2010 Fourth International Conference on Secure Software Integration and Reliability Improvement

PhishTester: Automatic Testing of Phishing Attacks

Hossain Shahriar and Mohammad Zulkernine
School of Computing
Queen's University, Kingston, Canada
{shahriar, mzulker}@cs.queensu.ca

Abstract*— Phishing is a web-based attack where users are allured to visit fake websites and provide their personal information. Traditional anti-phishing tools are successful to mitigate the attack partially. Most of the tools are focused on protecting users. However, 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 web applications such as cross site scripting. An attacker might generate input forms by injecting script code and steal credentials. This paper attempts to address these issues by leveraging traditional web application testing method which can be seen as a complementary effort to current anti-phishing techniques. We consider a suspected website as a web application and test the application based on a behavior model. The model is described using the notion of Finite State Machine (FSM) that captures submission of forms with fake inputs and corresponding responses. We then identify several heuristic coverage criteria to detect inconsistencies which lead to the conclusion that a website is phishing or real. We implement a tool named PhishTester to automate the testing process. We evaluate the proposed approach with both phishing and real applications. The initial results show that the approach incurs negligible false negatives (less than 3%) and zero false positive for detecting phishing and real websites, respectively. The approach can be complementary to current anti-phishing tools to discover advanced phishing attacks.

Keywords: Phishing, finite state machine, application behavior model, heuristic coveage, cross site scripting.

I. INTRODUCTION

Phishing is a web-based attack which allures end users to visit fraudulent websites and give away personal information (*e.g.*, userid, password). The information is used to perform illegitimate activities such as online banking [12]. Phishing attacks cost billions of dollars losses to business organizations and end users [28]. 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 are based on email messages. These emails urge or prompt users to take immediate actions to avoid consequences such as bank account suspensions.

Many approaches and tools have been developed in recent years to combat phishing attacks. These include detecting suspicious websites with heuristics (*e.g.*, [1, 3, 5, 7]), educating and training users to avoid the phishing attacks [30], compiling white lists [21, 22] and blacklists [23], filtering emails that contain suspected URLs [11, 17], and customizing visual cues to distinguish real websites from fake websites [24]. Most popular browsers (*e.g.*, FireFox, Internet Explorer) have built-in phishing detection abilities based on white and blacklisted web sites. However, phishers may exploit cross site scripting (XSS) vulnerabilities to inject script code to generate HTML forms [16] and frames containing input forms [31, 38]. Unfortunately, traditional anti-phishing solutions cannot detect these sophisticated attacks. Moreover, there exists no approach for anti-phishing professionals, who manually verify suspected websites, and inform administrators to take down the fake sites. The time gap between uploading a fake site and taking it down is currently good enough (around 4-5 days) to lure a number of victims to give away information [7]. This situation motivates us to devise a passive testing approach for phishing website detection. We believe that an approach focused on testing phishing sites can reduce the number of victims and prevent real web sites from being taken down unnecessarily.

In this paper, we propose a testing approach to detect phishing websites. We are motivated by a number of observations.. First, phishing websites can be considered as web applications. Given an URL, one can consider the site to be a collection of web pages. We denote this set of pages as a web application. We assume that a phishing application is implementing functionalities just like a real web application. As the intention of phishers is to grab personal information, phishers modify real web application pages such as replacing a form target URL with his desired URL. While doing so, phishers introduce inconsistencies in terms of application behaviors (*i.e.*, acceptance of any arbitrary inputs) and navigation of pages (*e.g.*, form submission might result in a page which provides no links to traverse any other page). We consider such inconsistencies as faults. Thus, our aim is to test a suspected web application for phishing by revealing these inconsistencies.

A web application testing consists of two main phases: (i) develop a behavior model of an application under test based

978-0-7695-4086-3/10 $26.00 © 2010 IEEE
DOI 10.1109/SSIRI.2010.17

on available artifacts (e.g., source code, specification), and (ii) define appropriate test coverage criteria to generate test cases that expose implementation faults with high probabilities. Unfortunately, traditional web application models (*e.g.*, [19, 35]) cannot be directly applied to model the behavior of phishing web applications as we do not have program artifacts available (*e.g.*, dynamic script code employed by phishing site). Moreover, test coverage criteria defined in these works are not suitable for detecting phishing web applications as we do not know what kind of suspected forms are present unless we visit the suspected pages. To address these challenges, we first model the behavior of suspected phishing web applications through a Finite State Machine (FSM) based on known phishing behaviors. We then propose five test coverage criteria (denoted as heuristic coverage criteria) based on the FSM model. While an application is being tested with random inputs, it is marked as phishing or real, if a particular heuristic is satisfied. If no criterion is satisfied, a manual checking is required for the final decision. 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 detection tools are good for analyzing web pages contents or search results that are presented in English [7]. We implement and evaluate a prototype tool to automate the testing of a website for phishing named **PhishTester**.

The paper is organized as follows: Section II discusses an overview of phishing attack methods. Section III describes related works that detect phishing web pages and test web applications. In section IV, we describe the proposed behavior model along with heuristic coverage criteria. Section V discusses the implementation and evaluation results of PhishTester. Finally, Section VI draws some conclusions and discusses future work.

II. PHISHING TECHNIQUE

Phishers apply a wide range of tricks 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.

Spoofing website content: Attackers hide the real 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 **) [2, 8]. For example, the code * www.goodwebsite.org * shows *www.goodwebsite.org* in a browser. However, if a user clicks the link, the next page is fetched from *www.evilwebsite.org*. Moreover, browsers ignore anything written before the @ symbol in an URL [7, 8]. 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 real URL, which might not be noticed by victims. For example, *www.paypai.com* is a spoofed anchor of *www.paypal.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 [8]. For example, one might specify in the header section 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://*) [2]. 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 real website URLs. For example, the real website of NatWest bank is *www.natwest.com*. A phisher chooses a URL name which contains part of the real URL such as *www.natwest.com.mjhhdtr.com*.

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

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 scripting languages **[**37**]**. We provide some examples below.

(vi) Customizing status bar: Phishers apply JavaScript code to generate fake addresses in a status bar. Let us consider the code snippet shown in Figure 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 browser actually visits *www.evilwebsite.com*.

```
<a href = "www.evilwebsite.com" onMouseOver =
"window.status='www.goodwebsite.com'; return true"
onMouseOut = "window.status= 'done'; return true"> click here </a>
```

Figure 1. A JavaScript code snippet for hiding a real website URL

(vii) XSS-based form: Cross site scripting (XSS) is a common vulnerability in web-based applications, 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 [29]. 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 [2, 7, 8]. In most cases, a phishing site has no Domain Name Service (DNS) record. Current techniques have addressed the detection of spoofing techniques (i) to (vi). However, current tools cannot detect attacks that use XSS-based forms (*i.e.*, vii).

III. RELATED WORK

A. Phishing detection

We first discuss some works that detect phishing web pages or websites. Table I provides a brief summary of these works in comparison to our work with respect to five features. These include detection of attacks by supplying inputs (third column), testing of multiple pages (fourth column), examination of SSL certificates (fifth column), language independency (sixth column), and detection of XSS-based attacks (the last column). We now briefly discuss these works in the following.

TABLE I. A SUMMARY OF RELATED WORK FOR DETECTING PHISHING WEB PAGES

Work	Brief description	Input supply	Multiple pages	Certificate	Language independence	XSS-based attack
McRae *et al.* [4]	Track the IP address of phishers through web bugs and honeytokens.	Yes	No	No	Yes	No
Liu *et al.* [15]	Detect fake web pages based on visual similarities.	No	No	No	Yes	No
BogusBiter [12]	Supply bogus credentials when a web page is detected as phishing to avoid information leakage.	Yes	No	No	Yes	No
PhishGuard [14]	Submit fake credentials before and after actual user credentials.	Yes	Yes	No	Yes	No
AntiPhish [3]	Compare the domain of a visited URL with trusted domain names where a user previously submitted personal information.	No	No	No	Yes	No
DOMAntiPhish [1]	Compare DOM-tree similarities between saved web pages and a new web page.	No	No	No	Yes	No
Pan *et al.* [2]	Identify anomalies in web page identities by examining HTML contents, DOM objects, and HTTP transactions.	No	No	No	Yes	No
Dong *et al.* [5]	Identify phishing web pages by computing phishing scores.	No	No	No	Yes	No
CANTINA [7]	Analyze web page contents and broken hyperlinks to extract identity and verify by a trusted search engine.	No	No	Yes	No	No
Xiang *et al.* [18]	Extract the identity of a page, obtain the true domain, and compare the result with the current domain using a search engine.	No	No	Yes	No	No
SpoofGuard [8]	Compute the spoof index of a given site based on the characteristics used in previous phishing attacks.	No	No	Yes	Yes	No
Wenyin *et al.* [13]	Detect target phishing pages by constructing a semantic link network and reasoning.	No	Yes	No	Yes	No
Ma *et al.* [9]	Classify phishing URLs from real URLs based on lexical and host-based features.	No	No	No	Yes	No
Our work	**Detect phishing attacks based on an application behavior model.**	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**

McRae *et al.* [4] identify phisher locations by filling suspected HTML forms with web bugs (*e.g.*, a one by one white pixel image) that are barely noticeable. They log the IP address of suspected phishers who try to access web bugs through browsers. Liu *et al.* [15] detect phishing web pages for a real website URL. They develop an intermediate representation (in terms of blocks and features) of a real 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 *et al.* [12] develop the Bogusbiter tool which intercepts real credential of users, generates a large number of fake credentials, and places the real credential among the fake credentials to nullify the attack. A similar approach has been applied by Joshi *et al.* [14] (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 *et al.* [3] save a mapping between supplied credentials and corresponding trusted website 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 a mismatch, a website is suspected as phishing. Rosiello *et al.* [1] improve the technique of Krida *et al.* [3] by saving not only the association between user supplied credentials and website domains, but also the DOM of trusted web pages.

Pan *et al.* [2] 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 through DOM objects and HTTP transactions (*e.g.*, server form handler). Dong *et al.* [5] 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.* [7] develop the CANTINA tool which leverages the TF-IDF (term frequency and inverse term

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.* [18] 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 previous search result. If there is no common result in the two sets, the downloaded page is suspected as phishing.

Chou *et al.* [8] develop the SpoofGuard tool that detects phishing web pages based on heuristics and computes spoof scores based on matched heuristics. The score is computed by combining stateless evaluation (*e.g.*, a page containing Amazon logo asking userid and password), stateful evaluation (*e.g.*, whether a domain already visited before), and evaluation of the supplied 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.* [13] identify a phishing page target which helps in registering real pages in advance so that anti-phishing tool (Liu *et al.* [15]) 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 reasoning, it is considered as phishing. Ma *et al.* [9] classify phishing URLs from real URLs based on a set of features which include both lexical (*i.e.*, text properties of an URL) and host-based (*e.g.*, WHOIS, IP address properties) features.

B. Web application modeling and testing

Our work is also motivated by several approaches that test web applications by modeling behaviors. A brief summary of these works along with our work is shown in Table II. We discuss these works in brief now.

Fantinato *et al.* [36] propose a FSM-based web application model that captures data flow properties of an application such as data definition and usage to identify data flow anomalies. Song *et al.* [33] develop an extended FSM to model application navigations. The model is intended to reveal faults related to forward and backward button-based page visits. Andrews *et al.* [19] 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 applications while generating system level test cases. Han *et al.* [34] propose an adaptive navigation model of web applications using Unified Modeling Language (UML) state charts to model hyperlink addresses available to users at different access modes such as *login* and *not logged in*. Leung *et al.* [35] apply state charts to model the navigation of web pages where links can be generated by HTML anchor tags or scripting languages.

TABLE II. COMPARISON OF WEB APPLICATION TESTING WORKS

Work	Test criteria	Suitable for phishing?
Fantinato *et al.* [36]	5 test criteria related to dataflow	No
Song *et al.* [33]	No	No
Andrews *et al.* [19]	All transition pairs	No
Han *et al.* [34]	No	No
Leung *et al.* [35]	No	No
Ricca *et al.* [20]	5 test criteria related to page, hyperlink, and dataflow	No
Our work	**5 heuristic coverage criteria**	**Yes**

In contrast to all the above works, our objective is to detect phishing attacks by considering application behaviors with respect to form submissions and related responses. Thus, we include only a subset of web application behaviors. Moreover, our model is derived from the known symptoms of phishing attacks as opposed to application artifacts.

IV. BEHAVIOR MODEL AND TESTING

In this section we first introduce the proposed behavior model for web applications in Section A. We then define five heuristic coverage criteria to verify phishing and real sites in Section B. Section C shows a relationship between attack types and heuristic coverage.

A. Application behavior model

We apply Finite State Machine (FSM) notion to describe an application behavior. We develop it based on known symptoms and well known behavior of real web applications with respect to form submissions. We first describe the assumptions in the FSM. A phishing site often contains pages which may have (or have no) forms. However, if a page does not contain any form, it might have HTML elements such as buttons, which enable 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 always accepted.

The FSM is denoted by $<S, s_0, I, \delta, F>$, where S is a finite set of states, s_0 is the initial state, I is a finite set of inputs, δ is the state transition function, and F is a set of final states. Figure 2 shows the state transition diagram of the FSM model, which has eight states. For the sake of presentational convenience, we denote the states from S_1 to S_8. s_0 is the initial state which belongs to $\{S_1, S_2\}$. F is the final state which belongs to $\{S_3, S_4, S_5, S_6, S_7, S_8\}$. 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_1 to x_2) and corresponding responses (denoted as y_1 to y_7), which are discussed in detail in the following paragraphs. An application is phishing, if it can reach from an initial state to one of the final states. 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 III. There are two kinds of request: 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 seven interesting response features in the following paragraphs.

TABLE III. LIST OF REQUESTS AND RESPONSES

Name	Symbol	Description
x_1	*c*	Button click
x_2	*d*, *c*	Form filling and button click
y_1	!*f*, !*e*, !(*s*, *r*, *p*), !E_{max}, !F_{max}	No form, no error message, no occurrence of certificate attribute change, redirection, or post information to third party, # of error message is not maximum, # of form submission is not maximum.
y_2	!*f*, !*e*, (*s*, *r*, *p*), !E_{max}, !F_{max}	No form, no error message, occurrence of certificate attribute change, redirection, or post information to third party, # of error message is not maximum, #of form submission is not maximum.
y_3	*f*, !*e*, !(*s*, *r*, *p*), !E_{max}, !F_{max}	Form present, no error message, no occurrence of certificate attribute change, redirection, or post information to third party, #of error message is not maximum, # of form submission is not maximum.
y_4	*f*, !*e*, (*s*, *r*, *p*), !E_{max}, !F_{max}	Form present, no error message, occurrence of certificate attribute change, redirection, or post information to third party, # of error message is not maximum, # of form submission is not maximum.
y_5	*f*, *e*, !(*s*, *r*, *p*), !E_{max}, !F_{max}	Form present, error message, no occurrence of certificate attribute change, redirection, or post information to third party, # of error message is not maximum, # of form submission is not maximum.
y_6	*f*, *e*, !(*s*, *r*, *p*), E_{max},, !F_{max}	Form present, error message, no occurrence of certificate attribute change, redirection, or post information to third party, # of error message is maximum, # of form submission is not maximum.
y_7	*f*, !*e*, !(*s*, *r*, *p*), !E_{max},, F_{max}	Form present, no error message, no occurrence of certificate attribute change, redirection, or post information to third party, # of error message is not maximum, # of form submission is maximum.

(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 form submission or a page being non-existent.
(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) $\boldsymbol{E_{max}}$: A form submission might result in an error message provided that the supplied inputs are random. Real applications provide a limited number of attempts for submitting wrong credentials. However, a phisher might attempt to confuse users by showing variable number of error messages as they cannot verify submitted information. Thus, we choose a limit on error message count (three) due to form submission.
(vii) $\boldsymbol{F_{max}}$: The maximum number of form to be submitted is finite for a real application. In contrast, a phishing site 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 to a site, which can be set as six.

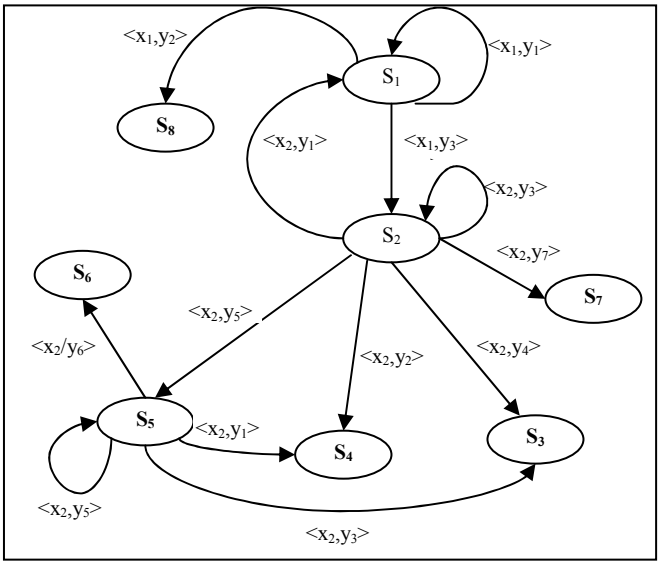

Figure 2. FSM model representing phishing attacks in web applications

For simplicity, we group *s*, *r*, and *p* as one observation, which is denoted as (*s*, *r*, *p*). This also means that a change of any of these features is treated as one observation to identity a site to be phishing. *!(s, r, p)* 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 32 possible responses with respect to the features which include *f*, *e*, (*s*, *r*, *p*), E_{max}, and F_{max}. However, in Figure 2, we only use seven interesting responses (denoted as y_1-y_7). The rest other combinations (*e.g.*, *f*, *e*, (*s*, *r*, *p*), E_{max}, F_{max}) 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 applications 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 site might follow only a subset of the diagram. An application can be created by a phisher or an application might be vulnerable to phishing attacks. In both cases, the FSM model represents an application behavior during phishing attacks.

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 heuristic coverage criteria based on the model to identify whether an application is phishing or not. The heuristics are expressed in terms of states, which we describe next.

B. Heuristic coverage criteria for detecting phishing attacks

We define four heuristic coverage criteria to detect phishing applications based on the FSM model. They are multiple states with no loop, multiple states with single loop, multiple states with multiple loops, and maximum form submission coverage. Since we want to identify whether a site is real or not, we define maximum error message coverage as another heuristic in this direction. Note that traditional test coverage criteria are used as systematic approaches for generating test cases to discover implementation faults. In contrast, the heuristic coverage is intended to discover phishing and real application. Nevertheless, these heuristics can be thought of as test generation techniques to discover phishing behaviors. It requires mapping abstract requests and responses with real requests and responses.

Multiple states with no loop: If a suspected site reaches more than one state, it indicates a phishing attack. For example, if a site reaches the state sequence (S_1, S_2, S_3), 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 $\{<x_1, y_3>, <x_2, y_4>\}$. We denote this heuristic criterion as "multiple states" in the rest of the paper.

Multiple states with single loop: If requests and the corresponding responses result in a site 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 real website. For example, an application might traverse the state sequence S_1, S_2, S_2, and S_4. Thus, it forms a loop around S_2. The corresponding test case revealing the attack is $\{<x_1, y_3>, <x_2, y_3>, <x_2, y_3>, <x_2, y_2>\}$. Note that meeting the single loop heuristic might not always indicate that a site is phishing. For example, a loop around state S_5 indicates that the site is real. In this case, maximum error message heuristic (described later in this section) also needs to be satisfied. For presentation convenience, we denote this heuristic criterion as "single loop" in the rest of the paper.

Multiple states with multiple loops: If requests and the corresponding responses result in the formation of more than one loop, a site can be suspected to be phishing. The heuristic might reveal advanced attacks that might force a victim to go through several pages where some pages may not contain forms. If a test case reaches more than one loop before reaching a final state (or exiting an application), then it satisfies multiple loop coverage. For example, the state sequence S_1, S_1, S_2, S_2, and S_4 represents a phishing attack having loops at states S_1 and S_2. The related test case is $\{<x_1, y_1>, <x_1, y_1>, <x_1, y_3>, <x_2, y_3>, <x_2, y_3>, <x_2, y_2>\}$. For presentation convenience, we denote this heuristic criterion as "multiple loops" in the rest of the paper.

Maximum form submission: A site that exceeds the maximum number of form submissions with random inputs is likely to be a phishing site. From state S_2, an example test case would be $\{<x_2, y_3>, <x_2, y_3>, <x_2, y_3>, <x_2, y_3>, <x_2, y_3>, <x_2, y_3>, <x_2, y_3>\}$.

Maximum error message: If a test case results in the maximum number of error messages, it indicates that the site is real as opposed to phishing. An example state sequence for testing a real page can be S_2, S_5, S_5, S_5, and S_5. The related test case would be $\{<x_2, y_5>, <x_2, y_5>, <x_2, y_5>, <x_2, y_5>\}$.

C. Relationship between attacks and heuristic coverage criteria

In this section, we describe how FSM-based heuristic coverage criteria can be applied to discover phishing attacks. A summary of the attacks and corresponding heuristic coverage is shown in Table IV. We divide phishing attack types into eight categories denoted as a_1 to a_8.

TABLE IV. PHISHING ATTACK TYPES AND CORRESPONDING HEURISTIC COVERAGE CRITERIA

Name	Attack type	Heuristic coverage criteria
a_1	Redirecting to real applications	Multiple states, single loop, multiple loops
a_2	SSL-based attack	Multiple states, single loop, multiple loops
a_3	Information sent to third party	Multiple states, single loop, multiple loops
a_4	No form in the beginning	Multiple states, single loop, and multiple loops
a_5	Variable error message	Multiple states single loop, multiple loops
a_6	Similar form pattern	Maximum form submission
a_7	Isolated page	Multiple states, single loop, multiple loops
a_8	Non-existent page	Multiple states, single loop, multiple loops

(i) **Redirecting to real application (a_1)**: Most phishing attacks redirect users to real sites after collecting personal information. More interestingly, this happens in the same browser window where a victim opens the phishing site. The multiple states, single loop, and multiple loops criteria satisfaction can be an indicator of such attacks.

(ii) **SSL-based attack (a_2):** Most phishing sites do not employ any SSL certificate-based communication. However, phishing sites might redirect users to real sites having SSL certificates (*e.g.*, Ebay and Paypal). Thus, multiple states, single loop, and multiple loop criteria might discover such attacks. Note that for a_1 attack type, we assume that both phishing and real sites have no SSL certificate-based encryption mechanism employed.

(iii) **Information sent to third party (a_3):** A phishing application 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, multiple states, single loop, and multiple loops criteria allow discovering the attack.

(iv) **No form in the beginning (a_4)**: To circumvent heuristics for detecting phishing pages, a recent approach employed by phisher is to develop a site whose 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 multiple states, single loop, and multiple loops criteria.

(v) **Variable error message (a_5)**: As phishing applications 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 real 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 multiple states, single loop, and multiple loops. Note that we assume that random inputs are semantically correct. For example, a random email must contain and '@' and '.' symbols in appropriate order.

(vi) **Similar form pattern (a_6)**: An attack might result in generation of one or more input 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 maximum number of form submission heuristic can reveal such attacks.

(vii) **Isolated page (a_7)**: An attack might grab user information and generate a page which has no more hyper link to visit another page. It can be detected by multiple states, single loop, and multiple loops criteria.

(viii) **Non-existent page (a_8)**: 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 real site. The attack can be detected by multiple states, single loop, and multiple loops coverage criteria.

Note that maximum error message heuristic is intended to detect real web applications that can validate user credentials. Thus, we do not map it with phishing attack types in Table IV. Moreover, we show that an attack type can satisfy multiple heuristic coverage criteria considering possible scenarios of state sequences. However, in an actual testing, an attack would satisfy at least one of the criteria cited in the third column of Table IV.

V. IMPLEMENTATION AND EVALUATION

A. Implementation

Given an URL of a website, we need to decide whether it is phishing or not. The offline 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 and input fields, construct the URLs corresponding to form submission, and fetch the next page. While doing so, the response features (*e.g.*, web site redirection) are observed by analyzing HTTP response header fields (*e.g.*, status code) and contents of new pages. To analyze HTML pages, we employ Jericho HTML parser [27] which is free and provides necessary API support for Java. If a form cannot be submitted further, then manual examination of the page is required. 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 [6] that include fake credit card numbers, expiry dates of credit cards, pin numbers, etc.

B. Evaluation

We perform two different experiments in this section to identify two issues: (i) Is the FSM-based approach effective for identifying phishing sites? (ii) Does the tool have any advantage compared to the other heuristic-based tools?

1) Tool effectiveness

We show the effectiveness of our approach by testing the reported URLs of PhishTank [32], which is free and widely used data source among phishing researchers [5, 7, 12, 25, 37, 38]. One important feature of 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 sites have been taken down by the administrators. Thus, we tested URLs that have been accessible at the time of the evaluation.

We chose 37 phishing URLs during the period July to December of 2009. Note that there have been actually more than 37 URLs that have been reported to be phishing during this time period. However, we are interested to examine unique phishing applications, which prohibit us to include a large portion of the reported URLs. Many reported URLs are actually just one unique application (*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 as a sample when multiple URLs represent the same web application. Nevertheless, the chosen web applications might not be complete as we sometimes find an application unavailable to download (*i.e.*, taken down by the related web site administrator).

Our chosen URLs belong to 20 organizations whose names and business types are shown in Table V. These include renowned banks such as Bank of America, HSBC, 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, Italian, and German. The third and fourth columns of Table V show the number of distinct phishing sites and the number of sites detected as phishing by PhishTester, respectively. The fifth column computes the false negative (FN) rate, which is the ratio of the number of sites not detected as phishing to the total number of actual phishing sites. The last column shows the overall false negative rate which is 2.70% (*i.e.*, 1/37).

TABLE V: EVALUATION RESULTS WITH PHISHTESTER

Name	Type	Phishing site	Detected as phishing	FN (%)	Real site	Detected as real	FP (%)
Paypal	**Ecommerce**	**8**	**7**	**12.5**	**1**	**1**	**0**
HSBC	Bank	1	1	0	1	1	0
Abbey	Bank	2	2	0	1	1	0
World of WarCraft	Online game	2	2	0	1	1	0
VISA	Credit	1	1	0	1	1	0
Bank of America	Bank	4	4	0	1	1	0
GameCreate	Game server management	1	1	0	1	1	0
MSN	Messenger	1	1	0	1	1	0
Rapidshare	File hosting	2	2	0	1	1	0
Ebay	Ecommerce	5	5	0	1	1	0
US Bank	Bank	1	1	0	1	1	0
Gebrauchtwagen	Ecommerce	1	1	0	1	1	0
Habbo	Social networking	1	1	0	1	1	0
Natwest bank	Bank	1	1	0	1	1	0
MBNA	Credit card	1	1	0	1	1	0
Banca Popolare Di Verona	Bank	1	1	0	1	1	0
Alliance & Leicester	Bank	1	1	0	1	1	0
American Express	Credit card	1	1	0	1	1	0
Carta Si	Credit card	1	1	0	1	1	0
CDCU	Credit union	1	1	0	1	1	0
Total		37	36	2.70	20	20	0

TABLE VI: HEURISTIC CRITERIA COVERAGE SUMMARY

Name	Multiple states	Single loop	Multiple loops	F_{max}
Paypal	7	4	2	0
HSBC	1	1	0	0
Abbey	2	0	0	0
World of WarCraft	2	1	0	1
VISA	1	0	0	0
Bank of America	4	3	0	0
GameCreate	1	1	0	0
MSN	1	1	0	1
Rapidshare	2	1	0	0
Ebay	5	3	0	0
US Bank	1	1	0	0
Gebrauchtwagen	1	0	0	0
Habbo	1	0	0	0
Natwest bank	1	1	0	0
MBNA	1	1	0	1
Banca Popolare Di Verona	1	1	0	0
Alliance & Leicester	1	0	0	0
American Express	1	0	0	0
Carta Si	1	1	0	0
CDCU	1	1	0	0
Total (%)	89.18	51.35	5.40	8.10

The first bold row (Paypal) indicates that manual testing has to be performed for that site. We notice that one of the Paypal sites has only one page having no form or button. When examining manually, we identify that the page resembles a Paypal confirmation page. However, no information is being sent to a phisher through this page. The sixth and seventh columns of Table V show the number of real website pages (login pages) and the number of pages detected as not phishing by PhishTester, respectively. The last column shows the false positive (FP) rate, which is the ratio of the number of sites detected as phishing (instead of real) to the total number of real site. We notice that the FP rate is zero (*i.e.*, 0/20).

TABLE VII: PHISHING ATTACK TYPES DETECTION SUMMARY

Name	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8
Paypal	7	7	0	1	3	0	0	0
HSBC	1	0	0	0	0	0	0	0
Abbey	2	0	0	0	0	0	0	0
World of WarCraft	0	1	0	0	0	1	0	0
VISA	0	0	0	0	0	0	1	0
Bank of America	4	3	0	0	0	0	1	0
GameCreate	1	0	0	0	0	0	0	0
MSN	0	0	0	0	0	1	0	0
Rapidshare	2	0	0	1	0	0	0	0
Ebay	5	4	0	0	0	0	0	0
US Bank	1	0	0	0	0	0	0	0
Gebrauchtwagen	1	0	0	0	0	0	0	0
Habbo	0	0	0	0	0	0	1	0
Natwest bank	1	0	0	0	0	0	0	0
MBNA	0	0	1	0	0	1	0	0
Banca Popolare Di Verona	0	0	0	0	0	0	0	1
Alliance & Leicester	1	1	0	0	0	0	0	0
American Express	1	1	0	0	0	0	0	0
Carta Si	1	1	0	0	0	0	0	0
CDCU	1	1	0	0	0	0	0	0
Total (%)	78.37	51.35	2.70	5.40	8.10	8.10	8.10	2.70

Table VI shows the heuristic criteria coverage summary for phishing sites only. The second, third, fourth, and fifth columns show the number of sites satisfied multiple states, single loop, multiple loops, and F_{max}, respectively. The last row shows the percentage of the total sites that satisfy the corresponding criteria. The result indicates that most of the current phishing sites can be discovered by multiple states and single loop heuristics. Moreover, several sites allow similar patterns of form submissions. Very few sites satisfy multiple loops heuristic. The results can be applied to prioritize heuristics to reduce the testing time.

Table VII shows a summary of attack types detected in our evaluation. We observe that most of the phishing sites redirect users to real applications (a_1). None of the phishing sites has employed SSL certificate-based communication. Some real sites do not use SSL such as Rapidshare and Habbo. However, most of the phishing sites redirect to real sites which have valid SSL certificate attributes (a_2). These cases represent the occurrence of both attack types a_1 and a_2. Moreover, variable error message (a_5), similar form pattern (a_6), and isolated page (a_7) phishing attack types are common. Several sites contain no form in the beginning (a_4). Very few sites send information to different domains (a_3) and demonstrate non-existent pages (a_8).

C. Comparison with other anti-phishing tools

All of the phishing sites tested and discussed in the previous section do not represent attacks that are due to the

exploitation of XSS vulnerabilities in trusted applications. As we intend to validate PhishTester for detecting phishing attacks that might be launched through XSS-based form injection in trusted applications, we are motivated to perform a second evaluation based on three real world applications. These applications have been reported to contain XSS vulnerabilities in Open Source Vulnerability Databases [19].

TABLE VIII: CHARACTERISITICS OF XSS VULNERABLE APPLICATIONS

Name	Type	OSVDB ID	Vulnerable file
MaxGuestbook-0.9.3	A simple guestbook	50654	maxGuestbook.class.php
Insanely SimpleBlog-0.5	A simple user blog	38210	index.php
MercuryBoard 1.1.5	A message board	41479	index.php

Table VIII shows the characteristics of the applications used in the evaluation. The vulnerable applications include guestbook, blog, and message board, where all of them 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 choose a trusted email page which is accessible by a valid user. We extract the HTML code related to input form and submission and modify the target URL to emulate an attacker controlled repository. The HTML code has been injected as part of a message for the three applications. The form target is modified set in different ways to emulate eight types of attacks (defined in Section IV). For example, a form submission results in redirecting to a real website is considered the attack type a_1. We deploy all the vulnerable applications in an Apache server. The server resides in a host running on Windows.

TABLE IX: COMPARISON OF PHISHTESTER WITH OTHER TOOLS

Attack type	NetCraft	SpoofGuard	PhishTester
a_1	N	N	Y
a_2	N	N	Y
a_3	N	N	Y
a_4	N	N	Y
a_5	N	N	Y
a_6	N	N	Y
a_7	N	N	Y
a_8	N	N	Y

We choose two heuristic-based anti-phishing tools to compare with PhishTester: *Netcraft* [10] and *SpoofGuard* [8]. They are free to download and widely used. The *Netcraft* is chosen as it not only detects phishing attacks, but also prevents XSS attacks. We believe that the tool closely matches with the objective of our approach. The SpoofGuard tool is chosen as it is a representative tool for all the heuristic-based phishing attack detection techniques. The Netcraft and SpoofGuard are installed in an IE browser and we do not modify the default settings of these tools after installation. Both of the tools generate warning dialogues, if a website is suspected as phishing. We first train these tools by visiting a trusted web page (an email login page) and actually logging followed by logging out. Next, we launch the IE browser and traverse the vulnerable applications to reach the pages displaying the injected messages. While enabling Netcraft and SpoofGuard tools, we record if any of the tools detect the application page as phishing while submitting forms. We provide the application URL reaching the injected message to PhishTester and observe the detection results. Table IX shows the comparison summary of the detection of attack types, where "Y" indicates that an attack injected page results in a warning and "N" implies that an attack injected page generates no warning.

We observe that Netcraft cannot detect XSS attacks that transfer user name and password information to third parties (*i.e.*, a_3). Moreover, it cannot detect any of the other seven attacks. SpoofGuard detects phishing attacks based on heuristics and a predefined number of warning signs such as matching current URL with previously visited URLs, presence of password field in HTML form, current image with previously visited sites image, and email referral. We note that phishing techniques can easily skip such heuristics. We found that the tool does not warn, if a form is filled out and sent to an attacker server. In contrast, PhishTester discovers the application sites suspected to be phishing. Therefore, PhishTester can detect advanced phishing techniques and play a complementary role, if it is used with the other anti-phishing techniques.

VI. CONCLUSIONS

Unfortunately most 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 web site. Moreover, current tools do not address phishing attack detection that might be performed by exploiting cross site scripting vulnerabilities in trusted applications. We propose an automated testing approach to mitigate these issues. Our approach is complementary to traditional detection tools targeted to save victims. We apply testing approach for suspected phishing websites by considering suspected sites as web applications. These applications demonstrate specific behaviors or responses with respect to input submissions. We model such application behavior with the notion of Finite State Machine (FSM). We then develop five heuristic coverage criteria to identify whether an application is phishing or real. We implement a prototype tool named PhishTester in Java. The approach has been evaluated against 37 reported phishing web applications belonging to 20 organizations. Our approach shows a lower false negative rate which is less than 3% and a zero false positive rate. 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 future work includes developing further heuristics to detect phishing attacks in the presence of embedded objects (*e.g.*, Flash) and malformed websites. Moreover, we plan to automatically report suspected web applications to anti-

phishing communities and notify ISP administrators by extending the PhishTester tool.

ACKNOWLEDGMENT

This work is partially supported by the Natural Sciences and Engineering Research Council of Canada (NSERC) and the Ontario Graduate Scholarship (OGS).

REFERENCES

[1] A. Rosiello, E. Krida, C. Kruegel, and F. Ferrandi, "A Layout-Similarity-Based Approach for Detecting Phishing Pages," *Proc. of the 3rd International Conference on Security and Privacy in Communication Networks*, Nice, France, Sept 2007, pp. 454-463.

[2] Y. Pan and X. Ding, "Anomaly-based Web Phishing Page Detection," *Proc. of the 22nd Annual Computer Security Applications Conference*, Miami, Florida, December 2006, pp. 381-392.

[3] E. Krida and C. Kruegel, "Protecting Users Against Phishing Attacks with AntiPhish," *Proc. of the 29th Annual International Computer Software and Applications Conference*, July 2005, Edinburgh, Scotland, pp. 517-524.

[4] C. McRae and R. Vaughn, "Phighting the Phisher: Using Web Bugs and Honeytokens to Investigate the Source of Phishing Attacks," *Proc. of the 40th Hawaii International Conference on System Sciences*, Hawaii, USA, January 2007, pp. 270c-270c

[5] X. Dong, J. A Clark, and J. Jacob, "User Behavior-based Phishing Websites Detection," *Proc. of International Multiconference on Computer Science and Information Technology*, Wisla, Poland, October 2008, pp. 783-790.

[6] Credit Card Number Generator - Test Data Generation, http://sqa.fyicenter.com/Online_Test_Tools/__Test_Credit_Card_Number_Generator.php

[7] Y. Zhang, J. Hong, and L. Cranor, "CANTINA: A Content-based Approach Detecting Phishing Websites," *Proc. of the 16th Intl. Conf. on World Wide Web*, Banff, Alberta, May 2007, pp. 639-648.

[8] N. Chou, R. Ledesma, Y. Teraguchi, D. Boneh, and J. Mitchell, "Client-side Defense Against Web-based Identify Theft," *Proc. of the 11th Annual Network and Distributed System Security Symposium (NDSS '04)*, San Diego, CA, February 2004.

[9] J. Ma, L. Saul, S. Savag, and G. Voelker, "Beyond Blacklists: Learning to Detect Malicious Web Sites from Suspicious URLs," *Proc. of the 15th International Conference on Knowledge Discovery and Data Mining*, Paris, France, pp. 1245-1254.

[10] B. Wardman and G. Warner, "Automating Phishing Website Identification through Deep MD5 Matching," *eCrime Researchers Summit*, Atlanta, USA, October 2008, pp. 1-7.

[11] M. Chandrasekaran, R. Chinchani, and S. Upadhyaya, "PHONEY: Mimicking User Response to Detect Phishing Attacks," *Proc. of Intl. Symposium on World of Wireless, Mobile and Multimedia Networks*, 2006, Niagara-Falls, NY, June 2006, pp. 668-672.

[12] C. Yue and H. Wang, "Anti-Phishing in Offense and Defense," *Proc. of the Annual Computer Security Applications Conference (ACSAC)*, Anaheim, California, December 2008, pp. 345-354

[13] L. Wenyin, N. Fang, X. Quan, B. Qiu, and G. Liu, "Discovering Phishing Target based on Semantic Link Network," *Future Generation Computer Systems*, Elsevier, Volume 26, Issue 3, March 2010, pp. 381-388.

[14] Y. Joshi, S. Saklikar, D. Das, and S. Saha, "PhishGuard: A Browser Plug-in for Protection from Phishing," *Proc. of the 2nd International Conference on Internet Multimedia Services Architecture and Applications*, Bangalore, India, December 2008, pp. 1-6.

[15] W. Liu, G. Huang, G. Huang, and A. Fu, "An Antiphishing Strategy Based on Visual Similarity Assessment," *IEEE Internet Computing*, Volume 10, Issue 2, March 2006, pp. 58-65.

[16] G. Zuchlinski, The Anatomy of Cross Site Scripting, November 2003.

[17] I. Fette, N. Sadeh, and A. Tomasic, "Learning to Detect Phishing Emails," *Proc. of the 16th Intl. Conf. on World Wide Web*, Banff, Alberta, Canada, May 2007, pp. 649-656.

[18] G.Xiang and J. Hong, "A Hybrid Phish Detection Approach by Identity Discovery and Keywords Retrieval," *Proceedings of the 18th International Conference on World Wide Web*, Madrid, Spain, April 2009, pp. 571-580.

[19] A. Andrews, J. Offutt, and R. Alexander, "Testing Web Applications by Modeling with FSMs," *Journal of Software and System Modeling*, 4(2):326–345.

[20] F. Ricca and P. Tonella, "Analysis and Testing of Web Applications," *Proc. of the 23rd International Conference on Software Engineering*, Toronto, Canada, May 2001, pp. 25-34.

[21] Y. Wang, R. Agrawal, and B. Choi, "Light Weight Anti-Phishing with User Whitelisting in a Web Browser," *Proc. of the IEEE Region 5 Conference*, Kansas City, April 2008, pp. 1-4.

[22] J. Kang and D. Lee, "Advanced White List Approach for Preventing Access to Phishing Sites," *Proc. of the Intl. Conf. on Convergence Information Technology*, Korea, November 2007, pp. 491-496.

[23] M. sharifi and S. Siadati, "A Phishing Sites Blacklist Generator," *Proceedings of International Conference on Computer Systems and Applications (AICCSA)*, Doha, Qatar, April 2008, pp. 840-843.

[24] R. Dhamija and J. Tygar, "The battle against phishing: Dynamic Security Skins," *Proc. of the Symposium on Usable Privacy and Security*, Pittsburgh, USA, July 2005, pp. 77-88.

[25] S. Sheng, B. Wardman, G. Warner, L. Cranor, J. Hong, and C. Zhang, "An Empirical Analysis of Phishing Blacklists," *Proc. of CEAS 2009 – 6th Conf. on Email and Anti-Spam*, July 2009, California, USA.

[26] M. Wu, R. Miller, and G. Little, "Web Wallet: Preventing Phishing Attacks by Revealing User Intentions," *Proc. of the Symp. On Usable Privacy and Security*, July 2006, Pennsylvania, pp. 102-113.

[27] Jericho HTML Parser, Accessed from http://jericho.htmlparser.net

[28] D. Geer, "Security Technologies go Phishing," *Computer Archive*, Volume 38, Issue 6, June 2005, pp. 18-21.

[29] T. Beardsley, Evolution of Phishing Attacks, January 2005, Accessed from www.antiphishing.org/Evolution of Phishing Attacks.pdf

[30] D. Irani, S. Webb, J. Giffin, and C. Pu, "Evolutionary Study of Phishing," *Proc. of the 3rd Anti-Phishing Working Group eCrime Researchers Summit*, October 2008, Atlanta, Georgia, pp. 1-10.

[31] Nexus, Applying XSS to Phishing Attacks, April 2007, http://nexus.playhack.net

[32] PhishTank, Access form phishtank.org

[33] B. Song, H. Miao, and S. Chen, "Modeling Web Browser Interactions and Generating Tests," *Proc. of the Intl. Conf. on Computational Intelligence and Security*, Suzhou, China, Dec. 2008, pp. 399-404.

[34] M. Han and C. Hofmeister, "Relating Navigation and Request Routing Models in Web Applications," *Proc. of the 10th Intl. Conference on Model Driven Engineering Languages and Systems*, Nashville, USA, September 2007, pp. 346-359.

[35] K. Leung, L. Hui, S. Yiu, and R. Tang, "Modeling Web Navigation by StateChart," *Proc. of the 24th International Computer Software and Applications Conference*, Oct 2000, Taipei, Taiwan, pp. 41-47.

[36] M. Fantinato and M. Jino, "Applying Extended Finite State Machines in Software Testing of Interactive Systems," *Lecture Notes in Computer Science* 2844, pp. 34-45, 2003.

[37] C. Ludl, C. Krueger, S. McAllister, E. Kirda, "On the Effectiveness of Techniques to Detect Phishing Sites," *Proc. of the 4th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, Switzerland, pp 20-39, 2007.

[38] Y. Zheng, S. Egelman, L. Cranor, and J. Hong, "Phinding Phish: Evaluating Anti-Phishing Tools," *Proc. of the 14th Annual Network and Distributed System Security Symposium*, San Diego, Feb 2007.