Razing to the Ground Machine-Learning Phishing Webpage Detectors with Query-Efficient Adversarial HTML Attacks

Biagio Montaruli\textsuperscript{1,2}, Luca Demetrio\textsuperscript{4,5}, Maura Pintor\textsuperscript{3,5}, Luca Compagna\textsuperscript{1}, Davide Balzarotti\textsuperscript{2}, Battista Biggio\textsuperscript{3,5}

\textsuperscript{1}SAP Security Research, \textsuperscript{2}EURECOM, \textsuperscript{3}University of Cagliari, \textsuperscript{4}University of Genova, \textsuperscript{5}Pluribus One
whoami

• Al Security researcher @ SAP Security Research

• Ph.D. candidate @ EURECOM

• Involved in the TESTABLE EU project

biagio.montaruli@sap.com
biagio.montaruli@eurecom.fr
biagiom
biagio-montaruli
Motivations and main takeaways

Phishing is a major attack vector to steal sensitive data from users
• Phishing attacks increased in 2023 by 102% quarter-over-quarter (QoQ)
• ML solutions are widely used to automate detection

Current adversarial attacks against ML-based phishing webpage detectors (ML-PWD) are “cheap”
• They adopt “cheap” manipulations that do not fully leverage domain knowledge
• What if the attacker is able to optimize the adversarial attacks using just the model output?

Towards a much fairer robustness evaluation of ML-PWD
• We designed 14 novel adversarial manipulations to evade some HTML features broadly used in the literature
• We proposed a new query-efficient black-box optimization algorithm tailored on such manipulations
• We managed to raze to the ground 6 state-of-the-art ML-PWD using just 30 queries
Phishing – An overview

How to fight phishing?

- Raising awareness through training
- Automated techniques based on ML
Machine Learning for anti-phishing

Webpage → Feature Extraction: $\begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{pmatrix}$ → ML model → Phishing, Benign

ML systems can be attacked!
Attacks against ML systems

<table>
<thead>
<tr>
<th>Attacker’s Goal</th>
<th>Misclassifications that do not compromise normal system operation</th>
<th>Misclassifications that compromise normal system operation</th>
<th>Querying strategies that reveal confidential information on the learning model or its users</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Attacker’s Capability</strong></td>
<td><strong>Integrity</strong></td>
<td><strong>Availability</strong></td>
<td><strong>Privacy / Confidentiality</strong></td>
</tr>
<tr>
<td><strong>Test data</strong></td>
<td>Evasion (a.k.a. adversarial examples)</td>
<td>Sponge Attacks</td>
<td>Model extraction / stealing</td>
</tr>
<tr>
<td>Training data</td>
<td>Backdoor/targeted poisoning (to allow subsequent intrusions) – e.g., backdoors or neural trojans</td>
<td>Indiscriminate (DoS) poisoning (to maximize test error)</td>
<td>Model inversion (hill climbing)</td>
</tr>
</tbody>
</table>

**Attacker’s Knowledge:** white-box / black-box (query/transfer) attacks (transferability with surrogate learning models)
Attack spaces of ML systems for anti-phishing

Problem Space
HTML Webpage

Feature Space
\[
\begin{pmatrix}
x_1 \\
x_2 \\
\vdots \\
x_d
\end{pmatrix}
\]
Feature Extraction

ML model

Phishing
Benign
Problem-space adversarial attacks

Why focusing on problem-space attacks when testing ML-based cybersecurity systems?
1. Threat model based on a black-box scenario: ML model and training data are not available
2. The target ML model may not be differentiable
   • Gradient-based techniques cannot be applied
3. Inverse feature-mapping problem

How to generate problem-space adversarial attacks?

Physically-realizable manipulations:
1) Satisfy physical constraints (i.e., format, executability)
2) Preserve the original functionality/semantic
Adversarial Machine Learning for anti-anti-phishing

Phishing webpage → Optimizer → Manipulated webpage → Feature Extraction → ML-PWD → Benign

Manipulations
State-of-the-art: SpacePhish

**3 Evasion spaces:**
1. Website
   - black-box (WA)
   - gray-box (WA)
2. Preprocessing (PA)
3. ML model (MA)

**3 ML models:**
- Convolutional Neural Network (CNN)
- Logistic Regression (LR)
- Random Forest (RF)

**3 Features groups:**
- URL ($F^u$, 35 features)
- HTML ($F^r$, 22 features)
- Combined ($F^c = F^u \cup F^r$, 57 features)
SpacePhish – Limitations

1) They focus on “cheap” manipulations that do not fully leverage the domain knowledge
2) Attacks are not optimized

The proposed manipulations are not so effective

Can we do better?


(a) Impact of WA on the ML-PWD trained on Zenodo.

(b) Impact of WA on the ML-PWD trained on δPhish.
Proposed methodology

We propose **14 novel functionality- and rendering-preserving HTML adversarial manipulations**

<table>
<thead>
<tr>
<th>Manipulation</th>
<th>Evaded feature(s)</th>
<th>Type</th>
<th>Manipulation</th>
<th>Evaded feature(s)</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>InjectIntElem*</td>
<td>HTML_freqDom, HTML_objectRatio, HTML_commPage, HTML_nullLinkWeb (int. links)</td>
<td>MR</td>
<td>InjectFakeCopyright</td>
<td>HTML_domCopyright</td>
<td>SR</td>
</tr>
<tr>
<td>InjectIntElemFoot*</td>
<td>HTML_commPageFoot, HTML_nullLinkFooter (int. links)</td>
<td>MR</td>
<td>UpdateIntAnchors</td>
<td>HTML_anchore (int. links), HTML_nullLinkWeb (useless links), HTML_nullLinkFooter (useless links)</td>
<td>SR</td>
</tr>
<tr>
<td>InjectIntLinkElem</td>
<td>HTML_metaScripts</td>
<td>MR</td>
<td>UpdateHiddenDivs</td>
<td>HTML_hiddenDiv</td>
<td>SR</td>
</tr>
<tr>
<td>InjectExtElem</td>
<td>HTML_freqDom, HTML_objectRatio, HTML_metaScripts, HTML_commPage</td>
<td>MR</td>
<td>UpdateHiddenButtons</td>
<td>HTML_hiddenButton</td>
<td>SR</td>
</tr>
<tr>
<td>InjectExtElemFoot</td>
<td>HTML_commPageFooter</td>
<td>MR</td>
<td>UpdateHiddenInputs</td>
<td>HTML_hiddenInput</td>
<td>SR</td>
</tr>
<tr>
<td>UpdateForm</td>
<td>HTML_SHF (int. links), HTML_loginaForm (int. links)</td>
<td>SR</td>
<td>UpdateTitle</td>
<td>HTML_URLBrand</td>
<td>SR</td>
</tr>
<tr>
<td>ObfuscateExtLinks</td>
<td>HTML_SHF (ext. links), HTML_brokenLink, HTML_anchore (ext. links), HTML_css, HTML_favicon (ext. links), HTML_loginaForm (ext. links)</td>
<td>SR</td>
<td>UpdateFrames</td>
<td>HTML_iFrame</td>
<td>SR</td>
</tr>
<tr>
<td>ObfuscateJS</td>
<td>HTML_statBar, HTML_rightClick, HTML_popUp</td>
<td>SR</td>
<td>InjectFakeFavicon</td>
<td>HTML_favicon (no favicon included)</td>
<td>SR</td>
</tr>
</tbody>
</table>

We design a new **query-efficient black-box optimizer** inspired to mutation-based fuzzing

**Algorithm 1:** 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: \( z^* \), the adversarial phishing sample.

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

*ObfuscateExtLinks* can be used to bypass the HTML_SHF feature, which checks for suspicious HTML forms:

$$HTML_{SFH} = \begin{cases} 
-1 & \text{if } n_{\text{susp}} < 0.5 \text{ (benign)} \\
0 & \text{if } n_{\text{susp}} \in [0.5, 0.75] \text{ (susp.)} \\
+1 & \text{if } \text{ratio} > 0.75 \text{ (phishing)} 
\end{cases}$$

A form is considered suspicious if:

- includes an external link
- the action attribute is set to `about:blank`:
  - it points to a blank webpage
- the action attribute is set to an empty string:
  $$\text{<form action="">}\text{<form id="myform" action="http://malicious.io">}\text{<form id="myform" action="#!">$$

```html
<!DOCTYPE html>
<html>
<head>
<title>Login</title>
<body>
<form id="myform" action="http://malicious.io">
  <label for="pwd">Enter your password: </label>
  <input type="password" name="pass" required>
</form>
</body>
</html>
```
UpdateHiddenDivs – Obfuscation of hidden `<div>`

`UpdateHiddenDivs` can be used to evade the `HTML_hiddenDiv` feature, which checks if there are `<div>` elements hidden by setting the `style` attribute to `visibility:hidden` or `display:none`

### `<div>` hidden using `display:none`
1) Remove `display:none` from the inline CSS style
2) Obfuscate it using the `hidden` attribute

```html
<!DOCTYPE html>
<html>
<head>
<title>Home</title>
</head>
<body>
  <div id="div1" style="display: none">
    <p>Text in the first div.</p>
  </div>
  <div id="div2" style="visibility: hidden">
    <p>Text in the second div.</p>
  </div>
</body>
</html>
```

### `<div>` hidden using `visibility:hidden`
1) Remove `visibility:hidden` from the inline CSS style
2) Obfuscate it using a new `<style>` object

```html
<!DOCTYPE html>
<html>
<head>
<title>Home</title>
</head>
<body>
  <style>
    #div2 {visibility: hidden;}
  </style>
  <div id="div1" hidden>
    <p>Text in the first div.</p>
  </div>
  <div id="div2">
    <p>Text in the second div.</p>
  </div>
</body>
</html>
```
Black-box optimizer

Algorithm 1: 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: $z^*$, the adversarial phishing sample.

1. $z^* = z$
2. $s^* = f(z^*)$
3. for $t$ in $SR$
4. \hspace{1cm} $z' = h(z^*, t)$
5. \hspace{1cm} $s' = f(z')$
6. \hspace{1cm} if $s' < s^*$
7. \hspace{1.5cm} $s^* = s'$
8. \hspace{1.5cm} $z^* = z'$
9. \hspace{1cm} for $r$ in [1, $R$]
10. \hspace{2cm} $C = \emptyset$
11. \hspace{1cm} for $t$ in $MR$
12. \hspace{2cm} $z' = h(z^*, t)$
13. \hspace{2cm} $s' = f(z')$
14. \hspace{2cm} $C = C \cup \{(z', s')\}$
15. \hspace{2cm} $z^b, s^b = get\_best\_candidate(C)$
16. \hspace{2cm} if $s^b < s^*$
17. \hspace{3cm} $s^* = s^b$
18. \hspace{3cm} $z^* = z^b$
19. return $z^*$
Black-box optimizer

Algorithm 1: 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: $z^*$, the adversarial phishing sample.

1. $z^* = z$
2. $s^* = f(z^*)$

for $t$ in $SR$

3. $z' = h(z^*, t)$
4. $s' = f(z')$
5. if $s' < s^*$
6. then $s^* = s'$
7. $z^* = z'$

for $r$ in $[1, R]$

8. $C = \emptyset$
9. for $t$ in $MR$
10. $z' = h(z^*, t)$
11. $s' = f(z')$
12. $C = C \cup \{(z', s')\}$
13. $z^b, s^b = get\_best\_candidate(C)$
14. if $s^b < s^*$
15. then $s^* = s^b$
16. $z^* = z^b$
17. return $z^*$

Initialization phase:
Initialize the best adversarial example and score with the initial phishing sample and its score.
Algorithm 1: 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:** $z^*$, the adversarial phishing sample.

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

Single-Round (SR) phase:
- Try sequentially each SR manipulation
- If it reduces the best score found so far, update the best adversarial example
Black-box optimizer

Algorithm 1: 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: \( z^* \), the adversarial phishing sample.

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

Multi-Round (MR) phase:
- Try sequentially each MR manipulation to generate new candidates
- Get the best candidate (with lowest score)
- If such a candidate reduces the best score, it becomes the new best adversarial example
Black-box optimizer

Algorithm 1: 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: \( z^* \), the adversarial phishing sample.

1. \( z^* = z \)
2. \( s^* = f(z^*) \)
3. for \( t \) in \( SR \)
   4. \( z' = h(z^*, t) \)
   5. \( s' = f(z') \)
   6. if \( s' < s^* \)
      7. \( s^* = s' \)
   8. \( z^* = z' \)
4. for \( r \) in \([1, R]\)
   5. \( C = \emptyset \)
   6. for \( t \) in \( MR \)
      7. \( z' = b(z^*, t) \)
      8. \( s' = f(z') \)
      9. \( C = C \cup \{(z', s')\} \)
5. \( z^b, s^b = \text{get_best_candidate}(C) \)
6. if \( s^b < s^* \)
   7. \( s^* = s^b \)
   8. \( z^* = z^b \)
9. return \( z^* \)

Final phase:
Return the best adversarial phishing example
Razing to the ground the ML-PWD

Main results

1. The proposed attacks raze to the ground all the ML-PWD
   • Only 14 queries for the ML-PWD trained on the HTML features ($F^H$)
   • In 30 queries the ML-PWD trained on the whole feature set are able to completely evade all the ML-PWD

2. HTML features matter
   • While targeting only the HTML features, the manipulations are very effective in evading the ML-PWD trained on $F^c$
   • The adversarial robustness mainly relies on the HTML features
   • The URL features do not provide substantial robustness

3. Effectiveness of the manipulations
   • The SR manipulations reduces the detection rate (DR) to 50%
   • The MR manipulations significantly enhance the attack effectiveness, reducing the DR to near-zero with few queries
Wrap-up

1. We propose 14 novel functionality- and rendering-preserving HTML adversarial manipulations
   → New “CVEs” for the evaluated ML-PWD (and their features)

2. We design a new query-efficient optimizer tailored on the proposed manipulations to generate adversarial phishing webpages in the problem space
   → Optimizing the choice of the manipulations is the key to success

3. We release the source code and ML models:
   https://github.com/advmlphish/raze_to_the_ground_aisec23
   → To foster reproducibility and a much fairer evaluation of the ML-PWD’s robustness


Credits: Hyrum Anderson, Kevin Roundy, Savino Dambra