Model Checking for the Defense against Cross-site Scripting Attacks

Yu Sun
School of Information Science & Technology
Southwest Jiaotong University
Chengdu, China
sunyu212@gmail.com

Dake He
School of Information Science & Technology
Southwest Jiaotong University
Chengdu, China
dkhe_scce@swjtu.cn

***Abstract*—The scripting languages (mostly JavaScript) have been widely used in the network application, for the sake of improvement of the user experience. This makes Cross-Site Scripting (XSS) attacks one of the most serious threats to Internet. A model checking method for the defense against cross-site scripting attacks is proposed in the paper. Bugs of the e-commercial website are found and counterexamples are showed by model checking. An operation behavior is judged if it conforms to requirements of the website for legal behavior, so as to prevent XSS attacks from the point of operation. The automatic modeling algorithm for the HTML code is proposed and the case of the performance of the algorithm is presented.**

Keywords-Cross-Site Scripting attacks; model checking; HTML; scripting languages; JavaScript

I. INTRODUCTION

Network applications are becoming one of the most important ways to providing network services now. In the network application, in order to improve the user experience, there is a trend that scripting languages (mostly JavaScript) have been widely used. However, this trend also makes XSS (Cross-Site Scripting attacks) one of the most serious threats to Internet. XSS attacks is to reveal the most direct harm to the user privacy of sensitive information, and make users' personal computers infected with viruses. Some famous social networking sites, such as Facebook, MySpace and Twitter, have been suffered XSS attacks. XSS has the features of self spread and fast spread, and simple implementation as well, so it attract more and more attention.

XSS attacks are essentially referred to the illegal scripts injected in a web page. When a user browses the page suffered XSS attacks, the scripts embedded in the web page will be triggered, resulting in some malicious attack effect. The way to inject malicious scripts into the web page can be classified into two categories:

(1) Persistent XSS attacks

Persistent XSS attacks [1][2] are referred in the literature as stored XSS attacks as well. Attackers inject malicious code into a permanent page, without being discovered and fixed, the malicious code potentially harmful. The real attack action occurs when a vulnerable user requests to access the web page. For example, attackers posted this message in a forum with vulnerability: <script> window.open ("www.evil.com") </script>. Assuming this script will not be found and removed. This information contains the malicious JS code will be stored on the server of the forum. Subsequently, the client browser of user who accesses the message will execute this malicious code embedded in the message. The result of the execution is to open a malicious page (i.e. www.evil.com), may lead to worms, viruses and other follow-up questions.

(2) Non-persistent XSS attacks

Non-persistent XSS attacks [3] are also referred in the literature as reflected XSS attacks. Different from the persistent XSS attacks that inject malicious code into the resources of web application, in non-persistent XSS attacks the malicious code involved reflect to the client directly. For example, the attacker tricks a user into clicking a malicious link by spam. If the user is fooled, malicious code will be included in the request as the accessory of the malicious link and is transmitted to the server of the trusted site, and then transmitted to the client as the accessory of the response from the server side. The malicious code embedded is executed in the client browser at the end.

The common defense methods against XSS attacks at present are: (1) to prevent malicious JS code injected into the server's database, such as content filtering is one of the representative approach [4-8]; (2) browser enhancement methods to enhance the browser's performance in some areas, preventing users from accessing untrusted web pages, the typical methods such as white list method [9-11]. Most of the existing methods are to prevent the malicious JS code to run the method from the point of intrusion (entrance). For the e-commerce systems and the online banking systems with higher security requirements on the website, the XSS defense approach based on the website behavior model is proposed in our previous paper. The website behavior model is generated with the analysis of the logical structure of the website and the operations that website allows users to do. From the perspective of behavior, an operation behavior is judged if it conforms to requirements of the website for legal behavior. The behavior that does not meet the corresponding website behavior model will be regarded as illegal behavior and ceased. This can be implemented from the point of operation to prevent malicious JS code to cause harm to clients. In the process of implementation of the method, due to the current diversity of sites and web technology, it is very difficult and very inefficient to model manually. Thus we need the automatic modeling methods for HTML to achieve automatic modeling of user website. The website behavior model generated in this way reflects the logic structure of the website and the possible action sequences that allow the user

978-0-7695-4719-0/12 $26.00 © 2012 IEEE
DOI 10.1109/CSSS.2012.537

to do. Bugs of the website are found and counterexamples are showed with model checker SMV based on the website behavior model expressed in CTL. If the user's browsing behavior is legal or not is judged by the detecting of browsing behavior and the website behavior model.

II. PRELIMINARIES

The website behavior model is a standardized and formal representation of the user behavior and logical structure of the website, for the interaction between the user browser and the web server. The website behavior model is used to describe the legitimate behaviors that the website permits, in other word, only the behaviors that are addressed in the definition of the website behavior model are the legitimate behaviors that the website permits.

The general process of the modeling of website is as follows: first by getting the website source code, the HTML code of all the web page of the website is obtained via the execution of website in the website execution simulator; then based on the obtained original HTML code for the website, the website behavior model is generated automatically by the automatic modeling method proposed in the paper.

For ease of presentation, the website behavior model M can be presented as the form of quadruple:

$M(s_0, A, S_t, S_s)$, where $s_0 \in S_s$.

s_0 denotes the initial state of the website behavior model, S_t denotes all the non-steady state of the website behavior model, S_s denotes all the steady state of the website behavior model, and A denotes all the Behaviors of the website behavior model, i.e. there is $a \in A$, $s_i, s_j \in S_t \cup S_s$, so that $a(s_i) = s_j$. The definitions related to website behaviors are presented as follows,

Steady State: refers to the web page that the website can stay for a long time and in a stable state.

Initial State: refers to the initial steady state after login to the corresponding website, usually refers to the home page of the corresponding website.

Non-steady State: refers to the transitional state of the website due to the occurrence of the action, it is an instant state.

Action: refers to the interactions like the click and input of user, or the actions or events that related to the jump of the web page and the submission of the form.

Action sequence: there exists a A_0 $\{a_1, a_2, ..., a_n\}$ contained in A, $s_i, s_j \in S_s$, so $a_n(...a_2(a_1(s_i))...) = s_j$, then A_0 is called a action sequence.

The modeling for the website behavior is the process to build the model structure consists of the above-mentioned elements, by the analysis of the original HTML code of the website. An examples of the website behavior model of some website is shown in Figure 1, the circles in the figure represent states, where the one with solid line represents steady state (only the URL type of state is the steady state); the one with dashed line represents non-steady state, including *btnClicked* (after click the button), *hrefClicked* (after click the hyperlink) and *Post* (after submit data) these three types of non-steady state. The figures in the circle represent the sequence number of the web page. The web page will be in different states after different actions occur. The rectangles in the figure represent actions, usually including *URL* (web page jumping), *Post* (submitting data), *btnClick* (clicking button), *hrefClick* (Clicking hyperlink) and other types of action. The combination of several different types of action sequence composes the structure of the website behavior model.

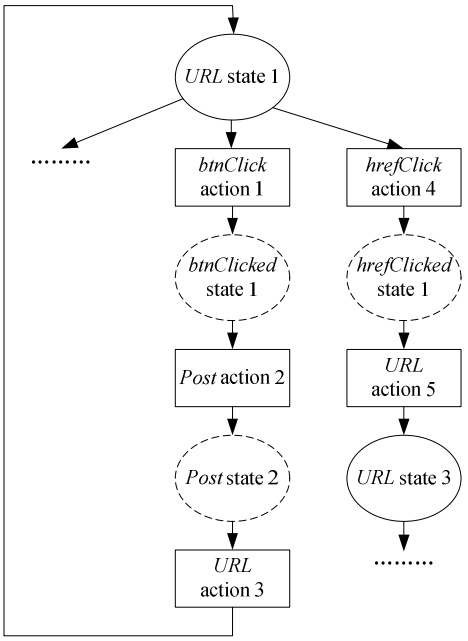

Figure 1. An example of the website behavior model (in part)

III. SYSTEM IMPLEMENT

The website behavior model is stored in the form of XML file in order to facilitate the client browser to recover the model in a unified manner. Therefore, the modeling of website refers to the automatic building of the XML file which described the structure and content of the website behavior model from the HTML code for the website. The XML file for the website behavior model of some e-commercial website is demonstrated in Figure 2.

```
<Model>
<States>
        <state>
                <id>0</id>
                <name>Home</name>
                <type>URL</type>
                <feature>http://127.0.0.1/Home/*$</feature>
                <actions>
                        <action>
                                <type>hrefClick</type>
                                <outstates>
                                        ...
                                </outstates>
                        </action>
                </actions>
        </state>
        <state>
                ...
        </state>
        ...
</States>
</Model>
```

Figure 2. The XML storage of the website behavior model

For the convenience of narration, the concepts about the modeling for the website behavior are defined and expressed as follows,

urlQueue: URL queue
MainURL: Home URL
States: The set of all the site states $S_s \cup S_t$
s_i: state i, $s_i \in S_s \cup S_t$
actions: A set of actions
outstates: The state set that after some kind of action occurring on some condition
<*a*>: the *a* tag in the HTML
<*form*>: the *form* tag in the HTML
<*input*> : the *input* tag in the HTML
<*button*>: the *button* tag in the HTML

Based on above definitions, an automatic modeling algorithm for the HTML code of the website is presented as follows,

An automatic modeling algorithm for the HTML code

Step 1: Create a state of URL type, namely, s_0; i = 1, *MainURL* is added into *urlQueue*.

Step 2: Remove the first URL from the head of *urlQueue*, record the state s_j of the corresponding URL type in $S_s \cup S_t$, and obtain the HTML code of the corresponding web page. If *urlQueue* is empty, then modeling is complete.

Step 3: Scan the HTML code, if <*a*> is found then go to the step 4; if <*form*> is found and its method attribute is *POST* then go to the step 5. After all the HTML code is scanned, go to the step 2.

Step 4: Create a s_i of *hrefClicked* type, and a corresponding s_{i+1} of URL type. If s_{i+1} does not belong to $S_s \cup S_t$, then s_{i+1} is added into $S_s \cup S_t$, and the *href* attribute of <*a*> is added into *urlQueue* as well. The *action* of URL type is added into the *actions* of s_i, and s_{i+1} is added into the *outstates* of the *action*. The *action* of *hrefClick* type is added into the *actions* of s_j , s_i is added into the *outstates* of the *action*. $i = i + 2$ and go back to the step 3 to continue scanning.

Step 5: Create a s_i of *POST* type. If s_i does not belong to $S_s \cup S_t$, then it is added into $S_s \cup S_t$. And continue scanning in the sub-label of <*form*>. If <*input*> or <*button*> is found then create a s_{i+1} of *btnClicked* type, a *action* of *POST* type is added into the *actions* of s_{i+1} and s_i is added into the *outstates* of it. The *action* of *btnClick* type is added into the *actions* of s_j and s_{i+1} is added into the *outstates* of it. $i = i + 2$; $j = j + 2$. Go back to the step 3 to continue scanning.

Note that in the HTML5 specification, the treatment of such labels as <*area*>, <*base*> is similar to that of label <*a*>.

The performance of the automatic modeling algorithm for HTML code is demonstrated with an example of simple HTML code shown in Figure 3.

```
<html>
        <head>
                <title>Title of page</title>
        </head>
        <body>
                <h2>This is my homepage. </h2>
                <p>
                <a href='http://127.0.0.1/a'>click here</a>
        </body>
</html>
```

Figure 3. An example of simple HTML code

The simple HTML code shown above is modeled via the automatic modeling algorithm for HTML code, and the result of the modeling (the website behavior model, stored in the XML file) is shown in Figure 4.

```
<Model>
<States>
        <state>
                <id>0</id>
                <name>Home</name>
                <type>URL</type>
                <feature>http://127.0.0.1/Home/*$</feature>
                <actions>
                        <action>
                                <type>hrefClick</type>
                                <outstates>
                                        <id>1<id>
                                </outstates>
                        </action>
                </actions>
        </state>
        <state>
                <id>1</id>
                <name>HomeHref1</name>
                <type>hrefClicked</type>
                <feature></feature>
                <actions>
                        <action>
                                <type>URL<type>
                                <outstates>
                                        <id>2</id>
                                </outstates>
                        </action>
                </actions>
        </state>
        <state>
                <id>2</id>
                <name>A</name>
                <type>URL</type>
                <feature>http://127.0.0.1/a/*$</feature>
                <actions>

                </actions>
        </state>
</States>
</Model>
```

Figure 4. Modeling results (the website behavior model, stored in the XML file)

Then the website behavior model can be translated into the model expressed in CTL (Computation Tree Logic) [12]. With this model in CTL, bugs of the website are found and counterexamples are showed using model checker SMV (Symbolic Model Verifier) [13]. By this way, bugs of some e-commercial websites related to transaction process which can lead to the illegal scripts injected in a web page were found before the websites online.

IV. CONCLUSIONS

In the network application, in order to improve the user experience, the scripting languages (mostly JavaScript) have been widely used; this makes XSS (Cross-Site Scripting attacks) one of the most serious threats to Internet. An automatic modeling method for the defense against cross-site scripting attacks is proposed in the paper. Bugs of the website are found and counterexamples are showed with model checker SMV based on the website behavior model expressed in CTL. An operation behavior is judged if it conforms to requirements of the website for legal behavior, so as to prevent XSS attacks from the point of operation. The automatic modeling algorithm for the HTML code is proposed and the case of the performance of the algorithm is presented.

REFERENCES

[1] Alcorna, W. Cross-site scripting viruses and worms–a new attack vector. Journal of Network Security, 2006(7):7–8, Elsevier, July 2006.

[2] Anupam, V. and Mayer, A. Secure Web scripting. IEEE Journal of Internet Computing, 2(6):46–55, IEEE, 1998.

[3] Jagatic, T., Johnson, N., Jakobsson, M., & Menczer, F. Social phishing. Communications of the ACM, 5(10), 94–100, 2007.

[4] Ismail, O., Etoh, M., Kadobayashi, Y., and Yamaguchi, S. A Proposal and Implementation of Automatic Detection/Collection System for Cross-Site Scripting Vulnerability. The 18th Internation Conference on Advanced Information Networking and Applications (AINA 2004), 2004.

[5] Kirda, E., Kruegel, C., Vigna, G., and Jovanovic, N. Noxes: A client-side solution for mitigating cross-site scripting attacks. The 21st ACM Symposium on Applied Computing, 2006.

[6] Pietraszeck, T. and Vanden-Berghe, C. Defending against injection attacks through context-sensitive string evaluation. Recent Advances in Intrusion Detection (RAID 2005), pp.124–145, 2005.

[7] Scott, D. and Sharp, R. Abstracting application-level web security. The 11th Internation Conference on the World Wide Web, pp. 396–407, 2002.

[8] Su, Z. and Wasserman, G. The essence of command injections attacks in web applications. The 33rd ACM Symposium on Principles of Programming Languages, pp. 372–382, 2006.

[9] Hallaraker, O. and Vigna, G. DetectingMalicious JavaScript Code in Mozilla. The 10th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS'05), pp.85–94, 2005.

[10] Jim, T., Swamy, N., Hicks M. Defeating Script Injection Attacks with Browser-Enforced Embedded Policies. International World Wide Web Conference, WWW2007, May 2007.

[11] Jovanovic, N., Kruegel, C., and Kirda, E. Precise alias analysis for static detection of web application vulnerabilities. 2006 Workshop on Programming Languages and Analysis for Security, pp. 27–36, USA, 2006.

[12] E.M. Clarke, E.A. Emerson, and A.P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. ACM Transactions on Programming Languages and Systems, 8(2):244-63, April 1986.

[13] K.L. McMillan. Symbolic Model Checking. Kluwer Academic Publishers, 1993.