such features, such as "#!". The original rendering is preserved since this manipulation does not affect it by design.

UpdateHiddenDivs. This manipulation is designed to evade the HTML_hiddenDiv feature by updating the way div elements (`<div>`) are hidden. It operates in different ways according to how a div element is hidden, i.e., by setting the style attribute to "display:none" or "visibility:hidden". The main difference between the two approaches consists in how they allocate the space for the hidden element when rendering the webpage. Specifically, the former (i.e., "visibility:hidden") still takes up space in the layout, while the latter (i.e., "display:none") does not take up any space. For instance, let's consider Listing 7 showing a div element hidden with "display:none" (line 7). It can be removed and, to achieve the same behavior and rendering, we can insert the hidden attribute (line 10 of Listing 8) in order to evade the HTML_hiddenDiv feature since it does not check for the presence of such attribute. However, we cannot adopt the same approach for obfuscating div elements hidden using "visibility:hidden" (line 11 of Listing 7), because this will change the rendering. In this case, we can still evade the HTML_hiddenDiv feature by removing "visibility:hidden" from the style attribute and inserting a new `<style>` element to achieve the same result (lines 5-7 of Listing 8).

UpdateHiddenButtons. This manipulation is designed to evade the HTML_hiddenButton feature by obfuscating all the disabled button elements. Specifically, for each disabled button, it removes the disabled attribute and inserts a new script element that, by exploiting JS, adds this attribute back during rendering using the setAttribute() DOM method. Notably, this approach is similar to the one adopted by *ObfuscateExtLinks* to obfuscate external links. Thus, both the rendering and original behavior are preserved.

UpdateHiddenInputs. This manipulation consists of evading the HTML_hiddenInput, and it operates in different ways according to whether the input element is hidden or disabled (since both are checked by the HTML_hiddenInput feature). Specifically, if the input element is hidden, this manipulation updates the value of its type attribute from "hidden" to "text" and then adds the hidden attribute. Otherwise, if the input element is disabled, then this manipulation operates in the same way as *UpdateHiddenButtons* by removing the attribute from the element and inserting it back during the rendering of the webpage by using JS. In both cases, the original behavior and rendering remain the same.

UpdateTitle. This manipulation aims to evade the HTML_URLBrand feature. Specifically, if the website's domain name is not included in the title element, this manipulation updates the webpage title with the website's domain name and then replaces back the original title during rendering using a script element (i.e., similarly as how *UpdateHiddenButtons* and *UpdateHiddenInputs* work).

UpdateIFrames. This manipulation adopts the same approach of *UpdateHiddenDivs*. Indeed, both the features look for the same patterns, but *UpdateIFrames* targets `<iframe>` elements in order to evade the HTML_iFrame feature.

InjectFakeFavicon. This manipulation is designed to inject a fake favicon in webpages that do not contain one, preventing them from being flagged as suspicious by the HTML_favicon feature.

Specifically, this manipulation injects a favicon element with a useless internal link, such as i.e., `<link rel="icon" href="#none">`, into the head of the webpage.

```
1   <!DOCTYPE html>
2   <html>
3   <head>
4   <title>Home</title>
5   <style>
6     #mypar {display: none;}
7   </style>
8   </head>
9   <body>
10    <p hidden="">Hidden text</p>
11    <p style="display:none">Hidden text</p>
12    <p id="mypar">Hidden text</p>
13    <noscript>
14      <p>Hidden text</p>
15    </noscript>
16  </body>
17  </html>
```

Listing 2: Example showing different approaches to hide HTML elements: using the `hidden` attribute (line 10), modifying the CSS style (lines 6 and 11), and embedding the element in `<noscript>` (line 14).

```
1   <!DOCTYPE html>
2   <html>
3   <head>
4   <title>Login</title>
5   </head>
6   <body>
7     <form id="myform" action="http://malicious.io">
8       <label for="pwd">Enter your password: </label>
9       <input type="password" name="pass" required>
10    </form>
11  </body>
12  </html>
```

Listing 3: Webpage including a form with a malicious external link (line 7) detected by the HTML_SHF feature.

4.2 Mutation-based Black-box Optimizer

To optimize the choice of the manipulations defined in subsection 4.1, we propose a black-box optimizer (shown in Algorithm 1) that is in line with the proposed threat model (see section 3). Our optimizer draws inspiration from the algorithm proposed in *WAF-A-MoLE* [18], which relies on mutation-based fuzzing techniques [50], recently shown to be promising for generating adversarial examples [18, 38]. Specifically, the algorithm of *WAF-A-MoLE* adopts an iterative approach consisting of consecutive mutation rounds with the aim to mutate the original malicious sample in order to minimize the confidence score returned by the machine-learning model. Starting from the original algorithm of *WAF-A-MoLE*, we have designed a novel one that is tailored to the proposed manipulations in order to improve its effectiveness, i.e., minimize the number of queries when generating the adversarial attacks. To this end, in the following, we first explain how the manipulations can be categorized in order to make the optimizer more query-efficient, and then we describe how the optimizer works step-by-step.

```
1   <!DOCTYPE html>
2   <html>
3   <head>
4   <title>Login</title>
5   <script type="text/javascript">
6   window.onload = function () {
7       document.getElementById("myform").setAttribute
    ("action", "http://malicious.io");
8   }
9   </script>
10  </head>
11  <body>
12    <form id="myform" action="#!">
13      <label for="pwd">Enter your password: </label>
14      <input type="password" name="passwd" required>
15    </form>
16  </body>
17  </html>
```

Listing 4: Adversarial phishing webpage generated using *ObfuscateExtLinks*, which obfuscates the malicious link (lines 5 - 9) in the original webpage of Listing 3.

```
1   <html>
2   <head>
3   <title>Home</title>
4   <script>
5     window.open("http://malicious.io", "_self");
6   </script>
7   </head>
8   <body>
9   </body>
10  </html>
```

Listing 5: Webpage using *window.open()* to load an external malicious link (line 5) detected by the HTML_popUP feature.

```
1   <html>
2   <head>
3   <title>Home</title>
4   <script>
5     let script = document.createElement("script");
6     script.innerHTML = atob("d2luZG93Lm9wZW4oImh0 \
7       dHA6Ly9tYWxpcY2lvdXMuaW8iLCAiX3NlbGYiKTs=");
8     document.head.append(script);
9   </script>
10  </head>
11  <body>
12  </body>
13  </html>
```

Listing 6: Adversarial phishing webpage manipulated using *ObfuscateJS* in order to obfuscate the JS code (lines 4 - 9) of the webpage shown in Listing 5.

Categorization of the HTML Manipulations. According to how the proposed manipulations can be applied to the input phishing webpage, they can be categorized into two main classes: single-round (SR), if they can be applied for just a single mutation round, or multi-rounds (MR), if they require more sequential mutation rounds. Specifically, SR manipulations generate the same output (i.e., a manipulated webpage) when used sequentially for more than one round, so it is sufficient to use them for a single round. On the other hand, this does not apply to MR manipulations, whose

```
1   <!DOCTYPE html>
2   <html>
3   <head>
4      <title>Home</title>
5   </head>
6   <body>
7      <div id="div1" style="display: none">
8        <p>Text in the first div.</p>
9      </div>
10
11     <div id="div2" style="visibility: hidden">
12       <p>Text in the second div.</p>
13     </div>
14  </body>
15  </html>
```

Listing 7: Webpage with two hidden div HTML elements (lines 7 and 11) detected by the HTML_hiddenDiv feature.

```
1   <!DOCTYPE html>
2   <html>
3   <head>
4   <title>Home</title>
5   <style>
6     #div2 {visibility: hidden;}
7   </style>
8   </head>
9   <body>
10     <div id="div1" hidden>
11       <p>Text in the first div.</p>
12     </div>
13
14     <div id="div2">
15       <p>Text in the second div.</p>
16     </div>
17  </body>
18  </html>
```

Listing 8: Adversarial webpage generated by manipulating the webpage of Listing 7 through *UpdateHiddenDivs*, which hides the div elements using CSS combined with the `<style>` element (line 6), and the `hidden` attribute (line 10).

output can change at each round. Furthermore, SR manipulations are independent of each other, while MR manipulations can be correlated, i.e., they can impact a common set of features.

To better explain the difference between the two classes, let's consider some of the manipulations defined in subsection 4.1. For instance, the *UpdateHiddenDivs* is an SR manipulation because, after it is used for the first time, all the related div elements are updated and there is no need to use it in the next rounds since no other manipulation can inject hidden div elements that may trigger the features (i.e., HTML_hiddenDiv) targeted by this manipulation. The same applies to other manipulations such as *UpdateHiddenButtons*, *UpdateHiddenInputs* and *UpdateTitle*. On the contrary, manipulations like *InjectIntElem* and *InjectExtElem* belong to the MR class because, in general, they need to be applied in multiple consecutive rounds to effectively evade the target HTML features. For instance, let's consider the HTML_commPage. In order to evade this feature, the attacker has to apply both *InjectIntElem* and *InjectExtElem* for multiple consecutive rounds to find the proper ratio between internal and external links. Clustering manipulations into

the two defined classes offers a significant advantage in enhancing the optimizer's efficiency. Indeed, if using the approach used in WAF-a-MoLE, which randomly selects manipulations for each mutation round, there's a risk of applying the same SR manipulation repeatedly in consecutive rounds, resulting in a significant waste of queries because the webpage would not be updated. Conversely, to address this issue our optimizer first executes the SR manipulations one by one, and then runs the main loop of mutational rounds by using only the MR manipulations.

Algorithm Description. Initially, the optimizer initializes the best adversarial example $z^{\star}$ and score $s^{\star}$ found so far with the initial phishing webpage z (line 1), and its score $f(z^{\star})$ (line 2). Then, it sequentially applies the SR manipulations (lines 3-5) and updates the best adversarial example and score found so far each time it finds a new manipulation that reduces the best score found so far (lines 6-8). Then, the optimizer executes the loop related to MR manipulations, which consists of R mutation rounds (line 9). Specifically, during each mutation round, the algorithm generates new candidates (i.e., adversarial phishing webpages) from the current best adversarial example by using one MR manipulation for each candidate (lines 11-14). Afterward, the algorithm selects the candidate having the lowest confidence score (line 15) and, in case its score is lower than the best score found so far (line 16), the chosen becomes the best adversarial example found so far (line 17). Finally, regarding the choice of the number of mutation rounds R, given the maximum query budget Q, it can be set using the following formula: $R = (Q - \#SR) / \#MR$, where $\#SR$ and $\#MR$ are the number of SR and MR manipulations, respectively.

5 EXPERIMENTAL ANALYSIS

In this section, we first describe the setup adopted in our experiments, and then we present and discuss the obtained results.

5.1 Experimental Setup

We now present the setup underlying our experimental analysis, conducted on an Ubuntu 18.04.6 LTS server equipped with an Intel Xeon E7-8880 CPU (16 cores) and 64 GB of RAM.

ML Algorithms. We evaluate the same machine-learning algorithms used in *SpacePhish* [5]:

- Logistic Regression (*LR*), a linear model also adopted in the Google phishing page filter [30, 44];
- Random Forest (*RF*), a tree-based ensemble learning algorithm [13] that has been shown outstanding performance in phishing detection tasks [47];
- Convolutional Neural Network (*CNN*), a deep learning [22] model used in [48] for detecting phishing webpages.

As for the feature set, we train each algorithm on the HTML features as well as the combination of both HTML and URL features, which are identified in *SpacePhish* as F^r and F^c, respectively [5]. The main reason for this choice is to assess the effectiveness of our adversarial attacks, particularly when incorporating supplementary features beyond those derived from the HTML code.

Dataset. We evaluate our approach on the *DeltaPhish* dataset [16], consisting of 5511 benign and 1012 phishing webpages. We perform a stratified random split (to preserve the original ratio between benign and phishing distributions) by using the 80:20 ratio, which

---

Algorithm 1: Mutation-based black-box optimizer to generate adversarial phishing webpages.

Data: z, the initial phishing sample;
  f, the machine-learning phishing webpage detector;
  h, the function to mutate the phishing webpages;
  R, the number of mutation rounds;
  SR the set of single-round (SR) manipulations;
  MR the set of multi-round (MR) manipulations.
Result: $z^{\star}$, the adversarial phishing sample.

1 $z^{\star} = z$
2 $s^{\star} = f(z^{\star})$
3 **for** t **in** SR
4      $z' = h(z^{\star}, [t])$
5      $s' = f(z')$
6   **if** $s' < s^{\star}$
7          $s^{\star} = s'$
8          $z^{\star} = z'$
9 **for** r **in** $[1, R]$
10      $C = \emptyset$
11   **for** t **in** MR
12          $z' = h(z^{\star}, [t])$
13          $s' = f(z')$
14          $C = C \cup \{(z', s')\}$
15      $z^b, s^b = get\_best\_candidate(C)$
16   **if** $s^b < s^{\star}$
17          $z^{\star} = z^b$
18          $s^{\star} = s^b$
19 **return** $z^{\star}$

---

is commonly used in related literature [3, 8]. In other words, 80% of both benign and phishing samples are used to build the training set, while the remaining 20% of samples are part of the test set.

Generation of Adversarial Phishing Webpages. We adopt the same approach of Apruzzese *et al.* [5]. In particular, we randomly select from the test set 100 phishing samples that are correctly classified by the best ML-PWD (typically F^c). Such 100 samples are used to evaluate the baseline detection rate of the target ML-PWD (i.e., no-atk), as well as to craft the adversarial examples using both the HTML adversarial attacks proposed in this work (our) and in *SpacePhish* (i.e., WA^r and $\widehat{\text{WA}^r}$) [5]. We would like to remind the reader that WA^r consists of injecting 50 hidden internal links, while $\widehat{\text{WA}^r}$ injects as many internal links as needed to meet the suspicious threshold (0.15) of the HTML_objectRatio feature. As for our approach, the query budget for optimizing the adversarial attacks is set to 36 queries, which implies 5 mutation rounds (i.e., $R = 5$ in Algorithm 1).

5.2 Results and Discussion

The experimental results are reported in Table 2 and Figure 2. The former shows the detection rate of the evaluated ML-PWD (*CNN*, *RF* and *LR*) on the baseline test set of 100 samples (no-atk), as well as their adversarial robustness against the attacks proposed in *SpacePhish* (WA^r and $\widehat{\text{WA}^r}$) in this work (our). The latter, instead,

reports the security evaluation curves that show the detection rate at 1% False Positive Rate (FPR) of the target ML-PWD w.r.t. the number of queries when the best sequence of manipulations is applied. It is worth noting that the drops in the detection rate represent manipulations that are effective in decreasing the confidence score and thus are included in the best (i.e., optimal) sequence of manipulations. Instead, flat regions indicate manipulations that are ineffective and thus are not used to generate the final adversarial example. Moreover, we have computed the detection rate at 1% FPR because this threshold is widely adopted in the literature [16, 19] as well as to perform a fair evaluation of the ML-PWD, i.e., they are evaluated assuming the same FPR. From the obtained results we can gain several takeaways that are described in the following.

Query-efficient Adversarial Attacks. The obtained results highlight that the proposed adversarial attacks clearly *raze to the ground* the detection rate of all the evaluated ML-PWD using just 30 queries, hence underlining the effectiveness of the proposed methodology. Specifically, by only using the SR manipulations (i.e., the first 11 queries shown on the left of the dotted vertical line in Figure 2) the average detection rate is lower than 50% for all the ML-PWD except for the *RF* model trained on the whole set of feature (F^c), whose detection rate is 53%. As for the MR manipulations, they play a crucial role in boosting the attack's effectiveness. Indeed, as depicted in Figure 2, finding the optimal number of internal and external elements to inject significantly reduces the detection rate to nearly zero within just a few queries. This also underlines that the HTML features related to the number of internal and external elements play a critical role in terms of adversarial robustness.

HTML Features Matter. Even more interesting is the fact that the proposed adversarial manipulations, while targeting the HTML features, have proven effective in evading the ML-PWD trained on the whole feature set F^c, including both the HTML and URL features. This underlines two key points. First, the adversarial robustness mainly relies on the HTML features, as also discussed above when analyzing the manipulations' effectiveness. Second, the supplementary URL features do not provide substantial benefits in terms of adversarial robustness. Indeed, an attacker can effectively evade the ML-PWD by exclusively leveraging the proposed manipulations targeting the HTML features.

ML algo	F	no-atk	WA^r	$\widehat{\text{WA}^r}$	our
CNN	F^r	0.81	0.33	0.78	**0.00**
	F^c	0.94	0.93	0.90	**0.00**
RF	F^r	0.95	0.90	0.79	**0.00**
	F^c	0.97	0.96	0.90	**0.00**
LR	F^r	0.72	0.51	0.53	**0.00**
	F^c	0.86	0.77	0.72	**0.00**

Table 2: Average detection rate at 1% FPR of the target ML-PWD (*CNN*, *RF* and *LR*) on the *DeltaPhish* dataset. Columns represent the baseline (no-atk), the attacks proposed in *SpacePhish* (WA^r and $\widehat{\text{WA}^r}$) [5], and our approach (our). The best results are in bold.
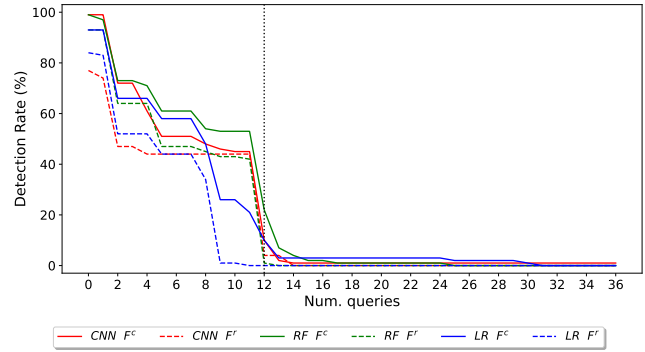

Figure 2: Security evaluation curves showing how the detection rate at 1% FPR of the target ML-PWD changes w.r.t. the number of queries when applying the best sequence of manipulations. Flat regions in the plot indicate manipulations that are not applied because they do not decrease the output score. The impact of SR and MR manipulations is shown on the left and right sides of the dotted vertical line, respectively.

6 CONCLUSIONS AND FUTURE WORK

In this work, we have introduced a novel methodology for generating query-efficient and notably effective HTML adversarial attacks. Specifically, we have designed a novel set of 14 functionality- and rendering-preserving manipulations that extend the current state-of-the-art, as well as a novel black-box optimizer tailored to such manipulations in order to generate adversarial phishing webpages that are able to *raze to the ground* several state-of-the-art machine-learning phishing webpage detectors (ML-PWD). Our experiments also reveal that the ML-PWD's adversarial robustness primarily depends on HTML features as our methodology effectively evades detection even when using additional URL features. To counter the adversarial attacks proposed in this work, a future work development is experimenting with well-known state-of-the-art approaches for increasing the adversarial robustness such as adversarial training [31, 51] and certified robustness techniques [33]. Moreover, although the HTML manipulations are specifically crafted to evade the features used in *SpacePhish* [6], another interesting future work is evaluating our methodology *in the wild*, i.e., assessing its effectiveness against production-grade phishing detectors, as well as other feature representations proposed in the literature. Finally, as for the proposed black-box optimizer, while it leverages the output scores to optimize the selection of the adversarial manipulations, in principle, it can be also extended to the *hard-label* scenario [43].

REFERENCES

[1] 2011. HTML 5. (April 2011). https://www.w3.org/TR/2011/WD-html5-20110405/.

[2] Sahar Abdelnabi, Katharina Krombholz, and Mario Fritz. 2020. VisualPhishNet: Zero-Day Phishing Website Detection by Visual Similarity. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security* (Virtual Event, USA) *(CCS '20)*. Association for Computing Machinery, New York, NY, USA, 1681–1698. https://doi.org/10.1145/3372297.3417233

[3] Rayah Al-Qurashi, Ahmed AlEroud, Ahmad A. Saifan, Mohammad Alsmadi, and Izzat Alsmadi. 2021. Generating Optimal Attack Paths in Generative Adversarial Phishing. In *2021 IEEE International Conference on Intelligence and Security Informatics (ISI)*. 1–6. https://doi.org/10.1109/ISI53945.2021.9624751

[4] Ahmed AlEroud and George Karabatis. 2020. Bypassing Detection of URL-Based Phishing Attacks Using Generative Adversarial Deep Neural Networks. In *Proceedings of the Sixth International Workshop on Security and Privacy Analytics* (New Orleans, LA, USA) *(IWSPA '20)*. Association for Computing Machinery, New York, NY, USA, 53–60. https://doi.org/10.1145/3375708.3380315

[5] Giovanni Apruzzese, Mauro Conti, and Ying Yuan. 2022. SpacePhish: The Evasion-Space of Adversarial Attacks against Phishing Website Detectors Using Machine Learning. In *Proceedings of the 38th Annual Computer Security Applications Conference* (Austin, TX, USA) *(ACSAC '22)*. Association for Computing Machinery, New York, NY, USA, 171–185. https://doi.org/10.1145/3564625.3567980

[6] Giovanni Apruzzese, Mauro Conti, and Ying Yuan. 2022. SpacePhish: The Evasion-space of Adversarial Attacks against Phishing Website Detectors using Machine Learning [Artifact]. In *Proceedings of the 38th Annual Computer Security Applications Conference*. ACM. https://spacephish.github.io/docs/ACSAC22_SpacePhish-supp.pdf

[7] Daniel Arp, Erwin Quiring, Feargus Pendlebury, Alexander Warnecke, Fabio Pierazzi, Christian Wressnegger, Lorenzo Cavallaro, and Konrad Rieck. 2022. Dos and Don'ts of Machine Learning in Computer Security. In *Proc. of USENIX Security Symposium*.

[8] 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)*. 1–4. https://doi.org/10.1109/ICMLANT53170.2021.9690540

[9] Alejandro Correa Bahnsen, Ivan Torroledo, Luis David Camacho, and Sergio Villegas. 2018. DeepPhish: Simulating Malicious AI. In *2018 APWG symposium on electronic crime research (eCrime)*. 1–8.

[10] Yang Bai, Yisen Wang, Yuyuan Zeng, Yong Jiang, and Shu-Tao Xia. 2023. Query efficient black-box adversarial attack on deep neural networks. *Pattern Recognition* 133 (2023), 109037.

[11] Benoît Bertholon, Sébastien Varrette, and Pascal Bouvry. 2013. JShadObf: A JavaScript Obfuscator Based on Multi-Objective Optimization Algorithms. In *Network and System Security*, Javier Lopez, Xinyi Huang, and Ravi Sandhu (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 336–349.

[12] Battista Biggio and Fabio Roli. 2018. Wild patterns: Ten years after the rise of adversarial machine learning. *Elsevier Pattern Recogn.* 84 (2018), 317–331.

[13] Leo Breiman. 2001. Random forests. *Machine learning* 45 (2001), 5–32. https://doi.org/10.1023/A:1010933404324

[14] Shuyu Cheng, Yinpeng Dong, Tianyu Pang, Hang Su, and Jun Zhu. 2019. Improving black-box adversarial attacks with a transfer-based prior. *Advances in neural information processing systems* 32 (2019).

[15] Euijin Choo, Mohamed Nabeel, Ravindu De Silva, Ting Yu, and Issa Khalil. 2022. A Large Scale Study and Classification of VirusTotal Reports on Phishing and Malware URLs. *arXiv preprint arXiv:2205.13155* (2022).

[16] Igino Corona, Battista Biggio, Matteo Contini, Luca Piras, Roberto Corda, Mauro Mereu, Guido Mureddu, Davide Ariu, and Fabio Roli. 2017. DeltaPhish: Detecting Phishing Webpages in Compromised Websites. In *Computer Security – ESORICS 2017*, Simon N. Foley, Dieter Gollmann, and Einar Snekkenes (Eds.). Springer International Publishing, Cham, 370–388.

[17] Luca Demetrio, Battista Biggio, Giovanni Lagorio, Fabio Roli, and Alessandro Armando. 2021. Functionality-Preserving Black-Box Optimization of Adversarial Windows Malware. *IEEE Transactions on Information Forensics and Security* 16 (2021), 3469–3478. https://doi.org/10.1109/TIFS.2021.3082330

[18] Luca Demetrio, Andrea Valenza, Gabriele Costa, and Giovanni Lagorio. 2020. WAF-A-MoLE: Evading Web Application Firewalls through Adversarial Machine Learning. In *Proceedings of the 35th Annual ACM Symposium on Applied Computing* (Brno, Czech Republic) *(SAC '20)*. Association for Computing Machinery, New York, NY, USA, 1745–1752. https://doi.org/10.1145/3341105.3373962

[19] Ambra Demontis, Marco Melis, Battista Biggio, Davide Maiorca, Daniel Arp, Konrad Rieck, Igino Corona, Giorgio Giacinto, and Fabio Roli. 2017. Yes, Machine Learning Can Be More Secure! A Case Study on Android Malware Detection. *IEEE Transactions on Dependable and Secure Computing* 16 (2017), 711–724.

[20] Ambra Demontis, Marco Melis, Maura Pintor, Matthew Jagielski, Battista Biggio, Alina Oprea, Cristina Nita-Rotaru, and Fabio Roli. 2019. Why Do Adversarial Attacks Transfer? Explaining Transferability of Evasion and Poisoning Attacks. In *28th USENIX Security Symposium (USENIX Security 19)*. USENIX Association, Santa Clara, CA, 321–338.

[21] Yang Gao, Benjamin M. Ampel, and Sagar Samtani. 2023. Evading Anti-Phishing Models: A Field Note Documenting an Experience in the Machine Learning Security Evasion Competition 2022. *Digital Threats* (jun 2023). https://doi.org/10.1145/3603507

[22] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning*. MIT Press. http://www.deeplearningbook.org

[23] 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).

[24] Abdelhakim Hannousse and Salima Yahiouche. 2020. Towards Benchmark Datasets for Machine Learning Based Website Phishing Detection: An experimental study. *CoRR* abs/2010.12847 (2020).

[25] Philippe Le Hégaret, Lauren Wood, and Jonathan Robie. 2004. *Document Object Model (DOM) Level 3 Core Specification*. Technical Report. W3C. https://www.w3.org/TR/DOM-Level-3-Core.

[26] Ankit Kumar Jain and Brij Bhooshan Gupta. 2018. Towards detection of phishing websites on client-side using machine learning based approach. *Telecommunication Systems* 68 (2018), 687–700.

[27] Simon Josefsson. 2003. The Base16, Base32, and Base64 Data Encodings. *RFC* 3548 (2003), 1–13. https://api.semanticscholar.org/CorpusID:5739143

[28] Kaspersky. 2023. Spam and phishing in 2022. https://securelist.com/spam-phishing-scam-report-2022/108692/

[29] Linfeng Li, Eleni Berki, Marko Helenius, and Saila Ovaska. 2014. Towards a Contingency Approach with Whitelist-and Blacklist-Based Anti-Phishing Applications: What Do Usability Tests Indicate? *Behav. Inf. Technol.* 33, 11 (nov 2014), 1136–1147. https://doi.org/10.1080/0144929X.2013.875221

[30] 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* (Montréal, Québec, Canada) *(WWW '16)*. International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, CHE, 345–356. https://doi.org/10.1145/2872427.2883060

[31] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. 2018. Towards Deep Learning Models Resistant to Adversarial Attacks. In *International Conference on Learning Representations (ICLR)*. https://openreview.net/forum?id=rJzIBfZAb

[32] Rami Mustafa A. Mohammad, Fadi A. Thabtah, and T. L. McCluskey. 2014. Intelligent rule-based phishing websites classification. *IET Inf. Secur.* 8 (2014), 153–160.

[33] Mark Niklas Müller, Franziska Eckert, Marc Fischer, and Martin T. Vechev. 2023. Certified Training: Small Boxes are All You Need. In *The Eleventh International Conference on Learning Representations*. https://openreview.net/forum?id=7oFuxtJtUMH

[34] Amirreza Niakanlahiji, Bei-Tseng Chu, and Ehab Al-Shaer. 2018. PhishMon: A Machine Learning Framework for Detecting Phishing Webpages. In *2018 IEEE International Conference on Intelligence and Security Informatics (ISI)*. 220–225. https://doi.org/10.1109/ISI.2018.8587410

[35] Adam Oest, Yeganeh Safaei, Penghui Zhang, Brad Wardman, Kevin Tyers, Yan Shoshitaishvili, and Adam Doupé. 2020. PhishTime: Continuous Longitudinal Measurement of the Effectiveness of Anti-phishing Blacklists. In *29th USENIX Security Symposium (USENIX Security 20)*. USENIX Association, 379–396.

[36] Alina Oprea and Apostol Vassilev. 2023. *Adversarial Machine Learning: A Taxonomy and Terminology of Attacks and Mitigations (Draft)*. Technical Report. National Institute of Standards and Technology. https://doi.org/10.6028/NIST.AI.100-2e2023.ipd

[37] Nicolas Papernot, Patrick McDaniel, and Ian Goodfellow. 2016. Transferability in machine learning: from phenomena to black-box attacks using adversarial samples. *arXiv preprint arXiv:1605.07277* (2016).

[38] Leo Hyun Park, Soochang Chung, Jaeuk Kim, and Taekyoung Kwon. 2023. GradFuzz: Fuzzing Deep Neural Networks with Gradient Vector Coverage for Adversarial Examples. *Neurocomput.* 522, C (feb 2023), 165–180. https://doi.org/10.1016/j.neucom.2022.12.019

[39] Peng Peng, Limin Yang, Linhai Song, and Gang Wang. 2019. Opening the Blackbox of VirusTotal: Analyzing Online Phishing Scan Engines. In *Proceedings of the Internet Measurement Conference* (Amsterdam, Netherlands) *(IMC '19)*. Association for Computing Machinery, New York, NY, USA, 478–485. https://doi.org/10.1145/3355369.3355585

[40] Pawan Prakash, Manish Kumar, Ramana Rao Kompella, and Minaxi Gupta. 2010. PhishNet: Predictive Blacklisting to Detect Phishing Attacks. In *2010 Proceedings IEEE INFOCOM*. 1–5. https://doi.org/10.1109/INFCOM.2010.5462216

[41] ProofPoint. 2023. State of the Phish 2023. https://www.proofpoint.com/us/resources/threat-reports/state-of-phish

[42] Suhas R. Sharma, Rahul Parthasarathy, and Prasad B. Honnavalli. 2020. A Feature Selection Comparative Study for Web Phishing Datasets. In *2020 IEEE International Conference on Electronics, Computing and Communication Technologies (CONECCT)*. 1–6. https://doi.org/10.1109/CONECCT50063.2020.9198349

[43] Satya Narayan Shukla, Anit Kumar Sahu, Devin Willmott, and Zico Kolter. 2021. Simple and Efficient Hard Label Black-Box Adversarial Attacks in Low Query Budget Regimes *(KDD '21)*. Association for Computing Machinery, New York, NY, USA, 1461–1469. https://doi.org/10.1145/3447548.3467386

[44] Fu Song, Yusi Lei, Sen Chen, Lingling Fan, and Yang Liu. 2021. Advanced evasion attacks and mitigations on practical ML-based phishing website classifiers. *International Journal of Intelligent Systems* 36, 9 (2021), 5210–5240. https://doi.org/10.1002/int.22510

[45] Todd Stansfield. 2023. *Q4 2022 Malware and Phishing Report*. Technical Report. Vade. https://www.vadesecure.com/en/blog/q4-2022-phishing-and-malware-report

[46] Lizhen Tang and Qusay H. Mahmoud. 2021. A Survey of Machine Learning-Based Solutions for Phishing Website Detection. *Machine Learning and Knowledge Extraction* 3, 3 (2021), 672–694. https://doi.org/10.3390/make3030034

[47] Ke Tian, Steve T. K. Jan, Hang Hu, Danfeng Yao, and Gang Wang. 2018. Needle in a Haystack: Tracking Down Elite Phishing Domains in the Wild. In *Proceedings of the Internet Measurement Conference 2018* (Boston, MA, USA) *(IMC '18)*. Association for Computing Machinery, New York, NY, USA, 429–442. https://doi.org/10.1145/3278532.3278569

[48] Wei Wei, Qiao Ke, Jakub Nowak, Marcin Korytkowski, Rafał Scherer, and Marcin Woźniak. 2020. Accurate and Fast URL Phishing Detector: A Convolutional Neural Network Approach. *Comput. Netw.* 178, C (sep 2020), 9 pages. https://doi.org/10.1016/j.comnet.2020.107275

[49] Wei Xu, Fangfang Zhang, and Sencun Zhu. 2012. The power of obfuscation techniques in malicious JavaScript code: A measurement study. In *2012 7th International Conference on Malicious and Unwanted Software*. 9–16. https://doi.org/10.1109/MALWARE.2012.6461002

[50] Andreas Zeller, Rahul Gopinath, Marcel Böhme, Gordon Fraser, and Christian Holler. 2023. Mutation-Based Fuzzing. In *The Fuzzing Book*. CISPA Helmholtz Center for Information Security. https://www.fuzzingbook.org/html/MutationFuzzer.html Retrieved 2023-01-07 14:53:00+01:00.

[51] H. Zheng, Z. Zhang, J. Gu, H. Lee, and A. Prakash. 2020. Efficient Adversarial Training With Transferable Adversarial Examples. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE Computer Society, Los Alamitos, CA, USA, 1178–1187. https://doi.org/10.1109/CVPR42600.2020.00126

[52] Yaoyao Zhong and Weihong Deng. 2021. Towards Transferable Adversarial Attack Against Deep Face Recognition. *IEEE Transactions on Information Forensics and Security* 16 (2021), 1452–1466. https://doi.org/10.1109/TIFS.2020.3036801