A Layout-Similarity-Based Approach for Detecting Phishing Pages

Angelo P. E. Rosiello*, Engin Kirda‡, Christopher Kruegel‡, and Fabrizio Ferrandi*

*Politecnico di Milano
angelo@rosiello.org,ferrandi@elet.polimi.it
‡Secure Systems Lab, Technical University Vienna
{ek,chris}@seclab.tuwien.ac.at

Abstract

Phishing is a current social engineering attack that results in online identity theft. In a phishing attack, the attacker persuades the victim to reveal confidential information by using web site spoofing techniques. Typically, the captured information is then used to make an illegal economic profit by purchasing goods or undertaking online banking transactions. Although simple in nature, because of their effectiveness, phishing attacks still remain a great source of concern for organizations with online customer services.

In previous work, we have developed AntiPhish, a phishing protection system that prevents sensitive user information from being entered on phishing sites. The drawback is that this system requires cooperation from the user and occasionally raises false alarms. In this paper, we present an extension of our system (called DOMAntiPhish) that mitigates the shortcomings of our previous system. In particular, our novel approach leverages layout similarity information to distinguish between malicious and benign web pages. This makes it possible to reduce the involvement of the user and significantly reduces the false alarm rate. Our experimental evaluation demonstrates that our solution is feasible in practice.

1 Introduction

Online services have become important part of our lives as they allow anytime, anywhere access to information. Clearly, such services are not only useful for Internet users, but they have also become indispensable for financial organizations because they help reduce operational costs. For example, there are millions of users who use the Internet for performing online banking transactions. The web is convenient for users as they are not bound to the opening hours of banks and do not have to be physically present. Unfortunately, the usefulness of online services has been overshadowed by large-scale phishing attacks launched against Internet users. Phishing is a form of identity theft in which a combination of social engineering and web site spoofing techniques are used to trick a user into revealing confidential information with economic value.

In a typical phishing attack, a large number of spoofed e-mails are sent to random users (i.e., analogous to spam e-mail). These e-mails are disguised such that an unsuspecting victim is easily convinced that the e-mail is coming from a legitimate organization such as a bank. Typically, these e-mails requests the victims to "update" their online banking information. In order not to raise suspicion, the attackers have to provide a plausible explanation for the sudden need to update this confidential information. For example, early phishing e-mails often contained the explanation that the computer systems of the organization were being restructured. Hence, customers were supposedly being asked to "verify" that their information was correct. However, because phishing has received significant press coverage and attention in the last couple of years, ironically, phishers are now often persuading victims to enter their online banking credentials as a precaution for the imminent phishing threat.

In phishing e-mails, the request to update confidential information is often accompanied by a subtle threat in order to make the persuasion of the victim easier. For example, the phishers may convince victims that the failure to update their information will result in their banking account being suspended.

Obviously, not all Internet users who receive the phishing e-mail will be 1) naive enough to give away their confidential information and 2) customers of the

organization in question. However, because of the very large number of e-mails that are sent, there is a good chance that at least some receivers will be a customer of the targeted organization and that they will be tricked into giving away their confidential information. In fact, the increasing number of phishing attacks and the resulting financial damages is a good sign that phishing is currently an easy, profitable and effective attack.

According to the anti-phishing working group, the phishing problem has grown significantly over the last years. For example, the number of unique phishing web sites has grown from more that 7000 in December 2005 to more than 28,000 December 2006. According to Gartner Inc. [7], a remarkable $2.8 billion was lost to phishers last year. Clearly, effective anti-phishing solutions are required to mitigate phishing attacks.

In this paper, we present a novel, layout-similarity-based approach for detecting phishing pages. Our approach is based on our previous work on mitigating phishing attacks, AntiPhish [9], and significantly improves it. We call our new approach DOMAntiPhish.

The original AntiPhish is based on the premise that while a user may be easily fooled by URL obfuscation or a fake domain name, a program will not. AntiPhish is a browser plug-in that keeps track of the sensitive information that the user enters into web forms. Whenever a piece of sensitive information that is associated with one site is entered on another site, an alert is generated. For example, an alert is generated when AntiPhish detects that an online banking password (e.g., a password associated with the domain www.bank.com) is being typed into a text field that is not on the online banking site.

The key disadvantage of AntiPhish is that manual interaction is required to specify the information on a web site that is considered sensitive. That is, the user must manually associate a piece of sensitive information with a site (or domain). Furthermore, when the same password is used on multiple web sites, false alerts are generated. This happens because AntiPhish will consider the legitimate reuse of a password on a second site as a phishing attempt.

Our novel detection approach, DOMAntiPhish, leverages the original idea of AntiPhish. The system also associates sensitive information with sites and monitors which data is sent to which domain. The key difference happens when DOMAntiPhish detects that a password that is associated with a certain domain is reused on another domain. In this case, the system does not immediately raise an alert, but compares the layout of the current page with the page where the sensitive information was originally entered. For this comparison, the Document Object Model (DOM) of the original web page and the new page are checked. When the system determines that these pages have a similar appearance, a phishing attack is assumed (and the user is warned appropriately). The reason is that a phishing page aims to mimic the original page, and thus, their layouts are expected to be similar. When the layouts of the two pages are different, we assume that the password is reused on a legitimate page. The reason is that a phishing page needs to appear similar to the original page; otherwise, users cannot be tricked into revealing their sensitive information.

The paper is structured as follows: The next section reviews related work. Section 3 describes our approach and provides details about the implementation of our system. Section 4 presents the experimental results that show that our approach is feasible in practice. Section 5 discusses the limitations of our approach, while Section 6 concludes the paper.

2 Related Work

Quite a number of anti-phishing solutions have been proposed to date. These approaches can generally be classified into five main categories: E-mail-based, blacklist-based, visual-clue-based, website-feature-based, and information-flow-based approaches.

2.1 E-mail-based approaches

Some of the approaches attempt to eliminate the phishing problem at the e-mail level by trying to prevent phishing e-mails from reaching the potential victims. E-mail-based approaches typically use filters and content analysis and are, hence, closely related to anti-spam research. If trained regularly, for example, Bayesian filters are actually quite effective in also intercepting phishing e-mails. The downside of anti-spam techniques is that their efficiency often depends on regular training. Furthermore, anti-spam tools are not always ubiquitously available. Also, note that no matter how effective, anti-spam solutions are not perfect and a some phishing e-mails may still reach potential victims.

Microsoft and Yahoo have also defined e-mail authentication protocols (i.e., Sender ID [14] and DomainKeys [25]) that can be use to verify if a received e-mail is authentic. Although authentication protocols could solve the spam problem, they are currently not used by the majority of Internet users. Unfortunately, the large-scale deployment of authentication protocols would require the modification (or adaptation) of existing infrastructures.

2.2 Blacklist-based approaches

The most popular and widely-deployed anti-phishing techniques are based on the use of blacklists of phishing domains. For example, Microsoft has recently integrated a blacklist-based anti-phishing solution into its Internet Explorer (IE) 7 browser. The browser queries lists of blacklisted and whitelisted domains from Microsoft servers and makes sure that the user is not accessing any phishing sites.

Google Safe Browsing [20], analogous to IE 7, uses blacklists of phishing URLs to identify phishing sites. The disadvantage of the approach is that non-blacklisted phishing sites are not recognized.

In contrast, the NetCraft tool bar [16] assesses the phishing probability of a visited site by trying to determine how old the registered domain is. The approach uses a database of sites that are maintained by the company. Hence, new phishing sites that are not in the database might not be recognized. Similarly, SiteAdvisor [13] is a database-backed solution. It includes automated crawlers that browse web sites, perform tests, and create threat ratings for each visited site. Unfortunately, just like other blacklist or database-based solutions, SiteAdvisor cannot recognize new threats that have not been analyzed.

The main problem with crawling and blacklist proposals is that the anti-phishing organizations find themselves in a race against the attackers. Unfortunately, there is always a window of vulnerability during which users are susceptible to attacks. Furthermore, the approaches are only as effective as the quality of the lists that are maintained.

2.3 Visual-clue-based approaches

One interesting solution that has been proposed by Dhamija et al. [3] involves the 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 effort by the user. In fact, the user needs to be aware of the phishing threat and actively check for signs that the site she is visiting is spoofed. Note that in a later study [4], Dhamija et al. report that more than 20% of the users do not take visual clues into consideration when surfing and that visual deception attacks can fool even the most advanced users.

2.4 Website-feature-based approaches

A well-known academic solution in literature is SpoofGuard [1, 21]. SpoofGuard looks for phishing symptoms (e.g., obfuscated URLs, suspicious sentences) in web pages. Similarly, Internet Explorer 7 is reported to have simple anti-phishing functionality [12] where the browser analyzes the HTML structure of a web page to determine the probability that a certain page is phishing.

The eBay tool bar [6] solution is specifically designed for eBay and PayPal and involves the use of a so-called "Account Guard" that changes color if the user is on a spoofed site.

One proposed approach that is closely related to our work is presented in Lui 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 if its visual similarity value is above a predefined threshold. The approach first decomposes the web pages into salient blocks according to "visual cues." The visual similarity between two web pages is then measured. A web page is considered a phishing page if the similarity to the legitimate web page is higher than a threshold. The differences to our work are twofold. First, we analyze the layout similarity of two web pages by comparing the HTML tags of the pages and extracting and comparing regular subgraphs from their DOM representations. Second, we do not require an initial list of pages that are considered legitimate. Instead, our system automatically selects the page to compare against based on the site on which a certain piece of information was initially entered.

2.5 Information-flow-based approaches

PwdHash [19, 18] is a well-known anti-phishing solution in literature. It creates domain-specific passwords that are rendered useless if they are submitted to another domain (e.g., a password for `www.hotmail.com` will be different if submitted to `www.attacker.com`). In comparison, AntiPhish [9] takes a different approach and keeps track of where sensitive information is being submitted. That is, if it detects that confidential information such as a password is being entered into a form on an untrusted web site, a warning is generated and the pending operation is canceled. The main disadvantage of AntiPhish is that it requires user interaction to specify which sensitive information should be captured and monitored. The approach presented in this paper significantly improves the original idea of AntiPhish by eliminating the necessary user interaction with an extra comparison step that analyzes the DOM structure of the pages.

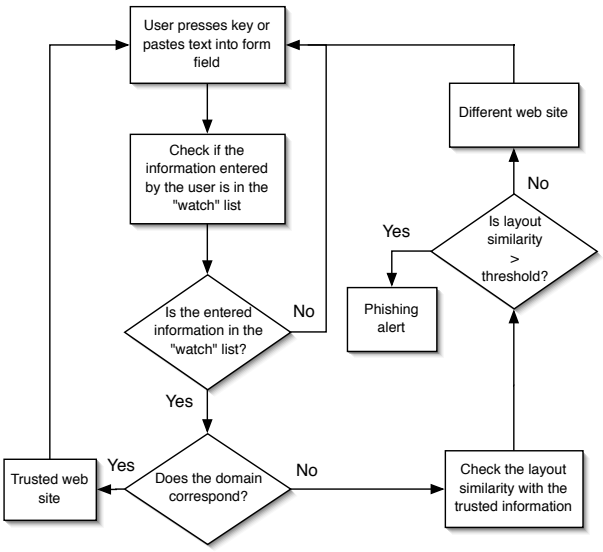

Figure 1. Flowchart showing how the sensitive information flow is controlled by the extended AntiPhish.

3 A Layout-Similarity-Based Approach

The approach presented in this paper leverages our previous work on the anti-phishing tool AntiPhish [9]. In the original AntiPhish system, a user can manually associate a piece of input with a certain domain. To this end, one first enters the desired value (such as a password) on a web page of this domain and then invokes AntiPhish to store the fact that this particular value is allowed to be sent to that domain. When the user later enters information into any form element of an HTML web page, the list of previously stored sensitive values is checked. For each value in this list that is identical to the one just entered by the user, the corresponding domain is checked. If the current site is not among these domains, a phishing attempt is assumed. The reason is that sensitive information is about to be transmitted to a domain that is not explicitly listed as trusted. If AntiPhish detects, for example, that the user has typed her online banking password into a text field on a web site that is not in the online banking web site domain, then it generates an alert and redirects to an information page about phishing attacks.

One of the significant limitations of the original system is that when a user is using the same piece of information on many different web sites, AntiPhish delivers incorrect phishing alerts (false positives). In this work, we added another layer to the control flow already supported by AntiPhish to mitigate the problem of reuse of sensitive information. More precisely, whenever the system detects that a piece of information is entered on a domain for which it has not been cleared, we do not immediately raise an alert. Instead, we check whether the current page is similar to the one on which the information has been entered originally. If the pages are similar, a phishing attempt is assumed. If the pages are different, we assume that a piece of information is legitimately reused.

After DOMAntiPhish is installed, every time the user successfully logs into a new web site, the browser will automatically store the hash of the entered password, using SHA-1, along with the DOM-Tree representation of the web site. That is, every time a password is entered, it is implicitly associated with the domain where it is used for the first time. This is in contrast to the old system, where passwords have to be explicitly and manually associated with domains. Whenever the password is reused, a similarity check then determines whether the reuse is legitimate (the pages are different) or a phishing attempt (the pages are similar).

In Figure 1, one can see the flowchart for DOMAntiPhish. After the domain check fails (the "No" branch is taken), the system computes the similarity between the current page and the page that is stored in the database. If this layout similarity is more than a predefined threshold value, then the web site is considered untrusted and an alert message is generated.

In the following paragraphs, we describe how we compute the similarity of two web sites by analyzing their DOM-Tree representations. In particular, we explore two methods: the first one is based on the simple tags comparison of the two web sites, while the second one is based on the identification of isomorphic subtrees.

3.1 DOM-Tree Extraction

The Document Object Model [23] (DOM)-Tree is an internal representation used by browsers to represent a web page. In order to extract the DOM-Trees of web pages, we developed an extension (i.e., plugin) for Firefox. An example of how a DOM-Tree is built over an HTML web page is shown in Figure 2. For every web site where a form is used to enter sensitive information successfully, its DOM-Tree data structure is associated within the "watch" list described in [9].

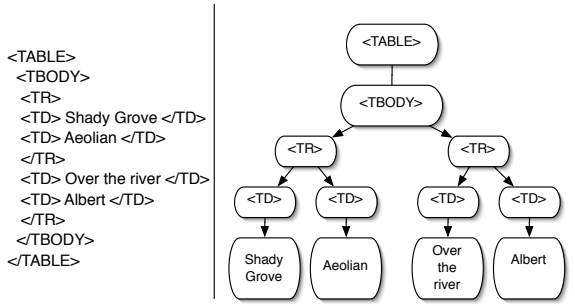

Figure 2. Example of a DOM-Tree representation.

3.2 Similarity Assessment

The layout similarity of two web sites is calculated considering their associated DOM-Tree representation. We start from the assumption that if two web sites have the same DOM-Tree, then they must produce an identical layout. It is still possible that two web sites with two different DOM-Trees could render the same layout, but if this is the case then:

- The administrator of the web site has done some structural modification.
- The web site was "copied" to look like the original web site, i.e., we are on a phishing web site.

Only in the second case we advise the user about the possibility of a phishing attack, since in the first case the domain of the web site is already in the trusted list.

Given two DOM-Trees, we compare their similarity in two different ways: 1) comparing the tags of the two web pages; 2) extracting regular subgraphs from the trees. Since the first approach is straight forward, in the following, we focus on the latter technique.

If regularity exists in a graph, it must be possible to identify a relevant number of templates and their instances in the examined graph. *Templates* represent particular subgraphs of the original graph with at least two instances. The generic regularity extraction problem consists of identifying large existing templates with many instances and covering the graph by the identified templates. Note that this problem was addressed in many different research areas such as in CAD circuit design [2, 10, 11] or graph mining [5]. For our purposes, regularity extraction consists of identifying the templates with the maximum number of vertices that have got at least two separate instances in the true and the phishing DOM-Trees, allowing some structural differences of the trees.

In the following paragraph, the regularity extraction problem formulation will be described in detail.

3.3 Formulation and Application of the Regularity Extraction Problem

The input for the regularity extraction problem are two DOM-Trees T and T' representing the true and the phishing web page. A tree is a graph $G(V, E)$ in which any two vertices are connected by exactly one path. Trees are used as a model for representing the structure of web sites, where the set of tags of a web site corresponds to the set of vertices V of a tree, while the hierarchy among the tags is modeled by the set of edges $E \subset VxV$. DOM-Trees also present attributes for each tag/vertex that can be considered as particular vertices of the graph.

As stated above, each vertex of the trees represents a tag of the web page and we attach to every vertex of the trees a unique label to distinguish them.

In order to polynomially bind the complexity of the templates generation phase, as also assumed by [2], we assign for every vertex $v \in G$ by a function $k : E \rightarrow 1, \ldots, k_f$ a unique index $1 < i < f$ to each outgoing edge from nodes $u_1, .., u_f$, directly in the formulation of the problem. In this way the permutations of outgoing edges are not considered.

A subgraph $G_i(V_i, E_i)$ of a graph $G(V, E)$ is *consistent* if and only if $V_i \subset V$, $E_i \subset E$ and G_i does not include disconnected subgraphs.

Two subgraphs G_i and G_j are *equivalent* if and only if:

- they are *isomorphic* [8].
- types of the corresponding vertices are the same.
- indices of corresponding edges are the same.

A *template* represents the equivalence class of the just described relation among two or more subgraphs. For our purposes, we allow a *weak equivalence* definition only considering as very strong condition the one that asserts the equivalence of the types of the vertices, while adding a penalty (i.e., the similarity penalty) if the other remaining two conditions are not completely respected.

Now that consistency, weak equivalence, and template notions were formalized, we define the *template generation*, *regularity extraction* and *coverage* problems:

Definition 3.1 (Template Generation Problem) *Given two DOM-Trees $T(V, E)$ and $T'(V', E')$ find all the equivalent or weak equivalent pairs of subgraphs in*

T and T' that are not completely included in any other subgraph of T and T' respectively.

Definition 3.2 (Regularity Extraction Problem) *Given two DOM-Trees $T(V, E)$ and $T'(V', E')$, find the set of templates Ω to cover T and T', where the number of vertices in $T_i \in \Omega$ are maximized and the similarity penalty is minimized.*

Definition 3.3 (Graph Covering Problem) *Given a graph $G(V, E)$ and a set of templates Ω, find a cover of G such that $\forall\ T_i \in \Omega$, the number of vertices T_i is maximized and the total similarity penalty is minimized.*

To cover the graphs, given the set of templates, we heuristically solve the problem by choosing at each iteration a template and removing its instances in the true and phishing DOM-Trees. The template chosen follows a main criterion: *BMFF*, i.e., Best-Match-Fit-First, where at each iteration of the extracting process, the template with the maximum number of vertices and the minimum penalty is selected.

The layout similarity of the true and the phishing web site is defined as the ratio of the weighted number of matched vertices of the DOM-Trees to the number of total vertices in the true web page, as shown in Equation 1.

$$\Gamma = \sum_{i=0}^{i=V} \frac{W(V_i)}{V_i} \tag{1}$$

where W is a function that assigns a similarity weight between 0 and 1 to each vertex of the true DOM-Tree, while V_i represents the *i-th* vertex of the true DOM-Tree.

Two web pages are considered similar if and only if the layout similarity value Γ, as defined in (1), exceeds a certain threshold δ, i.e.: $\sum_{i=0}^{i=V} W(V_i)/V_i > \delta$, where the value of the threshold δ can be found empirically and modified by the user of the application.

The procedure to automatically identify the layout similarity between two DOM-Trees consists of three main steps:

- Initialization.
- Templates Computation.
- Coverage.

In the following paragraphs the just above introduce steps will be described in detail.

3.3.1 Initialization

In the initialization phase, all the possible compatible pairs of vertices from the true and the phishing DOM-Trees are extracted and stored in a map. The map will contain the seeds for the next templates computation phase. Two tags are considered compatible if and only if they have the same *tagName* (i.e., the same type).

For example, if we consider the *HTML* source code of the true and phishing web pages shown in Table 3, we will obtain the following pairs of vertices from their DOM-Trees: *(HTML,HTML); (HTML;BODY); (HTML;BR); (HTML;BR); (BODY;HTML); (BODY,BODY); (BODY;BR); (BODY;BR)*. Only two pairs are compatible: $(HTML; HTML); (BODY; BODY)$. These pairs are stored in the compatibility map. Note that this map can contain at most $O(V * U)$ compatible pairs of vertices, where V is the number of vertices of the first tree and U in the second one.

Tags/vertices that do not impact on the layout of a web site, such as $META$ tags, are not considered during the matching process.

3.3.2 Templates Computation

The templates computation step is based on the approach proposed in [2] to identify equivalent trees in a graph. The algorithm starts from the seeds computed in the initialization phase and proceeds comparing the children vertices of each root iteratively, until compatible vertices are found. It is important to notice that a template is a representative sample of a set of isomorphic trees, eventually considering some penalty.

For each root with a different number of children with respect to its counterpart in the compatibility list, or with different types of children, or with different attributes attached to the vertices, we add a similarity penalty using the weight function W given in Equation 1. The penalty associated with each vertex and attribute represents its importance to the total layout similarity and is assigned empirically.

The templates computation algorithm is shown in Table 1. Let V be the set of vertices of the true DOM-Tree and U the set of vertices of the phishing DOM-Tree, the computational complexity of the proposed algorithm is $O(|V|)$ if $|V| > |U|$ else $O(|U|)$, since it consists in just visiting and comparing the vertices of the true and of the phishing DOM-Tree.

Let us consider the example in Figure 3b to demonstrate how the templates are computed. The compatibility map (from the initialization phase) contains the two pairs of seeds: $(HTML; HTML); (BODY; BODY)$. For the pair

Templates Computation Algorithm
INPUTS: vertex v, vertex u, firstSubTree *empty*, secondSubTree *empty*

WHILE continue_while ∃ equivalent_subTrees_branches **DO**
firstSubTree = getSubTree(u, firstSubTree);
secondSubTree = getSubTree(v, secondSubTree);
IF are_similar(firstSubTree, secondSubTree) **THEN**
float penalty=compute_similarity_penalty( );
store_subTrees(u, v, firstSubTree, secondSubTree, penalty);
END IF
END WHILE

Table 1. Templates computation algorithm.

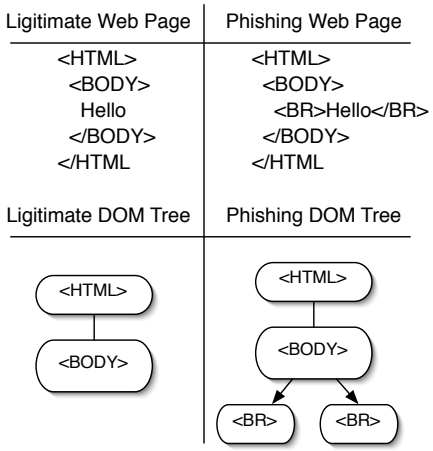

Figure 3. HTML code of two web pages and their DOM-Tree representations.

(HTML;HTML), before considering the children, i.e., $(BODY; BODY)$, we compare the attributes of the vertices. If they are found different, an additional penalty is added. Since no attributes are found for the vertex $< HTML >$, the pair of children (BODY;BODY) is checked and gets inserted into the template, since they are equivalent. No attributes are found, therefore, the children of $(BODY; BODY)$ should be compared. In this case, for the true DOM-Tree, no children are found, while there are two children for the phishing DOM-Tree (i.e., $< BR >, < BR >$) . Thus, a similarity penalty is added. The templates computation algorithm proceeds comparing the second pair of vertices $< BODY, BODY >$ in the same way as shown for the pair $< HTML, HTML >$.

All the computed templates are stored in a map (*templates_map*), which will be analyzed to extract the best fitting template during the coverage step.

3.3.3 Coverage

After every pair of compatible vertices is considered, and a tree is associated with them, the template with the maximum number of vertices and the minimum penalty is extracted. We called this greedy heuristic coverage criterion *Best-Match-Fit-First*. The vertices that are part of the extracted template will cover the tags of both the true and the phishing DOM-Trees.

For example, for the DOM-Trees in Figure 3b, the *Best-Match-Fit-First* heuristic criterion selects (from the *templates_map* computed in the templates computation phase) the tree $\{HTML; BODY\}$. This tree completely covers the vertices of the true DOM-Tree, while leaving uncovered two vertices in the phishing DOM-Tree ($\{BR\}$,$\{BR\}$).

The coverage phase is repeated until or no more template is extracted or the true or phishing DOM-Tree is completely covered.

Threshold	SubTrees FP [%]	SubTrees FN [%]	Tags FP [%]	Tags FN [%]
0	100	0	100	0
0,1	87.32	0	90.86	0
0,2	63.38	0	75.96	0
0,3	46.48	0	55.29	0
0,4	31.45	0	40.86	0
0,5	16.90	0	30.29	0
0,6	7.51	0,03	18.27	0
0,7	0	18,42	12.5	0
0,8	0	39,47	5.29	21.05
0,9	0	73,68	0.48	50
1	0	100	0	100

Table 2. Different threshold values versus percentage of false positive and false negative.

Simple Tags Comparison Algorithm

INPUTS: $vector < tags > u$, $vector < tags > v$

```
int coveredTags=0;
FOR ( int i = 0; i < u.size(); i++ )
 FOR ( int j = 0; j < v.size(); j++ )
  IF ( u[i] == v[j] ) THEN
     coveredTags++;
     v.erase ( j );
     break;
  END IF
 END FOR
END FOR
```

Table 3. Simple tags comparison algorithm.

3.4 Simple Tags Comparison Method

Another way to estimate the layout similarity between two web pages does not consider the global structure of the DOM-Trees, i.e., the subtrees, neglecting the connections among the tags. In this case, every tag of the original web page is compared with every tag of the potential phishing web page. If a match is found, then the identified pair of tags is covered. The routine is repeated until every tag of the original or potentially phishing web page is covered, or no more matches are found. The algorithm is reported in Table 3.

The similarity value is computed using the function Γ defined in Section 3.3, but in this case, the weight function W is the constant $'1'$, i.e., no penalty is considered at all. This approach is more efficient than the one proposed in Section 3.3, but also less effective in terms of false positives, as proved by experimental results in Section 4.

3.5 Implementation Details

We implemented the similarity layout assessment as a Mozilla browser extension [15] (i.e., a plug-in) using the Mozilla XML User-Interface language (XUL) [17], Javascript and Java. Our implementation is based on the implementation of AntiPhish described in [9].

If the domain control in the flow-chart illustrated in Figure 1 fails, the browser invokes a Java application to compute the value of layout similarity. In this case, there are two possibilities:

1. The user has the same password for two different web sites.
2. The user is on a phishing web site.

Note that AntiPhish [9] generates an alert in *both* cases described above. In contrast, our extension raises an alert if and only if the layout similarity value exceeds a given threshold δ. In this way, we remove the false alarms in the (common) cases in which a user is reusing the same password on different web sites.

As a good start value for the similarity threshold, we chose $\delta = 0.5$ based on our experimental results reported in Section 4. A higher similarity threshold is dangerous since the attacker could probably be able to imitate the legitimate web site by changing the DOM-Tree representation of the spoofed web site.

4 Experimental Results

We tested DOMAntiPhish on a set of phishing web sites on `phishtank.com`. This well-known site collects real phishing pages. During the similarity computation process, for the isomorphic subtrees identification algorithm, we added a penalty of 0.3 if two corresponding tags had different types or if a tag did not have

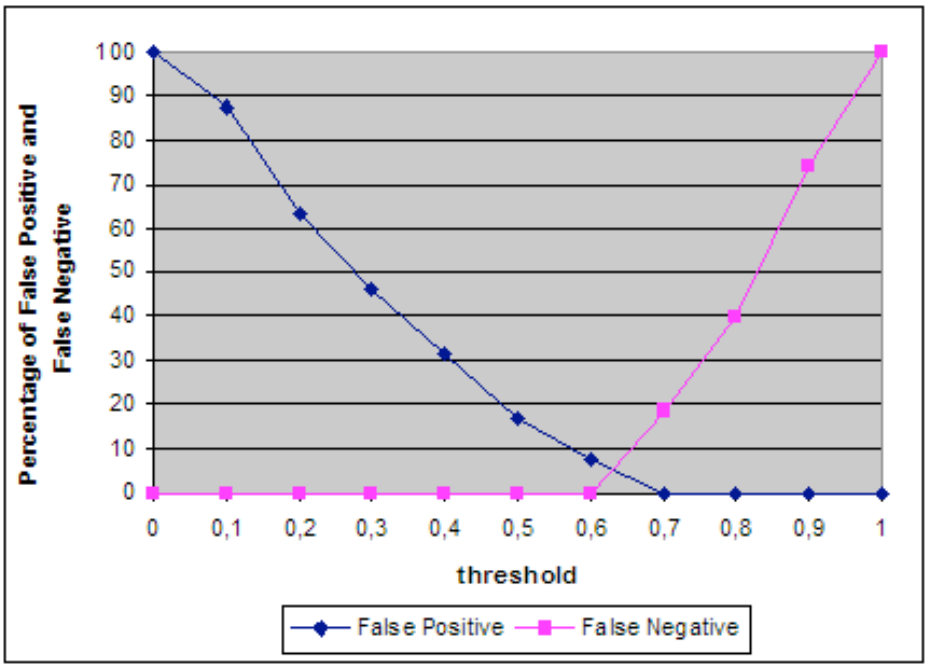

Figure 4. Percentage of false positive and false negative for different threshold values.

children and its matched counterpart did. If two attributes of matched tags were different, a penalty of 0.1 was added. Moreover, if the attributes had different values, then a penalty of 0.05 was added, too. The penalty values were determined empirically by having as objective function the minimization of false positive and negative results for low and high threshold values respectively.

In Table 2, we report the percentage of false positives (FP) and false negatives (FN) for different threshold values obtained executing both the discussed methods (i.e., isomorphic subtrees identification and simple tags comparison) on over two hundred different web sites (randomly obtained using [22]). The results are strongly correlated to the layout characterization of the legitimate web pages. In fact, if the legitimate web page contains many particular elements in the DOM-Tree, it is easier to distinguish it from another web page. Table 2 shows that by choosing $\delta = 0.5$ as a threshold value, all the phishing web sites are correctly identified with an acceptable percentage of false positives (i.e., 16.90% using the isomorphic subtrees identification algorithm and 30.29% using the simple tags comparison approach).

5 Limitations

One limitation of our current approach is that it could be possible for attackers to use a combination of images to create a spoofed web page that looks visually similar to a legitimate web page. Hence, the DOM of the spoofed web page would be different and detection would be evaded. One possibility of dealing with this limitation could be to take a conservative approach and to tag web pages as being suspicious that contain a large number of images or that mainly consist of images.

Another possible problem could be DOM obfuscation attempts that would make the visual look similar to the legitimate web page while at the same time evading detection. Note, however, that our approach raises the difficulty bar for creating phishing pages. Furthermore, one can always take a more conservative approach by reducing the phishing alert threshold. Also, if phishers are forced to alter the look and feel of their phishing pages, these pages will become less convincing and more suspicious to the victims.

6 Conclusion

Phishing is an important problem that results in identity theft. Although simple, phishing attacks are

highly effective and have caused billions of dollars of damage in the last couple of years. Hence, phishing attacks are still and important problem and solutions are required.

A number of industrial and academic anti-phishing solutions have been proposed to date to mitigate phishing attacks. Unfortunately, all of these solutions have important shortcomings. In this paper, we presented an automated, client-side (i.e., browser plugin-based) solution to protect naive, technically unsophisticated Internet users against phishing attacks. Our approach makes DOM-based layout comparisons of legitimate sites with potential phishing sites to detect phishing pages. Our experimental evaluation demonstrates that our solution is feasible in practice.

Acknowledgements

This work was supported by the Austrian Science Foundation (FWF) under grants P18368 (Omnis) and P18764 (Web-Defense), and by the Secure Business Austria competence center.

References

[1] 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.

[2] A. Chowdhary, S. Kale, P. Saripella, N. Sehgal, and R. Gupta. A general approach for regularity extraction in datapath circuits. *IEEE Transactions on CAD*, 18(9), 1999.

[3] 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.

[4] 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.

[5] S. Djoko, D. J. Cook, and L. B. Holder. Analyzing the Benefits of Domain Knowledge in Substructure Discovery. In *Proceedings of the First International Conference on Knowledge Discovery and Data Mining*, pages 75–80, 1995.

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

[7] Gartner Press Release. Gartner Says Number of Phishing E-Mails Sent to U.S. Adults eearly Doubles in Just Two Years . `http://www.gartner.com/it/page.jsp?id=498245`, 2006.

[8] F. Harary. *Graph Theory*. Addison-Wesley, 1969.

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

[10] T. Kutzenschebauch, , and L. Stok. Regularity Driven Logic Synthesis. In *Proceedings of the International Conference on Computer-Aided Design*, pages 439–446, 2000.

[11] T. Kutzschebauch. Efficient Logic Optimization Using Regularity Extraction. In *Proceedings of the International Conference on Computer Design*, pages 487–493, 2000.

[12] C. Ludl, S. McAllister, E. Kirda, and C. Kruegel. On the Effectiveness of Techniques to Detect Phishing Sites. In *Detection of Intrusions and Malware and Vulnerability Assessment (DIMVA) 2007 Conference, Lucerne, Switzerland*. Springer LNCS, July 2007.

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

[14] Microsoft. Sender ID Home Page. `http://www.microsoft.com/mscorp/safety/technologies/senderid/default.ms%px`, 2007.

[15] Mozilla Extensions. Home Page. http://update.mozilla.org/extensions/, 2005.

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

[17] Nick Dikean. XULTU Tutorial. http://www.xulplanet.com/, 2005.

[18] 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.

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

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

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

[22] URoulette. Home Page. http://www.uroulette.com, 2007.

[23] W3C. Document Object Model. `http://www.w3.org/TR/2004/REC-DOM-Level-3-Core-20040407/core.html`, 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`, 2007.