Visual-Similarity-Based Phishing Detection

Eric Medvet
DEEI
University of Trieste, Italy
emedvet@units.it

Engin Kirda
Eurecom, France
kirda@eurecom.fr

Christopher Kruegel
University of California, Santa Barbara, USA
chris@cs.ucsb.edu

Abstract

Phishing is a form of online fraud that aims to steal a user's sensitive information, such as online banking passwords or credit card numbers. The victim is tricked into entering such information on a web page that is crafted by the attacker so that it mimics a legitimate page. Recent statistics about the increasing number of phishing attacks suggest that this security problem still deserves significant attention.

In this paper, we present a novel technique to visually compare a suspected phishing page with the legitimate one. The goal is to determine whether the two pages are suspiciously similar. We identify and consider three page features that play a key role in making a phishing page look similar to a legitimate one. These features are text pieces and their style, images embedded in the page, and the overall visual appearance of the page as rendered by the browser. To verify the feasibility of our approach, we performed an experimental evaluation using a dataset composed of 41 real-world phishing pages, along with their corresponding legitimate targets. Our experimental results are satisfactory in terms of false positives and false negatives.

1 Introduction

Phishing is a form of online fraudulent activity in which an attacker aims to steal a victim's sensitive information, such as an online banking password or a credit card number. Victims are tricked into providing such information by a combination of spoofing techniques and social engineering. In practice, the victims often receive an email that tries to convince them to visit a web page that has been prepared by the attacker. This page mimics and spoofs a real service such as an online banking web site. Legitimately looking web forms are provided through which the attacker can harvest and collect confidential and sensitive information.

Although tricking people to make financial profit is an old idea, criminals have realized that social-engineering-based attacks are simple to perform and highly effective over the Internet. Hence, although highly publicized, phishing is still an important security problem and many Internet users fall victim to this fraud. Note that such attacks are not only problematic for Internet users, but also for organizations that provide financial services online. The reason is that when users fall victim to phishers, the organization providing the online service often suffers an image loss as well as financial damage.

In recent years, phishing attacks have gained attention because their numbers and their sophistication have been increasing. The Anti Phishing Work Group detected more than 25,000 unique phishing URLs in December 2007 [1]. Also, creating a phishing site has become easier. "Do-it-yourself" phishing kits [20] created by criminals can easily be used by technically unsophisticated attackers. Moreover, recently, more sophisticated phishing attacks have emerged. For example, [13] shows how attackers exploited an application-level vulnerability to insert a phishing page into a legitimate and trusted bank web site to steal banking credentials.

A typical phishing attack may be based on several techniques, including exploiting browser vulnerabilities or performing man-in-the middle attacks using a proxy. However, the most straightforward and widespread method consists of deploying a web page that looks and behaves like the one the user is familiar with.

In this paper, we present an effective approach to detect phishing attempts by comparing the visual similarity between a suspected phishing page and the legitimate site that is spoofed. When the two pages are "too" similar, a phishing warning is raised. In our system, we consider three features to determine page similarity: text pieces (including their style-related features), images embedded in the page, and the overall visual appearance of the page as seen by the user (after the browser has rendered it). We quantify

the similarity between the target and the legitimate page by comparing these features, computing a single similarity score.

We chose to perform a comparison based on page features that are visually perceived. This is because phishing pages mimic the look-and-feel of a legitimate site and aim to convince the victims that the site they are visiting is the one they are familiar with. Once trust is established based on visual similarity, there is a higher chance that the victim will provide her confidential information. Typically, a victim's visual attention focuses both on the global appearance of the page and on salient details such as logos, buttons, and labels. In fact, these observations are supported by both common sense and literature. For example, Dhamija et al. [5] show that about one in four users base their trust only on page content (i.e., images, page design, colors) to decide whether the page is legitimate or not. Obviously, these users are the ones that are most prone to fall victim to a phishing attack.

The solution that we propose in this paper was inspired by two previous, open-source, anti-phishing solutions: AntiPhish [8] and DOMAntiPhish [16]. AntiPhish is a browser plugin that keeps track of sensitive information. Whenever a user attempts to enter sensitive information on one site, and this information has previously been associated with a different, trusted site, a warning is generated. This is effective when a user inadvertently enters bank login information on a phishing site. However, AntiPhish suffers from the problem that legitimate reuse of credentials is also flagged as suspicious. To address this usability problem, DOMAntiPhish was proposed. For that approach, the authors compared the Document Object Models (DOMs) of the pages under analysis to determine whether the two pages are similar. When information is reused on a page that is similar to the original page (that is associated with the sensitive data), a phishing attempt is suspected. When the information is entered on a site that is completely different, the system assumes legitimate data reuse. Although DOMAntiPhish is able to identify phishing pages effectively, its major limitation is that the DOM tree is not necessarily a reliable feature to establish similarity between pages. In some cases, it is possible for the attacker to use different DOM elements to create a similar look-and-feel and appearance of a page. Furthermore, a phishing site that only consists of images cannot be detected.

In this paper, we propose a novel comparison technique that eliminates the shortcomings of AntiPhish and DOMAntiPhish. Our solution can be used together with these tools, but can also be integrated into any other anti-phishing system that can provide a list of legitimate sites that can be potential targets of phishing attempts.

We performed a real-world evaluation to verify the phishing detection effectiveness of our approach. Our dataset contained 41 real phishing pages, together with the corresponding, legitimate pages. Our results, in terms of false alarms and missed detection, are satisfactory. We observed no false positives. Furthermore, only two phishing attempts, which actually did not visually resemble the legitimate web pages, were missed.

The paper is structured as follows. The next section discusses related work. Section 3 presents the details of our approach. Section 4 describes the experiments we performed to verify the feasibility and effectiveness of our solution. Finally, Section 5 concludes the paper.

2 Related Work

Phishing is an important security problem. Although phishing is not new and, hence, should be well-known by Internet users, many people are still tricked into providing their confidential information on dubious web pages.

To counter the phishing threat, a number of anti-phishing solutions have been proposed, both by industry and academia. A class of anti-phishing approaches aims to solve the phishing problem at the email level. The key idea is that when a phishing email does not reach its victims, they cannot fall for the scam. Hence, filters and content-analysis techniques are often used to attempt to identify phishing emails before these emails are delivered to users. Clearly, this line of research is closely related to anti-spam research. By continuously training filters (e.g., Bayesian filters), a large number of phishing emails can be blocked. This is because such emails often contain words that may be identified as suspicious tokens that do not frequently occur in legitimate emails (e.g., "update", "login", etc.). The main disadvantage of anti-spam techniques is that their success depends on the availability of these filters and their proper training. That is, when the user does not actively help in training the filter, the filter typically does not perform as well as expected. Furthermore, even when filters are trained well and a user rarely receives any spam or phishing emails, once a phishing email bypasses the filter, the user's belief of the legitimacy of this mail is strengthened.

One reason for the abundance of spam and phishing emails is the fact that the Simple Mail Transport Protocol (SMTP) does not contain any authentication mechanisms. The sender information in an email message can easily be spoofed. To address this problem, Microsoft and Yahoo have defined email authentication protocols (Sender ID [12] and DomainKeys [25]) that can be use to verify whether a received email is authentic. If widely used, these solutions could help to prevent spam emails and, as a byproduct, decrease the number of email-based phishing attacks. Unfortunately, however, these protocols are currently not used by the majority of Internet users.

Well-known, academic, browser-integrated solutions to mitigate phishing attacks are SpoofGuard [3, 21] and PwdHash [18, 17]. SpoofGuard looks for phishing symptoms (such as obfuscated URLS) in web pages. PwdHash, in comparison, creates domain-specific passwords that are

rendered useless if they are submitted to another domain (e.g., a password for `www.onlinebank.com` will be different when submitted to `www.attacker.com`).

As discussed previously, AntiPhish [7, 8] is a system that keeps track of *where* sensitive information is being submitted to. Whenever sensitive information, which has been previously entered on one site, is transmitted to another site, a warning is raised. This is a problem when a user wants to deliberately reuse information on multiple sites, since the system will generate warnings for each site where data is reused. A solution that addresses this limitation of AntiPhish is DOMAntiPhish [16]. More precisely, DOMAntiPhish leverages a comparison between the DOM tree of the first page, where the information was originally entered, and the second page, where the information is reused. When the two pages are found to be similar, a phishing attack is signaled. Otherwise, the system assumes legitimate reuse of information. Unfortunately, the system cannot cope with DOM obfuscation attacks, where the attacker uses a different DOM to create a similar page. Furthermore, DOMAntiPhish is ineffective against phishing pages that use mostly images.

Dhamija et al. [4] proposed a solution that makes use of a so-called dynamic security skin on the user's browser. The technique allows a remote server to prove its identity in a way that is easy for humans to verify, but difficult for phishers to spoof. The disadvantage of this approach is that it requires support from the user. That is, the user needs to actively check for signs that indicate a spoofed page. There are two problems with this approach. First, users that are victimized by phishing attacks are unsophisticated, and they do not pay sufficient attention to the presented signs. Second, in a later study [5], Dhamija et al. have shown that more than 20% of the users do not take visual cues into consideration at all. Also, visual deception attacks can fool even the most sophisticated users.

The most popular and widely-deployed anti-phishing techniques are based on the use of blacklists. These blacklists store a set of phishing domains that the browser prevents the user from visiting. Microsoft has recently integrated a blacklist-based anti-phishing solution into its Internet Explorer (IE) 7 browser. Similar tools include Google Safe Browsing [19], NetCraft tool bar [15], eBay tool bar [6], and McAfee SiteAdvisor [11].

The approach that is closest to our work was presented in Liu et al.'s short paper [24]. The authors analyze and compare legitimate and phishing web pages to define metrics that can be used to detect a phishing page. They classify a web page as a phishing page when its visual similarity value is above a predefined threshold. The approach first decomposes the web pages into salient blocks according to "visual cues." Then, the visual similarity between two web pages is computed. A web page is considered a phishing page if the similarity to the legitimate web page is higher than a threshold. The main differences to our approach are the following: First, we do not need an initial list of legitimate pages. Instead, our system automatically selects the page to compare against based on the site on which a certain piece of information was initially entered. This allows our system to perform fewer comparisons and warn more aggressively about phishing pages. Second, we use a richer set of features to perform the visual comparison computation and perform our experimental evaluation on a larger dataset (the authors of [24] used only 8 phishing pages).

3 Our Approach

In the following subsection, we provide a high-level overview of how our system can be used to detect phishing pages. Then, we discuss in detail how we extract signatures from web pages, and how we use these signatures to compare two pages to determine their visual similarity.

3.1 Using the System

One possible application scenario for our system is to integrate the visual similarity detection scheme into the open-source tool AntiPhish [8, 16]. AntiPhish tracks the sensitive information of a user and generates warnings whenever the user attempts provide this information on a web site that is considered to be untrusted. It works in a fashion similar to a form-filler application. However, it not only remembers *what* information (i.e., a $\langle \mathrm{username}, \mathrm{password} \rangle$ pair) a user enters on a page, but it also stores *where* this information is sent to. Whenever a tracked piece of information is sent to a site that is not in the list of permitted web sites, AntiPhish intercepts the operation and raises an alert. Although simple, the approach is effective in preventing phishing attacks. Unfortunately, when a user decides to reuse the same $\langle \mathrm{username}, \mathrm{password} \rangle$ pair for accessing different online services, too many undesired warnings (i.e., false positives) are raised. By integrating the comparison technique into the existing AntiPhish solution, we can prevent AntiPhish from raising warnings for sites that are visually different. The underlying assumption is that a phishing page aims to mimic the appearance of the targeted, legitimate page. Thus, when two pages are similar, and the user is about to enter information associated with the first page on the suspicious, second page, an alert should be raised. When the two pages are different, it is unlikely that the second page tried to spoof the legitimate site, and thus, the information can be transmitted without a warning.

Of course, our technique can also be used in other application scenarios, as long as a baseline for the suspicious page is available. That is, we need to know what the legitimate page looks like so that we can compare against it. For example, the approach could be part of a security solution that works at the mail server level. Whenever a suspected phishing email is found, the potential phishing URL is extracted from the email. Then, the corresponding legitimate

page is obtained, using a search engine or, based on keywords, selecting among a predefined set of registered pages. Finally, a comparison is initiated and, if the outcome is positive, the email is blocked.

To compare a target page (i.e., suspected page) with a legitimate page, four steps are required:

1. Retrieve the suspicious web page w.
2. Transform the web page into a *signature* $S(w)$.
3. Compare $S(w)$ with the stored signature $S(\hat{w})$ of the supposed legitimate page $\hat{w}$ (i.e., the page targeted by the phishing page).
4. If the signatures are "too" similar, raise an alert.

Steps 2 and 3 represent the core of our technique. We discuss these steps in detail in the next two subsections. The actual implementation of Step 4 depends on the specific application scenario in which the approach is used. For example, in Antiphish, raising an alert implies that the submission of sensitive data is canceled and a warning is displayed to the user.

3.2 Signature Extraction

A signature $S(w)$ of a web page w is a quantitative way of capturing the information about the text and images that compose this web page. More precisely, it is a set of features that describe various aspects of a page. These features cover (i) each visible text section with its visual attributes, (ii) each visible image, and (iii) the overall visual look-and-feel (i.e., the larger composed image) of the web page visible in the viewport[1]. The following paragraphs describe in more detail the features that our system extracts from a web page.

Text elements. A text element is a visible piece of text on the web page that corresponds to a leaf text node in the HTML DOM tree. For each piece of text, we extract:

1. its textual content,
2. its foreground color,
3. its background color,
4. its font size,
5. the name of the corresponding font family, and
6. its position in the page (measured in pixel starting from the upper left corner).

Thus, for each text element, we obtain a 6-tuple t that we call *text tuple*. The signature of the page contains a vector $t_0, \ldots, t_n$ of k tuples, where each of the k tuples represents one visible piece of text on the web page.

[1]The viewport is the part of the web page that is visible in the browser window.

Image elements. For each visible image of the web page, our technique extracts:

1. the value of the corresponding `src` attribute (i.e., the source address of the image),
2. its area as the product of width and height, in pixel,
3. its color histograms,
4. its 2D Haar wavelet transformation, and
5. its position in the page.

Thus, for each image element on the page, we obtain a 5-tuple i that we call *image tuple*. The signature $S(w)$ contains a vector $i_0, \ldots, i_m$ of tuples, where each tuple represents one visible images on the web page.

To extract a color histogram from an image, we proceed as follows: First, we resize the image, obtaining a square image of $l \times l$ pixels. For the image elements, we set $l = 128$. In case the image width w and height h are both lower than l, we adjust the size to obtain a square image of $2^k \times 2^k$, where k is the greatest value for which $2^k \leq w$ and $2^k \leq h$. The resize operation leverages a built-in function in Firefox that is able to adjust images to fit into rectangles of arbitrary dimensions. It is done for performance reasons, so that the following computations can be executed faster.

Using the scaled image, the second step is to compute the histogram of the RGB components, using n histograms cells. For the image elements, $n = 5$. This is done for all three colors in the RGB color-space. For example, consider the red component and 5 histogram cells. We count the number of pixels whose red component value is in the range $[0, 50]$ (recall that each color channel has 8 bits) to obtain the first red histogram bar, the number of pixels whose red component value is in the range $[51, 101]$ for the second bar, and so on; finally, we normalize the bar values such that their sum is equal to 1.

The 2D Haar wavelet transformation [22] is an efficient and popular image analysis technique that, essentially, provides low-resolution information about the original image. It can be calculated in $O(n)$ time, where n is the size of a square, gray-scale image. The wavelet transformation operates on the scaled, square image, which is obtained as described previously. In a first step, this image is converted into a gray-scale version. Then, we compute the 2D Haar wavelet transformation and take the first $m \times m$ wavelet coefficients at the low-resolution end of the matrix. For image elements, we use a value of $m = 8$.

Overall appearance. Finally, we consider the overall image corresponding to the viewport of the web page as rendered by the user agent (i.e., the upper left portion of the rendered web page that fits a browser window maximized on a typical display – in our case, we used a screen resolution of 1280×800 pixels). For this image, we extract:

1. its color histograms and

2. its 2D Haar wavelet transformation.

For the overall appearance, we obtain a single pair o that we call *overall image tuple*, which represents the overall visual image of the web page. The color histogram and the wavelet transformation computation is performed in the same way as for the individual image elements. To capture the overall appearance image with higher precision, we compute the features on a larger image. That is, we use $l = 256$, meaning that the size of the scaled-down image is 256×256 pixels. Also, the number of cells for the color histogram is increased to $n = 8$, and we select $m = 16$ wavelet coefficients.

Page signature. Once we have extracted the features that capture the overall appearance of a page and each of its text and image elements, we can store this page's signature $S(w)$. The signature is simply the set of all text tuples, image tuples, and the overall image tuple: $S(w) = \langle t_0, \ldots, t_n, i_0, \ldots, i_m, o \rangle$.

3.3 Signature comparison

Once two signatures $S(w)$ and $S(\hat{w})$ are available, we can compute the similarity score between the corresponding web pages w and $\hat{w}$. To this end, we start by comparing pairs of elements from each page. Of course, elements are only compared with matching types (e.g., text elements are only compared with other text elements). That is, we compare all pairs of text elements to obtain a similarity score s^t. Then, we compare all image pairs to obtain a similarity score s^i. Finally, the overall appearances of the two pages are used to derive a similarity score s^o. Using these three scores, a single similarity score $s \in [0, 1]$ is derived that captures the similarity between the pages w and $\hat{w}$. The following paragraphs discuss in more detail how this similarity score is determined.

Text elements. Concerning the text elements, the comparison is done as follows. For each pair of text tuples t_i of $S(w)$ and $\hat{t}_j$ of $S(\hat{w})$, the following computation is performed:

- We compute the similarity between the two textual contents T and $\hat{T}$ as:

$$1 - \frac{d_l(T, \hat{T})}{\max(\text{length}(T)), \text{length}(\hat{T})}$$

  where $d_l(T, \hat{T})$ is the Levenshtein distance [10] between T and $\hat{T}$;

- We compute the similarity between the two foreground colors C and $\hat{C}$ as $1 - \frac{1}{(3 \cdot 255)} L_1(C, \hat{C})$, where L_1 is the 1-norm distance (also known as Taxicab metric or Manhattan distance [9]) between the colors expressed as 8-bit RGB points (i.e., $L_1(C, \hat{C}) = |r - \hat{r}| + |g - \hat{g}| + |b - \hat{b}|$);

- We compute the similarity between the two background colors, in the same way as above;

- We compute the similarity between the two font sizes F and $\hat{F}$, expressed in pixel, as $1 - \frac{|F - \hat{F}|}{\max(F, \hat{F})}$;

- We compute the similarity between the name of the two font families, setting 0 if they are equal, 1 otherwise;

- We compute the similarity between the two positions in the pages as $1 - \frac{d}{M_d}$, where d is the Euclidean distance between the two points and M_d is the maximum Euclidean distance between two points in a viewport of 1280×600 resolution;

Note that all the obtained similarities are in the range $[0, 1]$, where 0 means no similarity and 1 means total match. We then sum the 6 individual similarities scores, using the weights $\frac{4}{15}, \frac{4}{15}, \frac{2}{15}, \frac{2}{15}, \frac{2}{15}, \frac{1}{15}$ (whose sum is equal to 1), and obtain the similarity $s^t_{i,j}$ between t_i and $\hat{t}_j$.

The weights were manually chosen based on our domain knowledge and the assessment of the importance of the corresponding features to the visual similarity between two text blocks. These weights only serve as a rough estimate for the different impact of features. Clearly, the content of the text and the text color are more important than the used font family. This intuition is reflected by the weights. The actual values have not been optimized, and it might be possible to further improve our system by tuning the weights.

Once we have obtained a similarity score between all pairs of text elements, we store them in a similarity matrix S^t. This matrix stores, for each pair of elements t_i and $\hat{t}_j$, the similarity between these two elements. The dimension of the matrix is $n \times m$, where n is the number of text elements on page w and m is the number of text elements on $\hat{w}$.

As an example, suppose we are comparing the textual parts of two pages whose corresponding HTML documents are:

```
<html>
  <h1 style="color:rgb(255,0,0);">Home banking</
      h1>
  <p>Welcome!</p>
  <p>Copyright 2007</p>
</html>
```

and:

```
<html>
  <center>
    <h1 style="color:red;">Your banking</h1>
    <p style="color:gray;">Welcome!</p>
  </center>
</html>
```

The corresponding text tuples are, respectively:

$t_1 = \langle \text{Home banking}, (255,0,0), (255,255,255), 32, \text{Serif}, (8,8) \rangle$

$t_2 = \langle \text{Welcome!}, (0,0,0), (255,255,255), 16, \text{Serif}, (8,66) \rangle$

$t_3 = \langle \text{Copyright 2007}, (0,0,0), (255,255,255), 16, \text{Serif}, (8,102) \rangle$

and:

$\hat{t}_1 = \langle \text{Your banking}, (255,0,0), (255,255,255), 32, \text{Serif}, (8,21) \rangle$

$\hat{t}_2 = \langle \text{Welcome!}, (128,128,128), (255,255,255), 16, \text{Serif}, (8,80) \rangle$

The system first compares each pair of text elements, computing a similarity score. For the tuples t_1 and $\hat{t}_1$, this yields:

$$S^t_{1,1} = \frac{4}{15}0.75 + \frac{4}{15}1 + \frac{2}{15}1 + \frac{2}{15}1 + \frac{2}{15}1 + \frac{1}{15}0.98375 = 0.93225$$

When the computation is performed for each pair, the similarity matrix S^t can be determined. In this example, this is the following 3×2 matrix:

$$S^t = \begin{pmatrix} 0.9322500 & 0.5493813 \\ 0.5740278 & 0.8649771 \\ 0.6062897 & 0.5948105 \end{pmatrix} \quad (1)$$

Image elements. Concerning the image part, the comparison is done as follows. For each image tuple i_i of $S(w)$ and for each image tuple $\hat{i}_j$ of $S(\hat{w})$:

- We compute the similarity between the two `src` attributes using the Levenshtein distance, as above;
- We compute the similarity between the two image areas A and $\hat{A}$, expressed in pixels, as $1 - \frac{|A-\hat{A}|}{\max(A,\hat{A})}$;
- We compute the similarity between the two matrices representing the color histograms C and $\hat{C}$ as $1 - L_1(C, \hat{C})$, using the 1-norm distance;
- We compute the similarity between the two matrix's representing the 2D Haar wavelet transformations, using the 1-norm distance as above;
- We compute the similarity between the two positions in the pages, as described previously for text tuples.

Again, all similarities are in the range $[0,1]$. We then sum these similarity values, using the weights $\frac{4}{11}, \frac{2}{11}, \frac{2}{11}, \frac{2}{11}, \frac{1}{11}$. The result is the distance $s^i_{i,j}$ between i_i and $\hat{i_j}$. Based on these distance values, we derive a similarity matrix S^i that captures the similarity between the image elements of $S(w)$ and $S(\hat{w})$. Again, the weights are set according to our assessment of the visual impact of each feature.

Overall appearance. Finally, concerning the appearance of the overall image, we compute similarity in terms of color histograms and of 2D Haar wavelet transformations, in the same way as described previously for images. We obtain the similarity index s^o as the average of the two values.

Individual similarity scores. To obtain a similarity score s from a similarity matrix S (s^t for the text matrix S^t and s^i for the image matrix S^i), we average the largest n elements of the similarity matrix. The largest n elements of the matrix are selected using the following iterative, greedy algorithm: (i) we select the largest element of the matrix; (ii) we discard the column and the row of the selected element. We repeat these steps until a number n of elements are selected or the remaining matrix is composed of either no rows or no columns. We set $n = 10$ for the text similarity matrix and $n = 5$ for the image similarity matrix. In other words, we extract the n most matching items (either among text blocks or among images) between the two web pages under comparison, avoiding to consider an item more than once.

Consider the previous example in Equation 1, which shows the similarity matrix for the text elements of two small web pages. In this case, the algorithm first selects the top, left element of the matrix. This removes the first row and the first column from further consideration. Then, then second element in the second column is selected. The algorithm terminates at this point. As expected, the two text elements are paired that share the word "banking" and "welcome", respectively. Based on the two values, a similarity score of $\frac{0.9322500+0.8649771}{2} = 0.89861355$ is obtained.

We consider the average of the greatest n values in the matrix instead of considering the whole matrix because we wish to avoid the case in which the comparison outcome is influenced mainly by many non-similar elements rather than by few, very similar elements (which are typically the ones that can visually lure the user). For example, consider a phishing page in which there are very few images (e.g., the logo and a couple of buttons) that are very similar to the ones in the legitimate page. Also, imagine that there are a large number of graphical elements, possibly small and actually rendered outside of the viewport, which are not present in the original page; if we would take the average over all the matrix elements, the outcome would be biased by the low similarity among the many dissimilar element. However, the user would be tricked by the few elements that are very similar.

Final similarity score. The final outcome of a comparison between two signatures is the similarity score s. This score is obtained based on the individual scores for text and image elements, as well as the overall appearance: $s = a^t s^t + a^i s^i + a^o s^o$. When s is large, the two pages are similar. A threshold t is used in order to discriminate between the two cases: w and $\hat{w}$ are considered similar if and only if $s \geq t$, not similar otherwise. We discuss how we determine suitable values for the coefficients a^t, a^i, a^o as well as for the threshold t in Section 4.2.

4 Experimental Evaluation

In this section, we discuss the experiments that we performed to demonstrate the effectiveness of our system to recognize similarities between phishing pages and their targets (i.e., the legitimate pages that are spoofed).

4.1 Dataset

First, we compiled a dataset that consists of negative and positive pairs of web pages. For the positive pairs, we selected pairs of real-world legitimate pages and corresponding phishing pages. We obtained the phishing pages from the PhishTank public archive (`http://www.phishtank.com`). For each phishing page, we retrieved the corresponding legitimate page by visiting the web site of the spoofed organization immediately after the attack appeared on PhishTank. To build the negative part of the dataset, we collected a number of common web pages, unrelated to the legitimate ones.

We partitioned the set of positive pairs into three subsets, based on their visual similarity. That is, we defined three levels of dissimilarity as perceived by a human viewer who manually looks and compares a legitimate web page and the corresponding phishing page. We denoted each subset with a dissimilarity level label: Level 0 identifies pairs with a perfect or almost perfect visual match. Level 1 identifies pairs with some different element or with some minor difference in the layout. Level 2 identifies pairs with noticeable differences. Figure 1 shows a sample of 3 positive pairs that represent different dissimilarity levels.

We chose to partition positive pairs into different subsets for the following reason: The majority of phishing pages exactly mimic the appearance of the legitimate page. This is not surprising, as the miscreants do not wish to raise suspicion. However, there are also cases where visual differences do exist. These differences may be simply due to the poor skills of the attacker (e.g., mistakes in a text translated to a foreign language). However, some differences may be voluntarily inserted, both at the source level or at the rendering level. This could be done to evade anti-phishing systems, while, at the same time, keeping the look-and-feel as close to the original web page as possible. Note that similar evasion techniques are sometimes used by spammers for image-based spam [2, 23]. That is, although some randomized alterations are applied to the original image, from the user's point of view, the image remains identical.

We also partitioned the set of negative pairs into two subsets. One subset consists of web pages with a login form; the second one has no such forms. We selected the pages in the first subset mainly from Internet banking web sites. The second subset was chosen by performing a manual selection of pages, aimed at obtaining a heterogeneous and random sample set varying in size, layout, and content. We chose to include a substantial portion of pages with a login form to make the experiments more realistic and challenging. This is because pages that contain a login form are more likely to be compared against legitimate pages when trying to detect phishing pages.

The dataset was composed of 41 positive pairs (20 of Level 0, 14 of Level 1, and 7 of Level 2). We had 161 negative pairs (115 with and 46 without a login form).

4.2 Testing Methodology

Using our dataset, we built a *training set* by extracting a small portion of pairs of pages. This subset was used to tune coefficients and threshold values used in the computation of the final similarity score (as discussed in Section 3.3). The training set was composed of 14 positive pairs (9 of Level 0, and 5 of Level 1) and 21 negative pairs (15 with and 6 without a login form).

Using the training set, we performed an optimization for the coefficients a^t, a^i, a^o, and the threshold t, using the following objective function that was minimized:

$$f = \mathrm{FPR} + \mathrm{FNR} + \frac{\sum_k e_k}{n_p + n_n} \qquad (2)$$

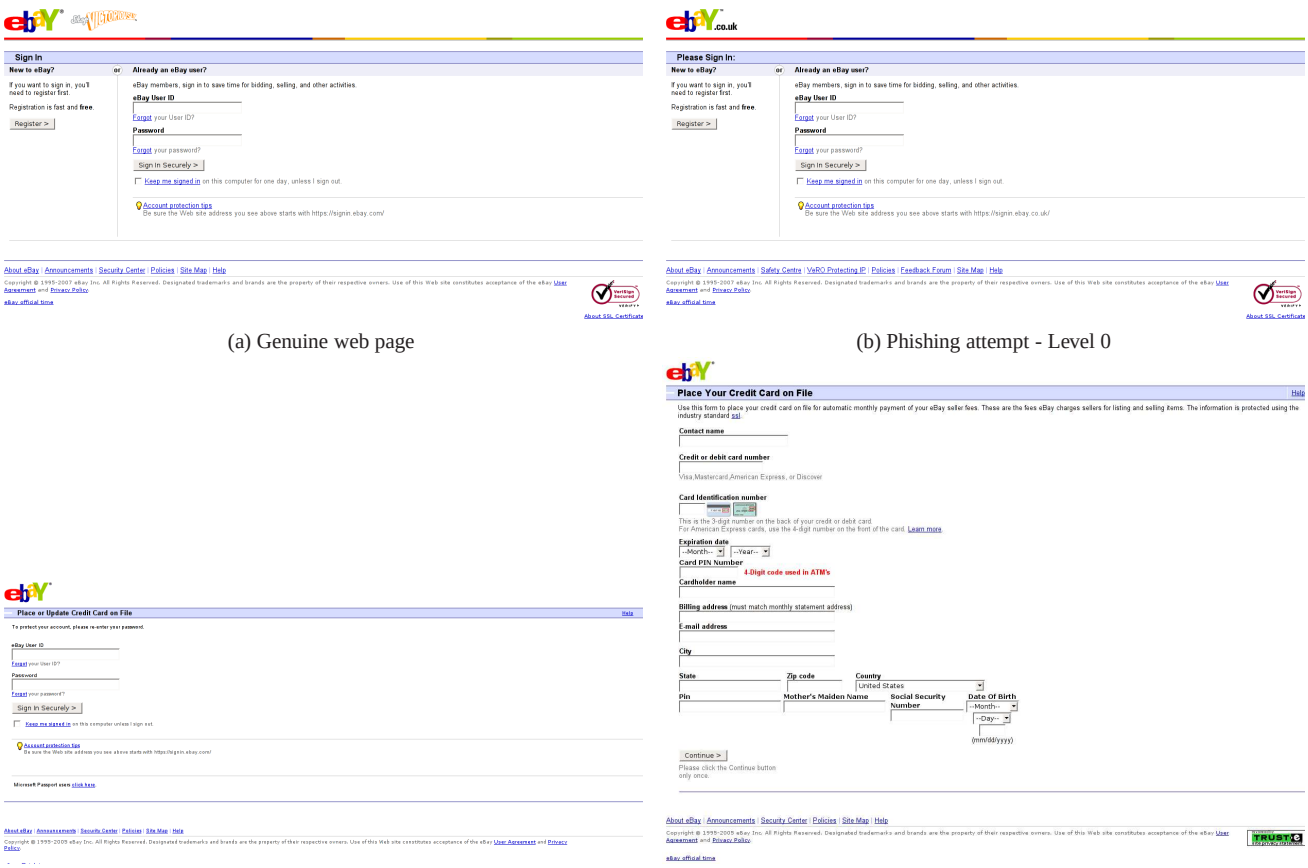
where n_p and n_n were the number of positive and negative pairs of the training set and e_k was a corrective error component. This corrective error component is computed for each reading of the training set as follows: $e_k = 0$ if and only if the reading was correctly evaluated (i.e., true positive or true negative), $e_k = |s - t|$ otherwise. We inserted this corrective error component to make the function f smoother and, hence, easier to minimize with the chosen algorithm. Otherwise, f would have been quantized as the dataset was finite.

To perform the optimization, we used an implementation of the simplex method [14]. The computation yielded the coefficient values $\langle a^t, a^i, a^o \rangle = \langle 2.11, 0.11, 1.20 \rangle$ and the threshold value $t = 0.956$. For these values, the objective function assumed the optimal value $f = 0$. That is, no false negatives or false positives were exhibited for these parameter settings.

4.3 Results

We then evaluated our approach using the parameter values computed as explained above over the remaining part of the dataset. For this experiment, the test set was composed of 27 positive pairs (11 of Level 0, 9 of Level 1, and 7 of Level 3) and 140 negative pairs (100 with and 40 without a login form).

Table 1 summarizes the results of the tests. It can be seen that our approach detects all phishing pages classified as Level 0 and 1, while it fails to detect two out of seven positive pairs of Level 2. Hence, we exhibit an overall false negative rate (FNR) equal to 7.4%. We verified, by visual



(a) Genuine web page

(b) Phishing attempt - Level 0

(c) Phishing attempt - Level 1

(d) Phishing attempt - Level 2

Figure 1: A sample of 3 positive pairs of different dissimilarity level.

Levels	t	f_p	n_n	FPR	f_n	n_p	FNR
All (0, 1 and 2)	0.956	0	140	0%	2	27	7.4%
Only 0 and 1	0.956	0	140	0%	0	20	0.0%
Only 0	0.956	0	140	0%	0	11	0.0%

Table 1: FPR and FNR for different datasets and t values.

inspection, that those two positive pairs were indeed difficult to detect by our visual-similarity-based approach. Figure 2 shows screenshots of one of the two undetected pairs; note that both the overall appearance and textual contents of the pages are significantly different. In comparison, all the positive pairs composed by pages shown in Figure 1 were correctly identified by our technique.

Also, Table 1 results show that our approach does not raise any false positive on the 140 negative pairs, which results in a false positive rate (FPR) of 0%. This includes all the negative pairs corresponding to pages containing a login form.

We also computed FPR and FNR for the same three dataset compositions for various values of the threshold t. Figures 3a and 3b show the results we obtained in the form of ROC curves (i.e., the plot of $1 - \text{FNR}$ as a function of FPR) for subsets of the test set including and not including, respectively, positive pairs of Level 2. One can see that there exists at least one threshold value for which our approach exhibits perfect behavior when considering a dataset which does not contain Level 2 positive pairs. The same information is shown in Figure 3c, where FPR and FNR are plotted as functions of the threshold t. This shows that our approach is robust to certain variations of t.

In general, the choice of t depends on the desired trade-off between possible false positives and possible false negatives. Hence, it depends on the context in which the proposed approach is supposed to be deployed. For example, if we implement the comparison approach as part of the AntiPhish tool, it may be preferable to select lower values for t. The reason is that when using the visual comparison component with AntiPhish, a large number of possible false positives is already filtered out, since the comparison is invoked only when a user's known credentials are about to be transmitted to an untrusted web site. Therefore, the comparison may be relaxed towards accepting more false positives (warnings) in favor of avoiding missed detections.

Finally, we measured the average computation time for comparing two pages. Note that such an operation involves both the signature extraction and the signature comparison

(a) Legitimate web page

(b) Phishing attempt - Level 2

Figure 2: One of the two missed positive pairs (Level 2). Note that the phishing page is visually significantly different than the legitimate one.

phase. In our experimental analysis, we focused on the comparison phase. The extraction phase consists mainly of retrieving information which, in practice, is already available to the browser that has rendered the page. Moreover, the legitimate page signature is typically extracted once at a previous point in time.

In our experiments, we found that it took about 3.8 seconds for positive pairs and about 11.2 seconds for negative pairs to be compared. These numbers were obtained on a dual AMD Opteron 64 with 8GB RAM running a Linux OS. Note that the comparison of two signature $S(w)$ and $S(\hat{w})$, as described in Section 3.3, requires a number of operations in the order of $m \cdot \hat{m}$, where m and $\hat{m}$ are the corresponding number of tuples. Clearly, waiting for more than 10 seconds for a single comparison is prohibitive. To address this problem, we implemented the following optimizations that result in a considerable reduction of computational costs.

The key idea to improve performance is to execute the comparison operations in an order such that the most expensive operations are executed last. In particular, during the similarity index computation (either s^t, for the text section, or s^i, for the images), we keep the n (with $n = 10$ for the text part and $n = 5$ for the image part) similarity values for the most similar pairs of tuples found so far. When evaluating a new pair of tuples, we can stop the evaluation once we determine that this pair cannot exceed the similarity values of one of the top n pairs, even if all remaining features comparisons yield perfect similarity.

For example, suppose that: (i) we are considering a pair of images, (ii) we have computed the distance in terms of positions on the page and image sizes, and (iii) the distances are such that with the given weights, the corresponding matrix element will be lower than the 5th greatest element of the matrix. In this case, we do not need to compute, nor extract, the two Haar transformations or the Levenshtein distances.

A similar optimization is performed based on the outcome of the overall image comparison. Once we determine that, after looking at the score for the overall appearance, two pages cannot exceed the similarity threshold t, then we do not need to compute any of the two similarity matrices for the text and image elements. This results in impressive speed-ups. A negative comparison between two pages is produced in a few milliseconds.

5 Conclusion

In this paper, we presented an effective and novel approach to detect phishing attempts by comparing the visual similarity between a suspicious page and the potential, legitimate target page. The proposed approach is inspired by two previous open source anti-phishing solutions: the AntiPhish browser plugin and its DOMAntiPhish extension. Our solution addresses the shortcomings of these approaches and aims to make these systems more effective.

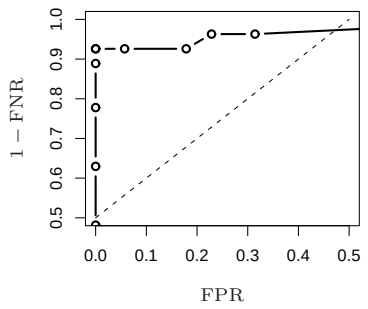
When checking for visual similarity, we consider three page features: text pieces, images embedded in the page, and the overall visual appearance of the web page as rendered by the browser. We consider features that are visually perceived by users because, as reported in literature, victims are typically convinced that they are visiting a legitimate page by judging the look-and-feel of a web site.

We performed an experimental evaluation of our comparison technique to assess its effectiveness in detecting phishing pages. We used a dataset containing 41 real phishing pages with their corresponding target legitimate pages. The results, in terms of false alarms and missed detection, are satisfactory. No false positives were raised and only two phishing attempts (that actually did not resemble the legitimate web page) were not detected.

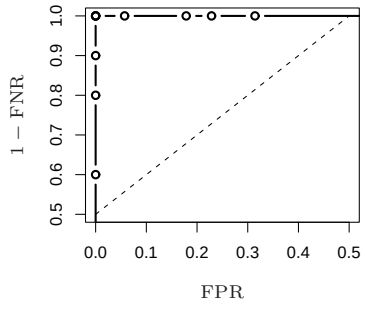
References

[1] APWG. Phishing Activity Trends - Report for the Month of December, 2007. Technical report, Anti Phishing Working Group, Jan. 2008. Available at `http://www.antiphishing.org/reports/apwg_report_dec_2007.pdf`.

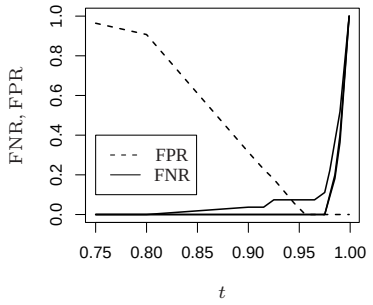
[2] H. Aradhye, G. Myers, and J. Herson. Image analysis for efficient categorization of image-based spam e-mail. *Document Analysis and Recognition, 2005. Proceedings. Eighth*



(a) All (Levels 0, 1, and 2)



(b) Levels 0 and 1



(c) False positive/negative rates vs. t

Figure 3: FPR and FNR with varying t. Figures 3a and 3b show ROC curves for different dataset compositions. In Figure 3c, the three solid lines represent FNR for different considered levels; the dashed line represents FPR.

International Conference on, 2:914–918, 29 Aug.-1 Sept. 2005.

[3] N. Chou, R. Ledesma, Y. Teraguchi, D. Boneh, and J. Mitchell. Client-side defense against web-based identity theft. In *11th Annual Network and Distributed System Security Symposium (NDSS '04), San Diego*, 2005.

[4] R. Dhamija and J. D. Tygar. The battle against phishing: Dynamic security skins. In *Proceedings of the 2005 symposium on Usable privacy and security, New York, NY*, pages 77–88. ACM Press, 2005.

[5] R. Dhamija, J. D. Tygar, and M. Hearst. Why Phishing Works. In *Proceedings of the Conference on Human Factors In Computing Systems (CHI) 2006, Montreal, Canada*. ACM Press, 2006.

[6] eBay. eBay tool bar. `http://pages.ebay.com/ebaytoolbar/`, 2007.

[7] E. Kirda and C. Kruegel. Protecting Users Against Phishing Attacks with AntiPhish. In *COMPSAC '05: Proceedings of the 29th Annual International Computer Software and Applications Conference (COMPSAC'05) Volume 1*, pages 517–524, Washington, DC, USA, 2005. IEEE Computer Society.

[8] E. Kirda and C. Kruegel. Protecting Users against Phishing Attacks. *The Computer Journal*, 2006.

[9] E. F. Krause. *Taxicab geometry*. 1987.

[10] V. I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady*, 10:707–710, 1966.

[11] McAfee. McAfee SiteAdvisor. `http://www.siteadvisor.com`, 2007.

[12] Microsoft. Sender ID Home Page. `http://www.microsoft.com/mscorp/safety/technologies/senderid/default.ms%px`, 2008.

[13] P. Mutton. Italian Bank's XSS Opportunity Seized by Fraudsters. Technical report, Netcraft, Jan. 2008. Available at `http://news.netcraft.com/archives/2008/01/08/italian_banks_xss_opportun%ity_seized_by_fraudsters.html`.

[14] J. Nelder and R. Mead. A simplex method for function minimization. *Computer Journal*, 7:308–313, 1965.

[15] NetCraft. Netcraft anti-phishing tool bar. `http://toolbar.netcraft.com`, 2007.

[16] A. Rosiello, E. Kirda, C. Kruegel, and F. Ferrandi. A Layout-Similarity-Based Approach for Detecting Phishing Pages. In *IEEE International Conference on Security and Privacy in Communication Networks (SecureComm)*, 2007.

[17] B. Ross, C. Jackson, N. Miyake, D. Boneh, and J. C. Mitchell. A Browser Plug-In Solution to the Unique Password Problem. `http://crypto.stanford.edu/PwdHash/`, 2005.

[18] B. Ross, C. Jackson, N. Miyake, D. Boneh, and J. C. Mitchell. Stronger Password Authentication Using Browser Extensions. In *14th Usenix Security Symposium*, 2005.

[19] F. Schneider, N. Provos, R. Moll, M. Chew, and B. Rakowski. Phishing Protection Design Documentation. `http://wiki.mozilla.org/Phishing_Protection:_Design_Documentation`, 2007.

[20] Sophos. Do-it-yourself phishing kits found on the internet, reveals Sophos. Technical report, Sophos, Aug. 2004. Available at `http://www.sophos.com/pressoffice/news/articles/2004/08/sa_diyphishing.%html`.

[21] SpoofGuard. Client-side defense against web-based identity theft. `http://crypto.stanford.edu/SpoofGuard/`, 2005.

[22] R. Stankovic and B. Falkowski. The Haar wavelet transform: its status and achievements. *Computers and Electrical Engineering*, 29:25–44, 2003.

[23] Z. Wang, W. Josephson, Q. Lv, M. Charikar, and K. Li. Filtering Image Spam with Near-Duplicate Detection. *Proceedings of CEAS 2007: Fourth Conference on Email and Anti-Spam*, Aug., 2007.

[24] L. Wenyin, G. Huang, L. Xiaoyue, Z. Min, and X. Deng. Detection of phishing webpages based on visual similarity. In *14th International Conference on World Wide Web (WWW): Special Interest Tracks and Posters*, 2005.

[25] Yahoo. Yahoo! AntiSpam Resource Center. `http://antispam.yahoo.com/domainkeys`, 2008.