Trustworthiness testing of phishing websites: A behavior model-based approach

Hossain Shahriar *, Mohammad Zulkernine

School of Computing, Queen’s University, Kingston, Canada

A R T I C L E   I N   F O

Article history:
Received 28 October 2010
Received in revised form 3 January 2011
Accepted 3 February 2011
Available online xxxx

Keywords:
Phishing
Trustworthiness testing
Behavior model
Heuristic coverage

A B S T R A C T

Phishing attacks allure website users to visit fake web pages and provide their personal information. However, testing of phishing websites is challenging. Unlike traditional web-based program testing, we do not know the response of form submissions in advance. There exists lack of efforts to help anti-phishing professionals who manually verify a reported phishing site and take further actions. Moreover, current tools cannot detect phishing attacks that leverage vulnerabilities in trusted websites such as cross site scripting. An attacker might generate input forms by injecting script code and steal credentials. To address these challenges, we propose testing suspected phishing websites based on trustworthiness testing approach. In a trustworthiness testing, a website is not tested against a set of known inputs and matched the expected outputs with the actual ones. Rather, we check whether the behavior (response) of websites matches with our knowledge of phishing or legitimate website behaviors to decide whether a website is phishing or legitimate. We consider a suspected website as a web-based program and test the program based on a behavior model. The model is described using the notion of Finite State Machine (FSM) that captures the submission of forms with random inputs and the corresponding responses. We then identify a number of heuristics followed by a set of heuristic combination to assist a tester deciding whether websites are phishing or legitimate based on their up-to-date behaviors. We implement a tool named PhishTester to automate the testing process. We evaluate the proposed approach with both phishing and legitimate websites. The results show that the approach incurs zero false negatives and positives in detecting phishing and legitimate websites, respectively. Moreover, our approach can detect advanced XSS-based attacks that many contemporary tools currently fail to detect.

© 2011 Elsevier B.V. All rights reserved.

1. Introduction

Phishing is a web-based attack that allures end users to visit fraudulent websites and give away personal information (e.g., user id, password). The stolen information is the beginning point of many illegitimate activities such as online money laundering [1]. Phishing attacks cost billions of dollars in losses to business organizations and end users [2]. The attack jeopardizes the prospects of e-commerce industries. Therefore, addressing phishing attacks is important.

There are two main activities performed by phishers to make an attack successful. They are (i) developing fraudulent websites, and (ii) motivating (or urging) users to visit those sites. The fake websites have similar look and feel of legitimate websites, which are owned by organizations such as banks, credit unions, and governments. Phishers download pages of legitimate websites and modify some parts of these pages. In particular, they modify the pages that contain forms to be filled out by end users. The modification results in sending user provided information to repositories accessible by attackers. The mechanism that invites users to visit fraudulent sites is based on email messages. These emails urge or prompt users to take immediate actions to avoid consequences such as bank account suspensions.

A number of approaches have been developed in recent years to combat phishing attacks. These include detecting suspicious websites with heuristics [3,4], educating and training users [5], compiling white lists [6,7] and blacklists [8], filtering emails [9,10], and customizing visual cues to distinguish legitimate websites from fake websites [11]. Most browsers (e.g., Firefox, Internet Explorer) have built-in phishing attack detection abilities based on white and blacklisted websites. However, there exists no testing approach for anti-phishing professionals, who manually verify suspected websites, and inform administrators to take down the fake sites. Moreover, phishers may exploit cross site scripting (XSS) vulnerabilities by injecting JavaScript code to generate HTML forms [12] and frames containing input forms [13,14]. Unfortunately, traditional anti-phishing solutions cannot detect these sophisticated attacks. This situation motivates us to devise a testing approach for phishing website detection.

0167-739X/S – see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.future.2011.02.001
Traditionally, testing is intended to compare an actual output with the expected output for a given input under the assumption that inputs and (or) expected outputs are known in advance. Most of the existing works have addressed achieving program code coverage and fault detection capabilities of test suites when the assumptions hold true [15,16]. However, there exist unique circumstances where the inputs or expected outputs are not known in advance. Testing is challenging for these “non-testable” programs [17]. In testing literature, a number of works have addressed these situations through trustworthiness testing of programs instead of testing program correctness (i.e., providing known inputs and matching with known outputs) [18–20]. Here, trustworthiness testing is not intended to detect whether the output for a given input is correct or not. Rather, it assesses whether outputs conform to some attributes (e.g., behavior) of programs and environments which can be used as a basis to believe that a program is implemented based on what it is intended to perform.

Unfortunately, testing of a suspected phishing website is a challenging task. First, we have no clue on the expected response of a form submission unless we provide inputs to a website. Second, due to the lack of specification or program code, we do not know what should be test coverage criteria to detect phishing or legitimate websites. We believe that these features of suspected phishing websites match with the notion of trustworthiness testing [18]. Thus, we are motivated to address the testing of suspected websites through trustworthiness testing. Our objective is not to test suspected websites with known and valid inputs to verify expected outputs. Rather, we test websites with random inputs whether the behavior of suspected websites match with known symptoms of phishing or legitimate websites. The process can assist a tester to decide if a suspected website is phishing or legitimate. Nevertheless, two major subsequent issues still remain: (i) how to match observed behaviors with known behaviors? (ii) what features of website behaviors (responses) should be considered as the basis of conclusion that a suspected website is either phishing or legitimate?

In this paper, we apply trustworthiness testing of suspected phishing websites to answer these issues. We first develop a Finite State Machine (FSM) model based on known phishing and legitimate website behaviors. The model is developed in a manner so that it can distinguish between phishing and legitimate websites with respect to form submissions with random inputs. We then develop a number of heuristics based on the FSM model states as well as response page features (e.g., presence of a common form that has been submitted before, displaying supplied inputs in response pages). We also develop a set of heuristic combinations that capture the most up-to-date behaviors of suspected websites. These combinations can be used to test both phishing and real websites. If no known combination of heuristic is satisfied, a manual checking is required for the final decision. Note that heuristics and heuristic combination are dynamically determined, and they cannot be compared to the meaning of traditional test coverage criteria (i.e., how much program artifact is being covered by a given test suite).

Our approach does not depend on any updated black or white lists. Moreover, it is independent of the language and textual contents of websites. Many traditional phishing website detection tools are good at analyzing pages that contain English texts [4]. We implement and evaluate a prototype tool to automate the testing of suspected phishing websites named PhishTester. We evaluate our approach with both phishing and legitimate websites. The results show that the approach results in zero false positive and negative warnings and incurs negligible manual effort. Moreover, our approach can detect advanced XSS-based phishing attacks that many existing tools fail to detect.

The paper is organized as follows: Section 2 discusses an overview of phishing attack methods. Section 3 describes related works on trustworthiness testing, phishing website detection, and testing of web-based programs using formal behavior models. In Section 4, we describe the proposed behavior model along with heuristics. Section 5 discusses the implementation and evaluation results of PhishTester. Finally, Section 6 draws some conclusions and discusses future work.

2. Phishing technique

Phishers apply a wide range of techniques that leverage the specification of Hyper Text Markup Language (HTML) and rich features of Document Object Model (DOM). We divide attack techniques into two categories: spoofing the website contents and leveraging the DOM-based features. They are discussed in the following two sub-sections.

2.1. Spoofing website content

Attackers hide the legitimate content of web pages so that victims trust the content of web pages and give away their confidential information. Several common approaches are described below.

(i) Spoofed anchor: In HTML, a href attribute points to the next page to be visited, if a user clicks on a hyperlink. However, one can specify an arbitrary domain name in the text portion of anchor tags (i.e., text written between the tags ⟨a⟩ and ⟨/a⟩) [21,22]. For example, the code ⟨a href= "http://www.example.com"⟩ www.goodwebsite.org ⟨/a⟩ shows www.goodwebsite.org in a browser. If a user clicks on the link, the next page is fetched from www.example.com. Moreover, browsers ignore anything written before the @ symbol in a URL [4,22]. For example, the URL pointed by the site www.bloomberg.com/www.badguy.com is actually www.badguy.com. Attackers use hexadecimal and unicode representation of URLs to hide plain text URLs. It is also common to substitute one or more letters of a legitimate URL, which might not be noticed by victims. For example, www.facebook.com is a spoofed anchor of www.facebook.com.

(ii) Domain name inconsistency: A phishing page shows a deviation between its current domain and the actual domain with respect to a claimed identity [22]. For example, one might specify in the header of an HTML page “Welcome to Ebay”. However, the domain from where the page is downloaded is not related to www.ebay.com.

(iii) Fake SSL Certificate: Many organizations use secured HTTP connections to transfer data between web applications and browsers. To cope with this, phishers might develop websites that support HTTPS communications (i.e., their pointed URLs start with https:// instead of http://) [21]. However, certificates used by phishers are self-created and not issued by trusted certificate providers such as Verisign.com. Browsers generate alarms, if there is any inconsistency in different fields of a certificate such as issuer name and expiry date. Unfortunately, end users often do not understand the meaning of different fields and rarely examine these certificates.

(iv) Sub-domain usage: Phishing URLs contain some parts of legitimate website URLs. For example, the legitimate website of NatWest bank is www.natwest.com. A phisher chooses a URL name which contains part of the legitimate URL such as www.natwest.com.mi.htmlout.com.

(v) Image: Images are used to substitute portion of legitimate web pages that might contain menus and logos. Sometimes, images are downloaded from a legitimate website in a phishing page.

2.2. Leveraging DOM-based feature

Some phishing attacks leverage the rich features (e.g., button click events) offered by Document Object Model (DOM) and
supported by client side scripting languages such as JavaScript [23].
We provide some examples below.
(i) Customizing status bar: Phishers apply JavaScript code to
generate fake addresses in a status bar. Let us consider the
code snippet shown in Fig. 1. When a user moves the mouse
on the hyperlink, he finds that a status bar is showing the
site www.goodwebsite.com. However, if he clicks on the link, a
(ii) XSS-based form: Cross site scripting (XSS) is a common
vulnerability in web-based programs, where user supplied inputs
are not filtered properly. As a result, arbitrary HTML or JavaScript
code can be injected through user inputs that alter the expected
DOM of a web page. XSS vulnerabilities can be leveraged to perform
phishing attacks by injecting HTML forms that can collect inputs
and send them to attacker controlled repositories [24]. To avoid a
user’s suspicion, it is common to load a form in an iframe whose
source might be loaded from an attacker supplied script code.
An HTML form can also be generated by injecting JavaScript code
through user inputs. Moreover, the injected code can be encoded to
be barely noticeable, and they are rarely detected by anti-phishing
tools.

There are some auxiliary symptoms of phishing web pages that
include the presence of a high percentage of objects downloaded
from different URL, inappropriate logos of companies, and a large
number of dots in the URL [21,4,22]. In most cases, a phishing site
has no Domain Name Service (DNS) record. Current techniques
have addressed the detection of spoofing website content and
a subset of DOM-based spoofing technique (e.g., customizing
toolbar). However, current tools cannot detect attacks that use
XSS-based forms (i.e., XSS-based form).

3. Related work

3.1. Trustworthiness testing

Trustworthiness testing has been pioneered by the work of self-
verifying data where explicit properties (e.g., source of input data,
content layout, and format) of data structures are checked during
testing as opposed to the verification of program outputs [20].
Several works have tested non-testable programs through testing of
trustworthiness.

Hook and Kelly [18] apply a testing technique to verify
program trustworthiness when exact outputs are unknown. They
apply mutation-based analysis that generates mutants (source
code modification based on a set of rules) of a given program
followed by a set of test cases (random inputs). While mutants
are killed by randomly generated test cases, the set of generated
outputs of mutant programs are compared with the set of
outputs of the original program. The idea is that if the maximum
deviation between the outputs generated by a mutant and an
original program is acceptable, then the implemented program
is considered as trustworthy. We believe that testing of suspected
phishing websites shares some common similarities of their work
as we do not know the response pages before testing.

Leung et al. [19] develop a set of test criteria for trustworthiness
testing of embedded programs (mobile handsets). Their developed
criteria are based on the intrinsic properties of production envi-
ronment (e.g., network wave should fluctuate, the signal strength
should be within a specified limits), operation environment (e.g.,
a mobile station sometimes receives signal from a faraway base
station as opposed to the nearest station), and benchmark results
(e.g., output of two mobile handsets should be close or similar).
In contrast, our approach tests the trustworthiness of suspected
websites based on known phishing and legitimate website behav-
iors. Our proposed heuristics are developed based on a FSM model
representing behaviors related to form submissions with random
inputs.

3.2. Phishing attack detection

We first discuss some works that detect phishing web pages
or websites. Table 1 provides a brief summary of these works
in comparison to our work with respect to five features. These
include detection of attacks by supplying random inputs, testing
of multiple pages, examination of SSL certificates, language
independence, and detection of XSS-based attacks. We now briefly
discuss these works in the following paragraphs.

Liu et al. [25] detect phishing web pages for a given legitimate
website URL. They develop an intermediate representation (in
terms of blocks and features) of a legitimate page. Then, suspicious
URLs are generated based on heuristic rules (e.g., by replacing
‘o’ with ‘0’) followed by downloading web pages from suspicious
URLs. These pages are converted to intermediate representation
and compared with the actual page to detect phishing based on
visual similarity assessment. However, the heuristics might not
generate all possible phishing URLs.

Yue and Wang [26] develop the BogusBiter tool that intercepts
credential of users, generates a large number of fake credentials,
and places the credential among the fake credentials to nullify the
attack. A similar approach has been applied by Joshi et al. [27]
(the PhishGuard tool) who intercept user submitted credentials.
However, to hide an actual supplied credential, they send another
set of fake credentials at the end. Krida and Kruegel [28] save a
mapping between supplied credentials and corresponding trusted
domains during the learning phase. In a detection phase, a
submitted credential is matched with the saved credentials, and
the current domain name is compared with the saved domain
names. If there is no match, a website is suspected as phishing.

Rosiello et al. [29] improve the technique of Krida and Kruegel
[28] by saving not only the association between user supplied
credentials and website domains, but also the DOMs of trusted web
pages.

Pan and Ding [21] detect phishing web pages by identifying the
anomalies in declared identities (e.g., keyword, copyright related
text present in HTML) and observing how anomalies manifest
due to DOM objects and HTTP transactions (e.g., server form
handler). Dong et al. [3] develop user profiles by creating or
updating binding relationships that relate user supplied personal
information and trusted websites. When a user is about to submit
his credentials, a detection engine generates a warning, if there is
no match between the current and the previously learned binding
relationship.

Zhang et al. [4] develop the CANTINA tool which leverages
the TF-IDF (term frequency and inverse document frequency)
algorithm to identify most weighted texts (or words) and generates
lexical signatures from the top five most important words. These
signatures are searched through a trusted search engine such as
Google. The resultant domain names are compared with the current
domain. If there is no match, then the current page is identified as
phishing.

Xiang et al. [30] apply information extraction and retrieval
techniques to detect phishing pages. The DOM of a downloaded
page is examined to recognize its identity through different
attributes (e.g., page title, copyright) and identify the actual
domain name based on the identity. Next, they search the current
domain of a suspected page and compare the result with the
Table 1
A summary of related works for detecting phishing web pages.

<table>
<thead>
<tr>
<th>Work</th>
<th>Brief description</th>
<th>Input supply</th>
<th>Multiple pages</th>
<th>Certificate</th>
<th>Language independence</th>
<th>XSS-based attack</th>
</tr>
</thead>
<tbody>
<tr>
<td>Liu et al. [25]</td>
<td>Detect fake web pages based on visual similarities.</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Yue and Wang [26]</td>
<td>Supply bogus credentials when a web page is detected as phishing to avoid information leakage.</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Joshi et al. [27]</td>
<td>Submit fake credentials before and after actual user credentials.</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Krida and Kruegel [28]</td>
<td>Compare the domain of a visited URL with trusted domain names where a user previously submitted personal information.</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Rosiello et al. [29]</td>
<td>Compare DOM-tree similarities between saved web pages and a new web page.</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Pan and Ding [21]</td>
<td>Identify anomalies in web page identities by examining HTML contents, DOM objects, and HTTP transactions.</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Dong et al. [3]</td>
<td>Identify phishing web pages by computing phishing scores.</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Zhang et al. [4]</td>
<td>Analyze web page contents and broken hyperlinks to extract identity and verify by a trusted search engine.</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Xiang and Hong [30]</td>
<td>Extract the identity of a page, obtain the true domain, and compare the result with the current domain using a search engine.</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Chou et al. [22]</td>
<td>Compute the spoof index of a given site based on the characteristics used in previous phishing attacks.</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Wenyin et al. [31]</td>
<td>Detect target phishing pages by constructing a semantic link network and reasoning.</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Ma et al. [32]</td>
<td>Classify phishing URLs from legitimate URLs based on lexical and host-based features.</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Our work</td>
<td>Detect phishing attacks based on application behavior model.</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

previous search output. If there is no common result in the two sets, the downloaded page is suspected as phishing.

Chou et al. [22] develop the SpoofGuard tool that detects phishing web pages based on heuristics and computes spoof scores based on matched heuristics. The heuristics include the features of stateless evaluation (e.g., a page containing Amazon logo asking user id and password), stateful evaluation (e.g., whether a domain already visited before), and input data (e.g., data has been sent to a site before). If the score exceeds a threshold, a page is suspected to be phishing.

Wenyin et al. [31] identify a phishing page target which helps in registering legitimate pages in advance so that anti-phishing tool [25] can use them when comparing with a suspected page. They develop a semantic link network (SLN) for a set of pages, where a node can be considered as a page, and a link connects two nodes. They compute implicit relationship among the pages by reasoning. If a suspected page targets other associated pages in all steps of the reasoning, it is considered as phishing. Ma et al. [32] classify phishing URLs from legitimate URLs based on a set of features that include both lexical (i.e., text properties of an URL) and host-based (e.g., WHOIS, IP address) features.

3.3. Web application modeling and testing

Our work is also motivated by several approaches that test web-based programs based on formally specified behaviors. We discuss these approaches in brief now.

Fantinato and Jino [33] propose a FSM-based model that captures data flow properties (e.g., data definition and usage) of a web-based program to identify data flow anomalies. Song et al. [34] develop an extended FSM to model program navigations. The model is intended to reveal faults related to forward and backward button-based page visits. Andrews et al. [35] propose testing of web applications at system level (i.e., black box) using an FSM model. They primarily address the state explosion problem for modeling web programs while generating system level test cases. Han et al. [36] propose an adaptive navigation model of web-based programs using Unified Modeling Language (UML) state charts to model hyperlink addresses available to users at different access modes (e.g., login and not logged in). Leung et al. [37] apply state charts to model the navigation of web pages where links can be generated by HTML anchor tags or scripting languages. Ricca and Tonella [38] perform static verification and dynamic validation.
of web-based programs based on a generic UML meta model. A program is considered as an instance of their proposed model. The static verification reveals faults related to hyperlinks (e.g., unreachable page). The dynamic validation executes a program for revealing faults related to data flow coverage (e.g., testing all navigation paths from every definition of a variable to every use of it).

A brief summary of these works along with our work is shown in Table 2. In contrast to all these works, our objective is to verify phishing attacks by considering program behaviors with respect to form submissions with random inputs and identifying matching of known up-to-date heuristic combination. The heuristics are based on states and specific features of response pages. Thus, we include only a subset of web-based program behaviors.

4. Behavior model and testing

In this section, we first introduce the proposed behavior model for web-based programs (or websites) in Section 4.1. We then define some heuristics criteria to verify phishing and legitimate sites in Sections 4.2–4.4. Section 4.5 shows a relationship between a number of phishing attack types and heuristics.

4.1. Program behavior model

We apply Finite State Machine (FSM) notion to describe a program’s behavior. We develop it based on known symptoms of phishing and legitimate websites with respect to form submissions with random inputs. We first describe the assumptions of the FSM. A phishing website often contains pages which may have forms. However, if a page does not contain any form, it might have HTML elements such as buttons, which enable one to reach another page containing a form. Submitting a form requires not only filling data, but also clicking a button to post the form. Any information provided through a form as part of a phishing attack is usually accepted.

The FSM is denoted by \( \langle S, S_0, I, \delta, F \rangle \), where \( S \) is a finite set of states, \( S_0 \) is the initial state, \( I \) is a finite set of inputs, \( \delta \) is the state transition function, and \( F \) is a set of final states. Fig. 2 shows the state transition diagram of the FSM that contains the states from \( S_0 \) to \( S_{10} \). \( S_0 \) is the initial state. If a downloaded page from a given URL contains no form, then the next state is considered as \( S_1 \). However, if the downloaded page contains a form, then the next state is considered as \( S_2 \). \( F \) is the final state which belongs to \( \{S_3, S_4, S_5, S_6, S_7, S_8, S_9, S_{10}\} \). Here, a state implies a web page rendered by a browser. To avoid the state explosion problem, we consider the content of a web page as a single state.

We denote inputs of the FSM as a pair of interesting requests (denoted as \( x_0 \rightarrow x_1 \)) and corresponding responses (denoted as \( y_1 \rightarrow y_0 \)), which are discussed in detail in the following paragraphs. A website is phishing or legitimate, if it can reach from an initial state to one of the final states. Some of the final states are legitimate \( \{S_4, S_5, S_6\} \), whereas some of the final states are phishing \( \{S_3, S_7, S_8, S_{10}\} \). Infeasible states are removed from the figure.

A state transition occurs for a given request and the corresponding response. A transition is labeled as (request, response) pair in the figure. For example, \( (x_1, y_1) \) implies that given the request \( x_1 \), the response is \( y_1 \). We summarize interesting requests and responses in Table 3. There are three kinds of requests: a page download from an initial URL (symbol \( i \)), a button click (symbol \( c \)) which might generate a new page (denoted as \( x_1 \)), and form submission that comprises of filling data form (symbol \( d \)) and clicking a button (denoted as \( x_2 \)).

We describe eight interesting response features in the following paragraphs.

(i) \( f \): It indicates that an input form is present in a downloaded page.

(ii) \( e \): An error message is present in a page, which is due to a form submission.

(iii) \( s \): A downloaded page contains different SSL signature attributes (e.g., issuer and expiry date) compared to the previous page.

(iv) \( r \): It indicates that a page is downloaded due to a redirection from the previous page. Moreover, in the new page there is no hyperlink to visit to the previous suspected domain.

(v) \( p \): This feature indicates that information supplied to a form is submitted to a different domain (or third party) with respect to the current suspected domain.

(vi) \( E_{\text{max}} \): A form submission might result in an error message provided that the supplied inputs are random. Legitimate websites allow a limited number of attempts for submitting wrong credentials. Thus, we choose a limit on error message count (three) due to the form submission.

(vii) \( F_{\text{max}} \): The maximum number of form to be submitted for performing a functionality is finite for a legitimate website. In contrast, a phishing website might be designed to repeat the submission of the same form(s) by a victim. Thus, we choose a maximum limit on the number of form submission for a suspected website which can be set as 6.

(viii) \( t \): A phishing website usually does not replay any supplied input in a response page (e.g., user id or email). In contrast, a legitimate website often shows the supplied inputs (e.g., a greeting message with provided email or user id). This feature allows us to differentiate between a legitimate and a phishing website.

For simplicity, we group \( s, r, \) and \( p \) as one observation, which is denoted as \( \langle s, r, p \rangle \). This also means that a change of any of these features is treated as one observation to identify a website to be phishing. \( \langle s, r, p \rangle \) indicates that there is no occurrence of changes of the corresponding features in a response. Here, the symbol \( ! \) represents that a feature is not present in a page. For example, \( !f \) indicates that there is no form present. Thus, we might observe a possible of 64 possible responses with respect to the features. However, in Fig. 2, we only use nine interesting responses (denoted as \( y_1 \), \( y_2 \)). The rest other combinations (e.g., \( f, e, \), \( s, r, p \), \( E_{\text{max}} \), \( F_{\text{max}} \), \( t \) ) are either infeasible or not related to attack cases, and we do not include them in the FSM.

The model provides us the flexibility to detect phishing websites that might steal information through an uncertain number of pages containing forms and employ various types of form generation techniques (e.g., non XSS-based, XSS-based). A phishing website might follow only a subset of the diagram. Moreover, the model differentiates a phishing and legitimate website. We apply an offline analysis approach to navigate and download all the accessible pages by submitting random inputs and observe interesting responses. To facilitate testing, we check several heuristics based on the model to identify whether a website is phishing or legitimate. We first develop some heuristics based on the states of FSM which are discussed next.

<table>
<thead>
<tr>
<th>Work</th>
<th>Test criteria</th>
<th>Suitable for phishing?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fantinato and Jino [33]</td>
<td>Data flow coverage.</td>
<td>No</td>
</tr>
<tr>
<td>Song et al. [34]</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Andrews et al. [35]</td>
<td>All transition pairs.</td>
<td>No</td>
</tr>
<tr>
<td>Han and Hofmeister [36]</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Leung et al. [37]</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Ricca and Tonella [38]</td>
<td>Page, hyperlink, and data flow.</td>
<td>No</td>
</tr>
<tr>
<td>Our work</td>
<td>Heuristic combination</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Table 2: Comparison of web application testing works.
Table 3
List of requests and responses.

<table>
<thead>
<tr>
<th>Name</th>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_0$</td>
<td>$i$</td>
<td>Web page downloaded from an initial URL.</td>
</tr>
<tr>
<td>$x_1$</td>
<td>$c$</td>
<td>Button click.</td>
</tr>
<tr>
<td>$x_2$</td>
<td>$d, c$</td>
<td>Form filling and button click.</td>
</tr>
<tr>
<td>$y_1$</td>
<td>$f, e, ! (s, r, p)$, $E_{\text{max}}$, $F_{\text{max}}$, $t$</td>
<td>No form, no error message, no occurrence of certificate attribute change, redirection, or posting information to third party, # of error message is not maximum, # of form submission is not maximum, input is not present in the response.</td>
</tr>
<tr>
<td>$y_2$</td>
<td>$f, e, ! (s, r, p)$, $E_{\text{max}}$, $F_{\text{max}}$, $t$</td>
<td>Form present, no error message, no occurrence of certificate attribute change, redirection, or posting information to third party, # of error message is not maximum, # of form submission is not maximum, input is not present in the response.</td>
</tr>
<tr>
<td>$y_3$</td>
<td>$f, e, ! (s, r, p)$, $E_{\text{max}}$, $F_{\text{max}}$, $t$</td>
<td>Form present, error message, no occurrence of certificate attribute change, redirection, or posting information to third party, # of error message is maximum, # of form submission is not maximum, input is not present in the response.</td>
</tr>
<tr>
<td>$y_4$</td>
<td>$f, e, ! (s, r, p)$, $E_{\text{max}}$, $F_{\text{max}}$, $t$</td>
<td>No form, no error message, occurrence of certificate attribute change, redirection, or posting information to third party, # of error message is not maximum, # of form submission is not maximum, input is not present in the response.</td>
</tr>
<tr>
<td>$y_5$</td>
<td>$f, e, ! (s, r, p)$, $E_{\text{max}}$, $F_{\text{max}}$, $t$</td>
<td>No form, no error message, no occurrence of certificate attribute change, redirection, or posting information to third party, # of error message is not maximum, # of form submission is not maximum, input is not present in the response.</td>
</tr>
<tr>
<td>$y_6$</td>
<td>$f, e, ! (s, r, p)$, $E_{\text{max}}$, $F_{\text{max}}$, $t$</td>
<td>Form present, no error message, no occurrence of certificate attribute change, redirection, or posting information to third party, # of error message is not maximum, # of form submission is maximum, input is not present in the response.</td>
</tr>
<tr>
<td>$y_7$</td>
<td>$f, e, ! (s, r, p)$, $E_{\text{max}}$, $F_{\text{max}}$, $t$</td>
<td>Form present, no error message, no occurrence of certificate attribute change, redirection, or posting information to third party, # of error message is not maximum, # of form submission is not maximum, input is present in the response.</td>
</tr>
<tr>
<td>$y_8$</td>
<td>$f, e, ! (s, r, p)$, $E_{\text{max}}$, $F_{\text{max}}$, $t$</td>
<td>Form present, no error message, occurrence of certificate attribute change, redirection, or posting information to third party, # of error message is not maximum, # of form submission is not maximum, input is not present in the response.</td>
</tr>
<tr>
<td>$y_9$</td>
<td>$f, e, ! (s, r, p)$, $E_{\text{max}}$, $F_{\text{max}}$, $t$</td>
<td>Form present, error message, no occurrence of certificate attribute change, redirection, or posting information to third party, # of error message is not maximum, # of form submission is not maximum, input is not present in the response.</td>
</tr>
</tbody>
</table>

Fig. 2. FSM model representing behaviors of phishing and legitimate websites.

4.2. State-based heuristics and their limitations

We define three state-based heuristics based on the FSM model. They are no loop, single loop, and multiple loops. In the remaining of this sub-section, we first introduce them followed by discussing their limitation to distinguish between phishing and legitimate websites.

No loop (H1). If a website traverses more than one state, it indicates that the website is phishing or legitimate. For example, if a website reaches the state sequence $\langle S_0, S_1, S_2, S_3 \rangle$, it indicates a phishing attack that contains two pages. The second page has the form, whereas the first page contains no form. The corresponding test case is $\{ (x_0, y_1), (x_1, y_2), (x_2, y_2) \}$. In contrast, a website having the state sequence $\langle S_0, S_1, S_2 \rangle$ indicates a legitimate website.

Single loop (H2). If requests and the corresponding responses result in a website to remain the same state more than once, a loop is formed. A test case having a loop with respect to a state can represent either a phishing or a legitimate website. For example, a website might traverse the state sequence $S_0, S_1, S_2, S_3, S_3$, and $S_3$. Thus, it forms a loop around $S_3$. The corresponding test case is $\{ (x_0, y_1), (x_1, y_2), (x_2, y_2), (x_3, y_3) \}$. Note that meeting the single loop heuristic might not always indicate that a site is phishing. For example, a loop around state $S_5$ (e.g., $S_0, S_1, S_3, S_5$) indicates that the website is legitimate. In this case, the maximum number of error message heuristic (described later in this section) also needs to be satisfied. Note that transition from $S_5$ to $S_5$ relates to a form submission response page that contains an error message related to the previous form submission, $S_5$ is an intermediate state that intends to capture legitimate websites which do not accept random inputs.

Multiple loops (H3). If requests and the corresponding responses result in the formation of more than one loop, then multiple loops heuristic is satisfied. The heuristic might reveal advanced attacks that might force a victim to go through several pages where some pages may not contain forms. For example, the attack having loops at states $S_1$ and $S_2$. The related test case is $\{ (x_0, y_1), (x_1, y_1), (x_1, y_2), (x_2, y_2), (x_2, y_2) \}$. We now show that solely depending on state-based heuristics for testing suspected phishing websites with random inputs might result in false positive (i.e., a legitimate website is identified as phishing) or negative (i.e., a phishing website is identified...
Table 4
Functionalities and example test scenarios.

<table>
<thead>
<tr>
<th>Functionality</th>
<th>Example</th>
<th>Scenario</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Registration</td>
<td>1</td>
<td>ph -&gt; ph (no form)</td>
<td>Registration data is provided to a phishing website, the response page has no form and resides in the same website.</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>ph -&gt; ph (form)</td>
<td>Registration data is provided to a phishing website, the response page has a form and resides in the same website.</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>ph -&gt; re (no form)</td>
<td>Registration data is provided to a phishing website and the response page has no form and resides in a legitimate website.</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>ph -&gt; re (form)</td>
<td>Registration data is provided to a phishing website and the response page has a form and resides in a legitimate website.</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>re -&gt; re (no form)</td>
<td>Registration data is provided to a legitimate website, the final response page contains no form and resides in the same website.</td>
</tr>
<tr>
<td>Login</td>
<td>6</td>
<td>ph -&gt; ph (no form)</td>
<td>Login data is provided to a phishing website, the response page has no form and resides in the same website.</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>ph -&gt; ph (form)</td>
<td>Login data is provided to a phishing website, the response page has a form and resides in the same website.</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>ph -&gt; re (no form)</td>
<td>Login data is provided to a phishing website, and the response page has no form and resides in a legitimate website.</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>ph -&gt; re (form)</td>
<td>Login data is provided to a phishing website, and the response page has a form and resides in a legitimate website.</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>re -&gt; re (form)</td>
<td>Login data is provided to a legitimate website, the final response has a form and resides in the same website.</td>
</tr>
</tbody>
</table>

Table 5
Test scenarios and state-based heuristics.

<table>
<thead>
<tr>
<th>Functionality</th>
<th>Example</th>
<th>Scenario</th>
<th>State sequence</th>
<th>State-based heuristic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Registration</td>
<td>1</td>
<td>ph -&gt; ph (no form)</td>
<td>$S_0, S_2, S_{10}$</td>
<td>H1, H2</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>ph -&gt; ph (form)</td>
<td>$S_0, S_2, S_7$</td>
<td>H1, H2</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>ph -&gt; re (no form)</td>
<td>$S_0, S_2, S_8$</td>
<td>H1, H2</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>ph -&gt; re (form)</td>
<td>$S_0, S_2, S_7$</td>
<td>H1, H2</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>re -&gt; re (no form)</td>
<td>$S_0, S_2, S_4$</td>
<td>H1, H2</td>
</tr>
<tr>
<td>Login</td>
<td>6</td>
<td>ph -&gt; ph (no form)</td>
<td>$S_0, S_2, S_{10}$</td>
<td>H1</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>ph -&gt; ph (form)</td>
<td>$S_0, S_2, S_7$</td>
<td>H1, H2</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>ph -&gt; re (no form)</td>
<td>$S_0, S_2, S_8$</td>
<td>H1</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>ph -&gt; re (form)</td>
<td>$S_0, S_2, S_7$</td>
<td>H1, H2</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>re -&gt; re (form)</td>
<td>$S_0, S_2, S_5$</td>
<td>H1, H2</td>
</tr>
</tbody>
</table>

as legitimate) conclusions. To demonstrate, we consider two most common functionalities namely login and registration that are present in legitimate websites. These functionalities are mimicked by most of the phishing websites. We assume that these functionalities require one or more form submissions, respectively. Note that our discussion is applicable for other common functionalities such as update and delete.

Let us assume that a registration requires visiting two web pages, whereas the login requires visiting one web page only. We also assume that a website generates a response page that may or may not contain a form. Based on these, we identify ten examples of performing these functionalities and observation of response pages. We denote these examples as test scenarios.

The third and fourth columns of Table 4 show test scenarios and corresponding descriptions for these functionalities. For example, in the first row, a phishing website (denoted as ph) gathers information for registration purpose and at the end of information collection a victim remains in the same phishing website. Moreover, the response page has no form. We denote this testing scenario as ph -> ph (no form). However, in the second row (ph -> ph (form)), a victim observes a response page that has a form and is downloaded from the same phishing website after a successful attack. Similarly, in the third row, we use the notation ph -> re (form) to represent an attack scenario where a phishing website grabs information and generates a response page that contains a form. Moreover, the page is downloaded from a legitimate (denoted as re) website.

Table 5 shows example state sequences (based on Fig. 2) for these functionalities and corresponding state-based heuristics that are satisfied. For example, in the third row, the registration information collection in a phishing website followed by redirecting a victim to a legitimate website containing no form results in the state sequence $S_0, S_2, S_8$ (based on Fig. 2). Here, $S_2$ creates a loop and the resulting state sequence satisfies single loop. Thus, the state sequence satisfies both H1 and H2 criteria. Let us assume that satisfying H1 and H2 implies that a website is phishing. Unfortunately, this would result in false positive conclusions for legitimate websites (Examples 5 and 10). On the other hand, if we decide that satisfying H1 and H2 indicates a website is legitimate, testing would suffer from false negative conclusions. In that case, many phishing websites cannot be detected (Example 1). Thus, it is obvious that state-based heuristics are not enough to precisely testing phishing and legitimate websites. This motivates us to develop additional heuristics that are based on form submission responses and form features to reduce false positives and negatives. We discuss these heuristics and their application to develop a number of heuristic combinations in Sections 4.3 and 4.4, respectively.

4.3. Response and form-based heuristics

We define heuristics that are related to the response of form submissions and feature of input forms for both login and registration functionalities. They are discussed below.

4.3.1. Response-based heuristics

We define three heuristics that relate the response of form submissions. These include maximum form submission, maximum error message, and the presence of supplied input.

Maximum form submission (H4). A website can provide login functionalities that can accept either random or legitimate inputs. In any case, a legitimate website often provides a limited number of form submissions. In contrast, a phishing website might be designed to accept unlimited number of form submissions. Thus, we need a heuristic that checks the maximum number of form submission during testing (H4). If a website exceeds the maximum number of form submissions with random inputs during testing, it is most likely to be a phishing website as opposed to a legitimate site. We consider the maximum number of form submission value as six in this paper.
Maximum error message (H5). Phishing websites rarely verify the provided information for login functionality. On the other hand, a legitimate website rejects random inputs and generates an error message in response pages. Moreover, a legitimate website might block a user for providing further random inputs. This observation leads to the development of a heuristic based on the maximum number of error message observed during form submission. If providing random inputs result in the maximum number of error messages, it indicates that a website is likely to be a legitimate as opposed to a phishing. We consider the maximum number of error message as three.

Presence of supplied input (H6). This criterion is satisfied, if the response of a form submission contains part of the random inputs that have been provided. A legitimate website often greets a user after a successful login or registration with the supplied name, user id, or email. On the other hand, a phishing website does not generate a response page that contains the supplied inputs. This observation motivates us to define a heuristic based on the presence of supplied inputs in a response page.

### 4.3.2. Form-based heuristics

We define two form feature-based (also denoted as form-based) heuristics which are discussed below.

No form (H7). This heuristic criterion checks whether a form submission results in a page that has no input form (or no hyperlink to proceed further). In other words, there is no way to proceed further from the response page. Note that if a response page is non-existent, we consider it as an example of no form (i.e., the heuristic is satisfied). This feature is useful to detect phishing websites that result in response pages with no further forms to perform functionalities.

Common form (H8). This criterion is satisfied, if a current form being submitted with random inputs matches with any of the forms that have been submitted before. Two forms are considered as common form if their field name and types are similar.

### 4.4. Heuristic combinations

Table 6 shows the mapping of test scenarios (discussed in Table 4), state, form submission response, and form-based heuristics in columns three to seven, respectively. We observe that using state, response, and form-based heuristics allow us to distinguish all the phishing websites from legitimate websites. Moreover, specific combination of heuristics can be identified from the table. For example, a legitimate website can be detected by checking two different combinations of heuristics: (i) H1, H2, H6, and H7, or (ii) H1, H2, H5, and H8. On the other hand, if a website behavior satisfies H1, H2, and H7 (the first row), then it can be concluded as phishing.

Table 7 shows the summary of the combination of heuristics that need to be satisfied to detect a website as phishing or legitimate. Note that we ignore the presence of a form in the response page after an attack. The first row (ph -> ph) shows that three combinations of heuristics can be tested for detecting phishing websites where after an attack a victim resides in phishing websites. Similarly, for ph -> re and re -> re, we identify four and two combinations of heuristics, respectively. The third column of Table 7 shows the derived generalized heuristic combinations. Here, we have eliminated the duplicate combination of heuristics that are present in test scenarios of ph -> ph and ph -> re (e.g., H1, H2, and H6).

Since we have not considered H3 in testing scenarios, we can generalize state-based heuristics by the notation (H1, H2, H3). This implies that a website might satisfy one or more state-based heuristics. However, the form submission response-based heuristics are specific towards phishing (H4) and legitimate websites (H5 and H6). Moreover, form feature-based heuristics are satisfied in combination of state and specific response-based heuristics. These lead to the development of three (denoted as C1–C3) and four general heuristic combinations (denoted as C4–C7) to test phishing and legitimate websites, respectively. The other combination of heuristics (e.g., {H1, H2, H3}, H5, H6 for testing legitimate websites) are not meaningful or invalid based on the assumptions.

A tester can track the heuristic combinations while testing suspected websites and decide if a website is phishing or legitimate. Another advantage of having heuristic combination-based detection is that phishing attacks mounted on top of XSS-based forms satisfy the identified heuristic combination. Nevertheless, if a website does not satisfy any of the combination of heuristic, then a manual checking is required to decide further. This would be possible if a website violates any of the assumption behind the FSM-based behavior model.

### 4.5. Relationship between attacks and heuristic combination

In this section, we describe how FSM-based heuristics can be applied to discover phishing attacks. A summary of some example attack types and corresponding heuristics (state, response, and
form) is shown in Table 8. We divide phishing attack types into eight categories (a1–a8).

(i) Redirecting to legitimate website (a1): Most phishing attacks redirect users to legitimate websites after collecting personal information. More interestingly, this happens in the same browser window where a victim opens the phishing site. The state (H1, H2, or H3) and form-based (H7 or H8) heuristics can detect the attack.

(ii) SSL-based attack (a2): Most phishing sites do not employ any SSL certificate-based communication. However, phishing sites might redirect users to legitimate sites having SSL certificates (e.g., Ebay and Paypal). Thus, state (H1, H2, or H3) and form-based (H7 or H8) heuristics can detect the attack. Note that for a1 attack type, we assume that both phishing and legitimate sites have no SSL certificate-based encryption mechanism employed.

(iii) Information sent to third party (a3): A phishing website might collect information and send it to a different domain to avoid user suspicion. In this case, a phisher’s data repository remains in a different location from the site where web pages are located. Often an email address is used instead of a domain. Thus, state (H1, H2, or H3) and form-based (H7 or H8) heuristics allow discovering the attack.

(iv) No form in the beginning (a4): To circumvent heuristics for detecting phishing web pages, a phisher might develop a website where the first page contains no form. However, it is common to have a button and clicking the button generates a new page that may contain an HTML form. In other words, phishers might try to position the form page further than usual to avoid anti-phishing tools. This attack type can be discovered by state (H1, H2, or H3) and form-based (H7 or H8) heuristic criteria.

(v) Variable error message (a5): As phishing websites cannot validate submitted information, a typical mechanism employed by a phisher is to generate an error message regardless of inputs to confuse victims so that victims become more careful in providing right information. If the form is submitted with another random input, then the previous error message disappears. In contrast, a legitimate site might show similar error message with random input submission and might log the IP address of a user to block submitting further information. Thus, any attack related to variable error message can be detected with state (H1, H2, or H3) and form-based (H7 or H8) criteria.

(vi) Similar form pattern (a6): An attack might result in generation of one or more inputs forms cyclically to users without any error message while submitting those forms. Thus, a user might experience in feeding information to the same form repeatedly. The attack can be detected by state (H1, H2, or H3), response (H4), and form-based (H8) criteria.

(vii) Isolated page (a7): An attack might grab user information and generate a page which has no more hyperlink to visit another page. It can be detected by state (H1, H2, or H3) and form-based (H7) criteria.

(viii) Non-existent page (a8): A form submission might result in an error message in the browser due to a non-existent page. This is common when a phishing site fails to redirect a victim to a legitimate site. The attack can be detected by state (H1, H2, or H3) and form-based (H7) criteria.

5. Implementation and evaluation

5.1. Implementation

Given a URL of a website, we need to decide whether it is phishing or legitimate. The testing could be performed by leveraging a crawler which can download static pages. The HTML code is scanned for the presence of form or clickable buttons. However, crawlers have limitations when comes to form submission with random inputs as well as discovering clickable elements that might generate new pages. Thus, we have implemented a Java class to download pages from URLs, analyze HTML code to identify form fields, construct URLs corresponding to form submissions, and fetch subsequent pages. While doing so, the response features are observed by analyzing HTTP response (e.g., status code) and contents of new pages (e.g., analyzing META tags to identify the URL of the redirected website). To analyze HTML pages, we employ the Jericho HTML parser [39] that provides necessary APIs for Java. The submitted inputs are drawn from a test suite repository which contains random inputs of different types such as name, address, and date of birth. We also gather inputs corresponding to bank information for the testing purpose [40] that include random credit card numbers, expiry dates of credit cards, pin numbers, bank account numbers etc.

5.2. Experimental evaluation

We perform experiments in this section to identify two issues: (1) How well the proposed approach can detect phishing websites (i.e., evaluation of false negative rate)? (2) Can the approach distinguish between phishing and legitimate websites (i.e., evaluation of false positive rate)?

5.2.1. Evaluation of false negative rate

We show the effectiveness of our approach by testing the reported URLs of PhishTank [41], which is free and widely used data source among anti-phishing researchers [3,4,26,42,23,14]. One important feature of the PhishTank data is that they are reported by volunteers. Thus, it might be possible that some reported sites are actually not phishing. To avoid any confusion, we only evaluate the sites that have been confirmed as phishing. Moreover, the reported URLs might not be accessible at a later time as these websites have been taken down by the administrators. Thus, we tested URLs that have been accessible at the time of the evaluation.

We chose 33 phishing URLs during the last week of July of 2010.2 Note that there have been actually more than 33 URLs that have been reported to be phishing during this time period. However, we are interested to examine unique phishing websites, which prohibit us to include a large portion of the reported

Table 8
Phishing attack types and heuristics.

<table>
<thead>
<tr>
<th>Name</th>
<th>Attack type</th>
<th>State</th>
<th>Response</th>
<th>Form</th>
</tr>
</thead>
<tbody>
<tr>
<td>a1</td>
<td>Redirecting to a legitimate website</td>
<td>H1, H2, H3</td>
<td>N/A</td>
<td>H7/H8</td>
</tr>
<tr>
<td>a2</td>
<td>SSL-based attack</td>
<td>H1, H2, H3</td>
<td>N/A</td>
<td>H7/H8</td>
</tr>
<tr>
<td>a3</td>
<td>Information sent to third party</td>
<td>H1, H2, H3</td>
<td>N/A</td>
<td>H7/H8</td>
</tr>
<tr>
<td>a4</td>
<td>No form in the beginning</td>
<td>H1, H2, H3</td>
<td>N/A</td>
<td>H7/H8</td>
</tr>
<tr>
<td>a5</td>
<td>Variable error message</td>
<td>H1, H2, H3</td>
<td>N/A</td>
<td>H7/H8</td>
</tr>
<tr>
<td>a6</td>
<td>Similar form pattern</td>
<td>H1, H2, H3</td>
<td>H4</td>
<td>H8</td>
</tr>
<tr>
<td>a7</td>
<td>Isolated page</td>
<td>H1, H2, H3</td>
<td>N/A</td>
<td>H7</td>
</tr>
<tr>
<td>a8</td>
<td>Non-existent page</td>
<td>H1, H2, H3</td>
<td>N/A</td>
<td>H7</td>
</tr>
</tbody>
</table>

2 The data set is accessible from the website http://research.cs.queensu.ca/~shahriar/phishing/repository-2010.rar.
URLs. Many reported URLs are actually just one unique program or website (e.g., a common set of web pages). However, they have been hosted in different domains and hence, different URLs have been reported in the PhishTank repository. We randomly consider one URL when multiple URLs represent the same website (program). Nevertheless, the chosen websites might not be complete as we sometimes find a website is inaccessible (i.e., taken down by a website administrator).

Our chosen URLs belong to 26 organizations whose names and business types are shown in the first and second columns of Table 9. These include renowned banks such as Bank of America, HSBC, and Abbey, as well as e-commerce-based organizations such as Ebay and Paypal. Several websites are implemented in the languages other than English such as French and German. The third and fourth columns of Table 9 show the number of websites mimicking login and registration functionalities, respectively. We note that most phishing websites collect registration (e.g., personal profile, banking profile) related information as opposed to login information.

Table 10 shows the evaluation results of suspected phishing websites. The second and third columns show the total number of phishing websites that are tested with random inputs and the number of websites that are detected as phishing, respectively. Our approach detects all the 33 suspected websites as phishing (i.e., zero false negative rates). The fourth and fifth columns show the number of phishing websites that result in response pages with no form and forms after collecting information, respectively. Note that these response pages are generated from the same phishing websites. We observe that most of the phishing websites that allow victims to stay in the same website generate response pages with no form after grabbing information. The sixth and seventh columns show the number of phishing sites that result in response pages from legitimate websites containing no form and forms, respectively. Most of the phishing websites that redirect victims to legitimate websites point to the web pages that contain forms.

Table 11 shows heuristic summary for all the phishing websites. Columns 2–9 show state (H1–H3), form submission response (H4–H6), and form-based (H7, H8) heuristics that are satisfied during testing. The last row of the table shows the average number of websites that satisfy each of the heuristics. We observe that most of the phishing websites satisfy no loop (H1) and single loop (H2) heuristics. Moreover, many phishing websites satisfy no form (H7) and common form (H8) heuristics. In contrast, very few phishing websites satisfy multiple loops (H3) and the maximum number of form submission (H4) heuristics. No phishing websites satisfy the maximum number of error message heuristic (H5) or generate response pages with supplied inputs (i.e., heuristic H6 is not satisfied).

The phishing websites that satisfy C1, C2, and C3 are shown in columns 10, 11, and 12 respectively. The last row of the table
Table 11
Summary of heuristics and their combinations.

<table>
<thead>
<tr>
<th>Name</th>
<th>H1</th>
<th>H2</th>
<th>H3</th>
<th>H4</th>
<th>H5</th>
<th>H6</th>
<th>H7</th>
<th>H8</th>
<th>C1</th>
<th>C2</th>
<th>C3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abbey Bank</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>ABN AMRO</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Alliance &amp; Leicester</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>ASB Bank New Zealand</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Bank of America</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>BattleNet</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>BBVA Net Office</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Chase</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Citi Bank</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Ebay</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Halifax Bank</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Hotmail</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>HSBC</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>ING Direct</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>IRS</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>KeyBank</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Lloyds TSB</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Mashreq</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>National City</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>NatWest Bank</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Northern Trust</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Paypal</td>
<td>6</td>
<td>4</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>4</td>
<td>2</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>RBS</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>USAA</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Wells Fargo</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>WT Direct</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Average (%)</td>
<td>100</td>
<td>75.8</td>
<td>3.0</td>
<td>12.0</td>
<td>0</td>
<td>0</td>
<td>48.5</td>
<td>51.5</td>
<td>48.5</td>
<td>39.4</td>
<td>12.1</td>
</tr>
</tbody>
</table>

shows the average number of websites that satisfy these heuristic combinations. We observe that most phishing websites satisfy C1 and C2. However, very few websites satisfy the C3 combination. Thus, we can conclude that C1, C2, and C3 can be applied to test phishing websites that mimic functionalities such as login and registration.

We map the phishing websites with attack types \(a_1 \ldots a_8\) that have been defined in Section 4. Table 12 shows attack type detection summary for the phishing websites. The last row shows the average number of phishing websites related to each attack type. From the last row, we observe that most phishing websites redirect victims to legitimate websites \(a_1\) and SSL-based attacks are common \(a_2\). Few phishing websites include the beginning pages with no form \(a_4\) and some websites end up in pages that contain no further links to proceed forward \(a_7\) or do not exist \(a_8\).

Moreover, many phishing websites are developed with only one form that is always provided to users even after providing random inputs \(a_6\).

5.2.2. Evaluation of false positive rate

We now evaluate whether our approach can detect legitimate websites based on the proposed heuristic combination or not. We select 19 white listed websites that are most popular and widely visited by users. They are chosen based on the ranking of white listed websites from Alexa [http://alexa.com]. We choose the websites that provide both login and registration functionalities. The selected websites include internet service provider, online shopping, email and file storage services, etc. We show the summary of evaluation for legitimate websites in Table 13 for login functionality.

The first and second columns of Table 13 show the name and business type of the legitimate websites. Columns 3–10 show the heuristics that are satisfied by the legitimate websites while testing login functionalities with random inputs. We note that all the websites satisfy H1 (no loop), H2 (single loop), H5 (maximum # of error message), and H8 (common form) simultaneously. This also implies that C7 is satisfied by all the legitimate websites. However, none of the legitimate websites with login functionality satisfy H3 (multiple loops), H4 (the maximum # of form submission), H6 (random inputs present in response pages), and H7 (no form).

Table 12 shows the evaluation results of legitimate websites for registration functionalities. Columns 2–9 show the heuristics \((H1\ldots H8)\) that are satisfied for each of the websites. The last four columns show the four heuristic combinations \((C4\ldots C7)\) that are satisfied. The last row shows the average number of websites that satisfy each of the heuristics and combination of heuristics. We observe that most of the legitimate websites satisfy H1, H2, H6, and H7 heuristics. Very few websites satisfy H3 and H8 heuristics.

Table 13: Evaluation summary of legitimate websites for the login functionality.

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>H1</th>
<th>H2</th>
<th>H3</th>
<th>H4</th>
<th>H5</th>
<th>H6</th>
<th>H7</th>
<th>H8</th>
<th>C4</th>
<th>C5</th>
<th>C6</th>
<th>C7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amazon</td>
<td>Online shopping</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>AOL</td>
<td>Internet</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Best Buy</td>
<td>Electronic vendor</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Craigslist</td>
<td>Classified ads</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Ebay</td>
<td>Online shopping</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Flickr</td>
<td>Picture gallery</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Future shop</td>
<td>Ecommerce</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Gmail</td>
<td>Email</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Hotmail</td>
<td>Email</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Kijiji</td>
<td>Classified ads</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>LinkedIn</td>
<td>Professional forum</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Megaupload</td>
<td>File storage</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>MySpace</td>
<td>Social network</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Paypal</td>
<td>Ecommerce</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>The Globe and Mail</td>
<td>News</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Twitter</td>
<td>Instant information</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Wordpress</td>
<td>Online blog generator</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Yahoo</td>
<td>Email</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Youtube</td>
<td>Video</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Average (%)</td>
<td>100</td>
<td>100</td>
<td>0</td>
<td>0</td>
<td>100</td>
<td>0</td>
<td>0</td>
<td>100</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>100</td>
<td></td>
</tr>
</tbody>
</table>

None of the websites satisfy H4 and H5 heuristics. Moreover, most of the legitimate websites satisfy C5. Several websites satisfy C4 and C6 combinations. However, none of the websites satisfy the C7 combination.

Based on the results, we can conclude that C4, C5, and C6 can be applied to test legitimate websites for registration related functionality. Moreover, C7 can be used to test a legitimate website for login related functionality.

5.3. Comparative evaluation

Since we intend to validate our testing approach for detecting phishing attacks that might be launched through XSS-based forms in trusted websites, we are motivated to perform a third evaluation. We also assess whether XSS-based phishing attacks are detected by other anti-phishing tools or not. All of the phishing websites that have been evaluated and discussed in Section 5.2 do not represent attacks that are due to the exploitation of XSS vulnerabilities. Thus, we choose related benchmark programs and develop an experimental setup before performing comparative evaluation. In the remaining of this section, we first discuss XSS vulnerable programs that are used in the evaluation followed by attack injection method. Then, we discuss the tools used for comparative assessment followed by the obtained results.

Vulnerable programs and attack injection. We choose three programs that have been reported to contain XSS vulnerabilities in Open Source Vulnerability Databases [43]. The vulnerable programs include guestbook, blog, and message board. Table 15 shows some characteristics of the programs that include vulnerability ID and file name. All the programs are implemented in PHP.

A simple form of phishing attack can be performed by injecting an HTML form (instead of writing non-malicious message to a blog or a guestbook) and asking a victim to provide information. We randomly choose a phishing (e.g., Hotmail) webpage and extract the HTML code related to input form. We then modify the form action URL to emulate an attacker controlled repository. The form target is modified to emulate different attack types (defined in Section 4.5. For example, a form submission results in redirecting to a legitimate website is considered the attack type $a_1$. The form is injected by providing HTML code through inputs to these programs. We deploy all the vulnerable programs in an Apache web server (version 2.0). The server resides in a host running Windows XP.

Tools chosen for comparative evaluation. We evaluate PhishTester against heuristic-based tools for login functionality. The primary reason for choosing heuristic-based tools is that our testing approach is based on heuristics that are related to program...
Table 15
Characteristics of XSS vulnerable programs.

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>OSVDB ID</th>
<th>Vulnerable file</th>
</tr>
</thead>
<tbody>
<tr>
<td>MaxGuestbook-0.9.3</td>
<td>A simple guestbook</td>
<td>50654</td>
<td>maxGuestbook.class.php</td>
</tr>
<tr>
<td>InsanelySimpleBlog-0.5</td>
<td>A simple user blog</td>
<td>38210</td>
<td>index.php</td>
</tr>
<tr>
<td>MercuryBoard 1.1.5</td>
<td>A message board</td>
<td>41479</td>
<td>index.php</td>
</tr>
</tbody>
</table>

Table 16
Comparison of PhishTester with NetCraft and SpoofGuard.

<table>
<thead>
<tr>
<th>Attack type</th>
<th>NetCraft</th>
<th>SpoofGuard</th>
<th>PhishTester</th>
</tr>
</thead>
<tbody>
<tr>
<td>a1</td>
<td>N</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>a2</td>
<td>N</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>a3</td>
<td>N</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>a4</td>
<td>N</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>a5</td>
<td>N</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>a6</td>
<td>N</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>a7</td>
<td>N</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>a8</td>
<td>N</td>
<td>N</td>
<td>Y</td>
</tr>
</tbody>
</table>

6. Conclusions

Phishing is a growing problem for website users. Unfortunately, most of the current anti-phishing tools are geared towards end users, and there exist a lack of effort to automate the task of anti-phishing professionals who manually verify a reported website. Moreover, current tools do not address phishing attack detection that might be performed by exploiting cross site scripting vulnerabilities in trusted websites. Testing of phishing websites is challenging due to uncertainty of website behaviors for random inputs and naturally fits well with the notion of trustworthiness testing. Here, trustworthiness testing is intended to check whether a program is performing functionalities against a set of benchmark or human knowledge of program behaviors. We propose trustworthiness testing of suspected phishing websites to mitigate these issues. We consider suspected websites as web-based programs. These programs should demonstrate different behaviors or responses with respect to random input submissions among phishing and legitimate websites. We model such behaviors with the notion of Finite State Machine (FSM). We then develop eight heuristics based on state, submission response, and form-based features. To facilitate the testing, we further develop seven heuristic combinations for testing phishing and legitimate websites. We implement a prototype tool named PhishTester in Java. The approach has been evaluated 33 unique phishing websites and 19 legitimate websites that belong to 45 organizations. These websites are related to login and registration functionalities. The results show that our proposed combination of heuristics can accurately detect all phishing (i.e., zero false negative) and legitimate websites (i.e., zero false positive). We further compare PhishTester with two popular heuristic-based anti-phishing tools and notice that the proposed approach can warn about potential phishing sites residing in trusted websites. Our approach is complementary to traditional anti-phishing tools that are targeted to save victims.

Our future work includes developing further heuristics to detect phishing attacks in the presence of embedded objects (e.g., Flash). Moreover, we plan to automatically report suspected websites to anti-phishing communities and notify ISP administrators by extending the PhishTester tool. Our current implementation does not handle random input submission to forms that contain captchas (i.e., a distorted image containing alphabet and numbers to disallow automatic form submission by programs). We notice that none of the phishing websites contain any captcha and most legitimate websites still do not generate captchas during login or registration functionalities. However, we suspect that phishers might employ captchas into their websites in near future. Thus, we plan to investigate automatic testing of phishing websites in presence of captchas in future.

Acknowledgements

This work is partially supported by the Natural Sciences and Engineering Research Council of Canada (NSERC). We also thank to the anonymous reviewers for their thoughtful comments to improve this paper.

References


Hossain Shariari is a Ph.D. candidate in the School of Computing, Queen’s University, Canada, where he is a member of the Queen’s Reliable Software Technology (QRTS) research group. He received his B.Sc. from Bangladesh University of Engineering and Technology and M.Eng. from Muroran Institute of Technology, Japan. Dr. Zulkernine received his Ph.D. from the University of Waterloo, Canada, where he belonged to the university’s Bell Canada Software Reliability Laboratory. In 2009–10, he was a Visiting Professor in the Dept. of Information Engg. & Computer Science of the University of Trento, Italy. Dr. Zulkernine’s current research focuses on software reliability and security. His research projects are funded by a number of provincial and federal research organizations of Canada, while he is having industry research partnerships with Bell Canada and Cloakware Corporation. Dr. Zulkernine is a senior member of the IEEE, a member of the ACM, and a licensed professional engineer of the province of Ontario, Canada. More information about his research and publications can be found at http://www.cs.queensu.ca/~zulkernine.

Mohammad Zulkernine is a faculty member in the School of Computing of Queen’s University, Canada, where he leads the Queen’s Reliable Software Technology (QRTS) research group. He received his B.Sc. degree in Computer Science from Queen’s University, Canada, while his M.Sc. thesis research on security testing received the IEEE Kingston Section Research Excellence award. Dr. Shariari also has been awarded the prestigious Ontario Graduate Scholarship and NSERC Post Graduate Scholarship for his Ph.D. study in Ontario, Canada. He is also a student member of the IEEE and the ACM. More information about his research and publications can be found at http://www.cs.queensu.ca/~shariari.