A Proposal and Implementation of Automatic Detection/Collection System for Cross-Site Scripting Vulnerability

Omar ISMAIL Masashi ETOH Youki KADOBAYASHI Suguru YAMAGUCHI
Graduate School of Information Science Nara Institute of Science and Technology
Ikoma, Nara 630-0192
{isumai-u, masash-e, youki-k, suguru} @is.aist-nara.ac.jp

Abstract

Cross-site scripting (XSS) attacks target web sites with Cookie-based session management, resulting in the leakage of privacy information. Although several server-side countermeasures for XSS attacks do exist, such techniques have not been applied in a universal manner, because of their deployment overhead and the poor understanding of XSS problems. This paper proposes a client-side system that automatically detects XSS vulnerability by manipulating either request or server response. The system also shares the indication of vulnerability via a central repository. The purpose of the proposed system is twofold: to protect users from XSS attacks, and to warn the web servers with XSS vulnerabilities.

1 Introduction

Hyper Text Transfer Protocol (HTTP) is an Internet client/server protocol designed for the rapid and efficient delivery of hypertext materials. Unfortunately, HTTP is a stateless protocol and provides neither support for sessions, nor it knows what HTTP request and response messages actually belong to the same user. In order to taking care of the stateless or connectionless problem between client and server, the cookie is defined to solve the problem.

The cookie is a general mechanism which server side connection (such as CGI scripts) can use to both store and retrieve information on the client side of the connection. To put it more plainly, the cookie is a mechanism that allows web site to record your comings and goings, usually without one’s knowledge or consent. Web servers use cookie that may contain the user’s name and credit card numbers[5]. Although this would be convenient for clients, it would also be risky. Since they are stored and transmitted in plain, cookies are readable and of course, forgeable. The cleartext nature of the cookie implies that a malicious intermediary between the client and the server would be able to intercept and modify the cookies. Therefore the standard specification of the cookie emphasizes that information of personnel and financial nature should only be sent over a secure channel[13]. However, even if the communication channel is secure (such as SSL), cookies can still be easy targets on users’ computers. There are various ways for a malicious party to steal this kind of information from the users’ personal computers, ranging from Trojan horses to Javascript bug exploits. For example, the cross-site scripting (XSS) is a very popular and effective attacking technique used by the malicious third party.

XSS is a web application level vulnerability that can be used by the malicious third party to easily bypass the cookie protection mechanism. Since the vulnerability resides at the web server side, various server side solutions are proposed for protecting users from the XSS attack. But most of them usually degrade the server performance gracefully and cause tremendous configuration overhead.

The contribution of this paper is a client-side solution that automatically detects XSS vulnerability by manipulating either client request or server response using user side local proxy servers, the proof of its correctness, and evaluation results demonstrating the effectiveness of this solution. The rest of the paper is organized as follows. In section 2 we briefly introduce XSS vulnerability and the related works. In section 3 we present our proposed system in detail. In section 4 we discuss the implementation of our proposed system. In section 5 we explain how we evaluate the system. Finally, section 6 concludes this paper and briefly discuss our future work[6].
2 Cross-Site Scripting (XSS)

We start by briefly explaining the XSS vulnerability and how the attack is carried out, as well as a brief survey on current solutions and problems.

2.1 Cross-Site Scripting Vulnerability

On February 20, 2000, CERT published information on newly identified security vulnerability affecting all web server products[7]. This vulnerability, known as Cross-Site Scripting (XSS), results when web applications mistakenly trust data returned from clients. For example, the URL field of a web site can be used to insert executable scripts.

As mentioned previously, XSS occurs when a web application gathers malicious data from attackers. The data is usually gathered in the form of hyperlink which contains malicious content (e.g., Javascript) within it. The attackers may put the link in a website, web boards or in an email, once the users click on the link, the request message with the malicious script will be sent to web applications (e.g., web servers). After the data is collected by the web application, it generates an output page for the users which contains the malicious scripts but it appears as valid contents from the websites. Without any security consideration, the user’s browsers will execute the malicious scripts. In short, servers that embed browser input into dynamically generated HTML pages can be manipulated into becoming a launch pad for running an attacker’s malicious code.

Servers that use static pages are immune to this type of attack because they have full control over how their web pages will be interpreted. The attacker does not modify the content of the web sites. The attacker merely inserts new scripts that can be executed by a browser. Therefore, the client’s information is the main target for XSS attacks, such as, the cookie and the data in the hidden field.

2.2 XSS Attack and Cookie Stealing

As illustrated in Fig. 1, ‘a’ is a user, ‘b’ is a web page which contains the link (the link is shown in Fig. 3) of attacker’s malicious scripts while the ‘c’ represents a trusted but XSS vulnerable server.

1. while the user visits the web page b,
2. the user may click on the link with the malicious script embedded;
3. then the request with embedded malicious scripts is sent to the web server c;
4. the trusted web server generates the response with malicious scripts and the user’s browser runs the malicious scripts without any security restrictions.

Web servers generate both text and HTML markup on its response pages. The client’s browsers then interpret the web pages. HTML uses special tags to distinguish text from markup language. Different characters are special at different points in the document which is depending on the grammar. The less-than sign \(<\) usually indicates the beginning of a HTML tag. A HTML tag can affect the formatting of the page or introduce a code that will be executed by the browsers. Such as in Javascript specification, \(<\text{SCRIPT}>\) and \(<\text{/SCRIPT}>\) indicate the beginning and ending of the Javascript code respectively.

For example, if the script in Fig. 2 is inserted in the text area of a searching engine, that might result in exposure of the cookie data in user’s computers.

When web servers generate pages by inserting dynamic data into a template, it should be checked to ensure that the data to be inserted does not contain any special characters but this is not always the case. Also, the user’s web browsers could mistake any special characters as HTML markup. This would result in the browsers mistaking some data values as HTML tags or scripts instead of displaying them as text. An attacker can choose the data that the web servers inserts into the web page, thereby tricking the user’s browsers into running malicious scripts or codes.

2.3 Related Works

Scott and Sharp[1] provide a web application input validation mechanism—a rule-based web applications-level firewall— to protect against XSS attacks. However, to adapt
this mechanism to web application requires that rules be defined for every single data entry point, a difficult task for web applications that have been developed over a long time period, since they often contain a complex structures with little documents. JWIG[2] projet provides web application input validation mechanism. However, it only works with web applications developed with the Java JWIG extension. Some software engineering approaches are also proposed such as WAVES[3], OWASP[4] for security assessment. However, while how to protect users against XSS attacks is one of their main tasks, they are not built for detecting XSS vulnerabilities on the Internet. On the server side, using proxy servers as application-level firewalls to filter out the malicious code is a common mechanism in most of the server side proposals, as shown in Fig. 4. However, one thing should not be avoided when discussing the server side solution, the performance. The resource consuming content checking and filtering of XSS in the server side can severely degrade the performance of the web server. This approach also should not be recommended as an effective way of protecting the user from XSS attack.

On the user/client side, the most effective solution is to disable all scripting language support in user’s browsers and e-mail readers[11]. If this is not a feasible option for business reasons, another recommendation is to use reasonable caution when clicking links in anonymous e-mails and dubious web pages. Also, keeping up to date with the latest browser patches and versions is important in protecting against other vulnerabilities which may expose. But usually, neither do users willing to disable all scripting language support, nor do they keen to keep their browsers up to date let alone how many of them are aware of the dangerous XSS.

### Table 1. Escape encoding for special characters

<table>
<thead>
<tr>
<th>Special Tags</th>
<th>Special Tags after Encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>&amp;</td>
<td>&amp;</td>
</tr>
<tr>
<td>&lt;</td>
<td>&lt;</td>
</tr>
<tr>
<td>&gt;</td>
<td>&gt;</td>
</tr>
<tr>
<td>&quot;</td>
<td>&quot;</td>
</tr>
<tr>
<td>’</td>
<td>'</td>
</tr>
</tbody>
</table>

### Table 2. Escape encoding

<table>
<thead>
<tr>
<th>Special Tags</th>
<th>Escape Encoding</th>
<th>Escape Encoding Again</th>
<th>In Browser</th>
</tr>
</thead>
<tbody>
<tr>
<td>&amp;</td>
<td>&amp;</td>
<td>&amp;</td>
<td>&amp;</td>
</tr>
<tr>
<td>&lt;</td>
<td>&lt;</td>
<td>&amp;lt;</td>
<td>&lt;</td>
</tr>
<tr>
<td>&gt;</td>
<td>&gt;</td>
<td>&amp;gt;</td>
<td>&gt;</td>
</tr>
</tbody>
</table>

Figure 3. A link containing XSS attempt

Figure 4. XSS current solutions

Figure 5. Comparison between scripts before escape encoding (a) and after escape encoding (b)

3 The Design of the Automatic Collection / Detection System for XSS Vulnerability

The system consists of a detection/collection proxy server and a database server. In the detection proxy server two modes are used to detect and collect the XSS attack information. The request change mode and the response change mode.

3.1 Response Change Mode

Once the users browse the web, the HTTP request(e.g., GET, POST) messages are captured and checked in the de-
tection/collection proxy server; if any characters in the request message match the HTML special tags, the requests are copied before sending to the requested web sites. Consequently, if the related response messages contain the same special language tags, the requested websites are considered as XSS vulnerable. Moreover, if the response messages contain special tags or malicious scripts, the proxy encodes the language tags and forwards the safe response message to the client. Meanwhile, the HTML alert message is inserted to response page for notification. This is called response change mode.

However, it does not work properly if the request and response messages contain multiple parameters with harmless HTML tags like `< or `<html>` embedded. It may be possible that the other parameters are included the special dangerous tags. In this circumstances, the response change mode can’t detect which parameters has XSS script tags and which ones don’t. By default, the proxy just assumes that any parameter with length longer than 10 characters should contain XSS scripts, those are not considered clear[11].

3.2 Request Change Mode

Due to the limited functionality of the response change mode, we propose another method called request change mode to handle the multi-parameter pitfall of response change mode.

As the Table 3 shows, in request change mode, when the system investigates the multi-parameter HTTP request message, it generates a random number – which will be used as an identifier or identity for parameters – and inserts the number just after the special characters (script language tags) in the first parameter; when it comes to the second parameter, the number is increased by one and inserted just after the special characters and similarly, the same goes to other parameters. Through the returning response, the system not only detects the XSS scripts but also identifies which parameters have XSS scripts such as, in this case, parameters 1 and 2. Thus, it avoids the false alert in the parameter 3 and 4.

Obviously, in request change mode, the system generates an extra request in addition to the original HTTP request. That says, the system sends the HTTP request with the `identity(random numbers)` for detection purpose before sending the original HTTP request. Thus, both the request and response are proceeding two times at target Web site. This may cause some problems with its aggressive probing on Web sites and the extra traffic it generates on the network.

### Table 3. Request Change Mode

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Before</th>
<th>After</th>
</tr>
</thead>
<tbody>
<tr>
<td>parameter1</td>
<td><code>&lt;s&gt;</code>test<code>&lt;s&gt;</code></td>
<td><code>&lt;234&gt;s&gt;</code>test <code>&lt;234/s&gt;</code></td>
</tr>
<tr>
<td>parameter2</td>
<td><code>&lt;s&gt;</code>test<code>&lt;s&gt;</code></td>
<td><code>&lt;235&gt;s&gt;</code>test <code>&lt;235/s&gt;</code></td>
</tr>
<tr>
<td>date</td>
<td><code>234</code></td>
<td><code>235</code></td>
</tr>
<tr>
<td>html</td>
<td><code>&lt;html&gt;</code></td>
<td><code>&lt;html&gt;</code></td>
</tr>
</tbody>
</table>

### Table 4. Information collected in database

<table>
<thead>
<tr>
<th>Element</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>hostName</td>
<td>XSS Vulnerable Host</td>
</tr>
<tr>
<td>pathName</td>
<td>XSS Vulnerable Path</td>
</tr>
<tr>
<td>request</td>
<td>XSS Vulnerable Request</td>
</tr>
<tr>
<td>parameterName</td>
<td>Parameter Name</td>
</tr>
<tr>
<td>parameter</td>
<td>Parameter contains XSS Script</td>
</tr>
<tr>
<td>date</td>
<td>The Time When HTTP Response is accepted</td>
</tr>
<tr>
<td>cookieScript</td>
<td>Whether the Cookie includes the Script or not</td>
</tr>
<tr>
<td>deleteFlag</td>
<td>Whether the Vulnerability is Corrected or Not</td>
</tr>
<tr>
<td>accessTime</td>
<td>The Time the Database is Updated</td>
</tr>
</tbody>
</table>

3.3 The Information Collection for XSS Vulnerability

Fig. 6 presents the system overview of the Automatic Detection/Collection system for XSS vulnerability. After the proxy server detects vulnerabilities, it sends those collected information such as host names, the parameter name, the path name etc. (Table 4) to the collection database server and such that the collected information can be shared among the proxy servers.

As illustrated in Fig. 7, the XML format is used to communicate between the proxy server and database. `addData`, `removeData`, `getList` methods are used to add data to database and upgrade the data, as well as share the data in the database respectively(as shown in Fig. 8).

4 Implementation

We have implemented our system using Java (J2SDK 1.4.1) in Redhat Linux 8.0 and the open source PostgreSQL7.2 is used as the collection database. The development of our detection/collection proxy server is an extension of the proxy server which is provided by Dr. Hiromitsu Takagi at AIST(National Institute of Advanced Industrial Science and Technology)[10]. As the detection part, two different detection mechanisms, response change mode and request change mode, are implemented. These will be discussed later.

4.1 Response Change Mode

Fig. 9 shows how the response change mode works.

1. Request Check
Figure 6. Automatic detection/collection system for XSS

The proxy checks whether its parameters include special characters. If there are, the detection/collection system will save a copy of the request in the proxy side and forward the original request. Otherwise the system just forward the request or response between the clients and servers.

2. Response Check

Followed by sending the request, the server generates its response. If the request is detected of containing the special characters, the detection/collection proxy compares the response message with the corresponding request message stored in the proxy server to see whether the same special characters are still included in the response message. If no special characters are found, the detection/collection proxy servers simply forward the response to the client. Otherwise, the system marks the server as XSS vulnerable and send the alert messages to the client. Meanwhile, the escape encoded response message will be sent to the client.

4.2 Request Change Mode

Fig. 10 illustrates a series of steps taken to accomplish the detection and collection procedures in request change mode. Every step is explained below.

1. Request Check
   Check whether the request message containing special characters.

2. Sending Dummy Request
   If the request message contains special character, the detection/collection server will save the copy of original request message and then to differ parameters in request message, random generated numbers are inserted to every parameter for indentification purpose before sending to the requested web server.

3. Dummy Response Check
   At this stage, the system investigates the server generated response message to see whether the Web server is XSS vulnerable. If the Web server is found vulnerable, the information about the Web server will be send to the database.
4. Sending the Request
If the web server is XSS vulnerable, the special characters in original request are escape encoded before sending to the web server. Otherwise, the detection/collection system simply forwards the original request to the server.

5. Response Check
Alert the user by embedding the alert HTML message in the response page.

4.3 XSS Collection Database Server
Corresponding to the queries from the detection/collection proxy server, three types of methods are used.

- **addData** adds the XSS vulnerability information with the timestamp to the database.
- **removeData** updates the database with renewed vulnerability information.
- **getList** updates the XSS vulnerable web site list information in detection/collection server from the the database.

### Table 5. Result for response change mode

<table>
<thead>
<tr>
<th>Parameters</th>
<th>GET</th>
<th>Collection</th>
<th>Encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>o</td>
<td>o</td>
<td>o</td>
</tr>
<tr>
<td>2</td>
<td>x</td>
<td>x</td>
<td>o</td>
</tr>
</tbody>
</table>

5 System Evaluation

We tested the system by using realistic examples. By realistic, we mean that the XSS vulnerable web applications for evaluating our system are realistically existing in the Internet. We gathered a number of XSS vulnerable websites[12] and manually inserted test scripts to the HTTP requests before sending to the vulnerable websites. The manually inserted test scripts are only their most simple type of XSS attack-scripts for demonstration purpose; more complex and complicated scripts (JavaScript, VBScript, etc.) exist, but it is out of our topic, besides XSS vulnerabilities exist in web applications not in scripts. Actually the HTML tags such as “<”, “>”, “<SCRIPT>”, “%3C” etc, are playing a major role in detecting XSS attacks; the scripts which is encapsulated by those HTML tags are not important.

The system requirement is to detect the malicious scripts embedded in the request/response message and the XSS vulnerabilities at the website and collecting XSS vulnerabilities information, as well as protecting the user/client from XSS attacks by encoding those special characters in the request/response messages.

Table 5 shows the result in the response change mode and Table 6 presents the result for the request change mode. ‘o’ means the system requirement is fulfilled while ‘x’ represents, only in some special cases, the requirement is not fulfilled. While investigating the request messages with multiple parameters the system failed to correctly detect the XSS vulnerability. As we pointed out at section 3, this can be solved by using the request mode. Table 6 shows that the request mode is very effective when encountered the multiple parameters problem.

At Table 7, we categorised our collected sites into 5 areas and listed the result for both the request change mode and response change mode. We constructed multi-parameter URLs with scripts when testing the system on request change mode. The result shows that the response change mode can effectively detect the vulnerability but when encountering multi-parameters, request change mode effectively detects which parameters contain malicious scripts.
Table 6. Result for request change mode

<table>
<thead>
<tr>
<th>Parameters</th>
<th>GET</th>
<th>POST</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Detection</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Collection</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Encoding</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 7. Evaluation of our system using realistic links

<table>
<thead>
<tr>
<th>Website Category</th>
<th>Total</th>
<th>Response Change</th>
<th>Request Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>News Media</td>
<td>3</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>Government Agencies</td>
<td>9</td>
<td>9</td>
<td>0</td>
</tr>
<tr>
<td>Online Stores</td>
<td>6</td>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>Search Engines</td>
<td>4</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>Technology</td>
<td>4</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>Miscellaneous</td>
<td>4</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>Total</td>
<td>32</td>
<td>32</td>
<td>0</td>
</tr>
</tbody>
</table>

6 Conclusion and Future Work

Our approach is an effective way to detect and collect XSS vulnerabilities. However, there are still many challenges to be addressed, especially, how to utilize the collected XSS informations in the central database, and how to make the system deployment universal. In the paper, we have presented a user-side proxy approach for automatically detecting and collecting Cross-Site Scripting Vulnerability. Two different detection modes, the response change mode and the request change mode, are discussed and evaluated with real-world examples respectively.

The evaluation showed that many “famous” sites are not secure against XSS vulnerabilities. The proposed approach and techniques described in this paper is useful in identifying web application security problems. Migrating the security responsibility from the server side to the client side have the advantage of a high-performance, dedicated XSS vulnerable detection and collection system at the client-site no matter whether the web servers are vulnerable or not.

Concretely, our future work is focussed on making the system more transparent between the client and the Internet; instead of manually inserting scripts and script tags for detection, the system will work without any intervention. Our another concern is how to verify and, most importantly, utilize the information collected at the central database. It is not guarantee that the collected data from detecting/collection proxies are 100% correct; false alert, human inference, and transmission interception may result in severe consequences.

Acknowledgment

The authors would like to thank Dr. Hiromitsu TAKAGI from AIST(National Institute of Advanced Industrial Science and Technology) for his many constructive advices and great help. The authors also wish to thank Mr. Yuuki TAKAHASHI for his strong support.

References


