第 XVIII 部

Cross-Site Scripting 脆弱性への対策手法

第 18 部
Cross-Site Scripting 脆弱性への対策手法

第 1 章 Introduction

Cross-Site Scripting (XSS) is caused by the failure of web applications to properly validate user input before returning it to the client's web browser. Although some approaches exist for defending against XSS attacks, XSS vulnerabilities continue to appear in web applications. These weaknesses, which often result from poorly developed web applications and data processing systems, allows attackers embedding malicious HTML-based contents, such as JavaScript, within client HTTP requests. Through embedding HTML code and scripting elements, it is possible to steal session ID information, thus, resulting in the leakage of privacy information.

The purpose of XSS-WG (Cross-Site Scripting Working Group) is to not only propose new approaches to detect and protect users from XSS attacks and vulnerabilities, but also implement and evaluate these approaches.

The output of the working group is an effective system that protects the users from XSS attacks and also enable detection of XSS vulnerable web applications. Currently, we propose a system that automatically detects XSS vulnerability by manipulating either request or server response at client side. The system also shares the collected vulnerability information via a central repository. Our approach is quite different from other works in the literature that only stress protection. In our case, the proposed system not only protects users from XSSattacks, but also detects the web servers with XSS vulnerabilities.

We evaluated the system in the WIDE camp. During the whole experiment period, the system investigated more than 8000 HTTP requests (HTTP GET requests) and the corresponding response messages. Through the detection mechanism, the system detected 13 XSS vulnerable Web sites (note that those aren't the XSS attacks, but the vulnerable Web sites that could be used to launch the XSS attacks).

We also evaluate the effectiveness of our system by probing the most representative, real-world XSS vulnerable web sites which we collected from the Internet. The result shows that, out of 32 XSS vulnerable web sites, our system successfully detected 30 Web sites. This indicates that our system's effectiveness against XSS attacks.

The rest of the report is organized as follows. In chapter 2 we explain the background, and in chapter 3 we briefly introduce XSS vulnerability and the related works. In chapter 4 we present our proposed system in detail. In chapter 5 we discuss the implementation of our proposed system. In chapter 6 we explain how we evaluate the system. Finally, chapter 7 concludes this report and briefly discuss our future work[29].

第 2 章 Background

For most major Internet financial institutions and retailers, the Internet provides both a cost-effective means of presenting their merchandise to customers, and a method of delivering personalized 24 hours a day, 7 days presense a week. In almost all cases, the preferred method of delivering these services is the HTTP protocol. Due to limitations within the protocol, there is no built-in facility to identify or track a particular customer (or session) uniquely within an application. In other words, HTTP is a stateless protocol. In

order to take 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[228]. 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 clear-text 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[224]. 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 and cause tremendous configuration overhead.

第 3 章　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.

3.1 Cross-Site Scripting Vulnerability

On February 20, 2000, CERT published information on newly identified security vulnerability affecting all web server products[27]. This vulnerability, known as Cross-Site Scripting (XSS), occurs 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.

3.2 XSS Attack and Cookie Stealing

As illustrated in Fig. 3.1, 'a' is a user, 'b' is a web page which contains the link (the link is shown in Fig. 3.3) of attackers' malicious scripts while the 'c' represents a trusted but a 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, <SCRIPT> and </SCRIPT> indicate the beginning and ending of the JavaScript code respectively.

For example, if the script in Fig. 3.2 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

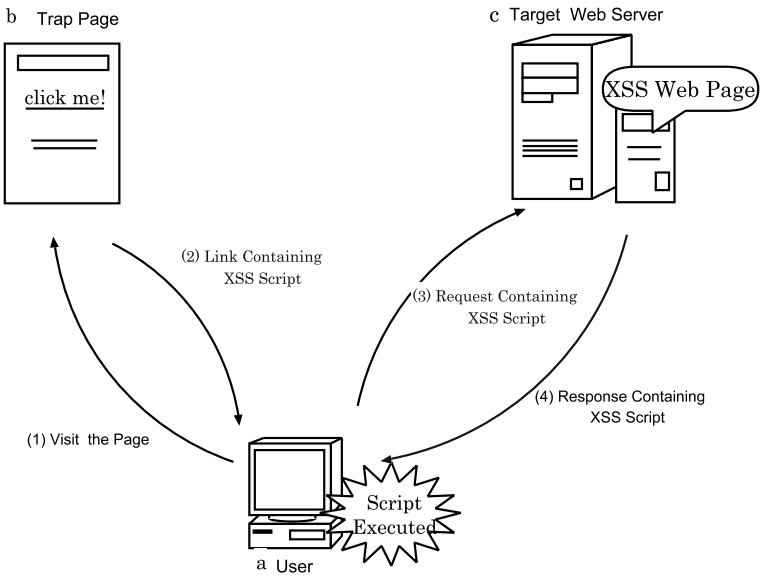

Fig. 3.1. The Principle of the XSS

```
<SCRIPT>document.write(document.cookie)</SCRIPT>
```

Fig. 3.2. Script typed into the search page

```
http://www.example.com/search.cgi?keyword =
<SCRIPT>document.write(document.cookie)</SCRIPT>
```

Fig. 3.3. A Link Containing XSS Attempt

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 insert into the web page, thereby tricking the user's browsers into running malicious scripts and codes.

3.3 Related Works

Various solutions are available to protect clients from XSS attacks. At first, the web application developers and vendors should ensure that all inputs from users are parsed and filtered properly. User input includes things stored in GET query strings, POST data, Cookies, URLs, and in general any persistent data that is transmitted between the browser and web server. The best philosophy to follow regarding user input filtering is to deny all but a preselected element of benign characters in the web input stream. This prevents developers from having to constantly predict and update all forms of malicious input in order to deny only specific characters (such as <, ;, > etc.)4),5). Additionally web application vulnerability scanners can be used to assist web developers and venders to test the vulnerabilities in their web applications. Once an application has evolved out of the design and development phases, it is important to periodically test for XSS vulnerabilities since application functionality is constantly changing due to upgrades, integration of third party technologies, and decentralized authoring website. Due to the operational overhead and dynamic features of web applications from different vendors and developers, it is very uncertain to say whether their products are immune for XSS attacks or not.

Various solutions are available to protect clients from XSS attacks. 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. 3.4. 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.

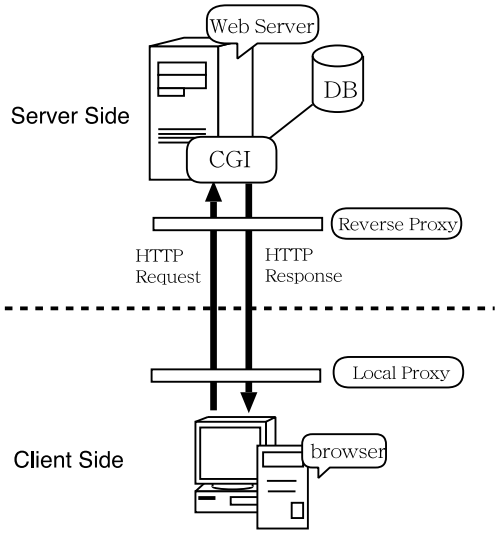

Fig. 3.4. Two Approaches to Address XSS Vulnerability

Scott and Sharp[257] 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 structure with little documents.

JWIG[290] project also 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[104], OWASP[291] 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.

```
(a) <SCRIPT>document.write(document.cookie)</SCRIPT>
(b) <SCRIPT>&lt;SCRIPT&gt;document.write(document.cookie)
    &lt;/SCRIPT&gt;
```

Fig. 3.5. Comparison between scripts before escape encoding (a) and after escape encoding (b)

Table 3.1. Escape Encoding for Special Characters

Special Tags	Special Tags after Encoding
&	&amp;
<	&lt;
>	&gt;
“	&quot;
’	&#39;

Table 3.2. Escape Encoding

Special Tags	Escape Encoding	Escape Encoding Again	In Browser
&	&amp;	&	&amp;
<	&lt;	<	&lt;
>	&gt;	>	&gt;

On the user/client side, the most effective solution is to disable all scripting language support in user's browsers and e-mail readers[223]. 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.

第 4 章 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.

4.1 Response Change Mode

Once the users browse the web, the HTTP request (e.g., GET, POST) messages are captured and checked in the detection/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 these circumstances, the response change mode can't detect which parameter has XSS script tags and which one hasn't. In default, the proxy just assumes that any parameter with length longer than 10 characters should contain XSS scripts, those are not considered clear[223].

4.2 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 4.1 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

Table 4.1. Request Change Mode

Parameter Name	Before	After
parameter1	<s>test</s>	<234s>234test<234/s>234
parameter2	<s>test</s>	<235s>235test<235/s>235
parameter3	<	<236
parameter4	<html>	<237html>237

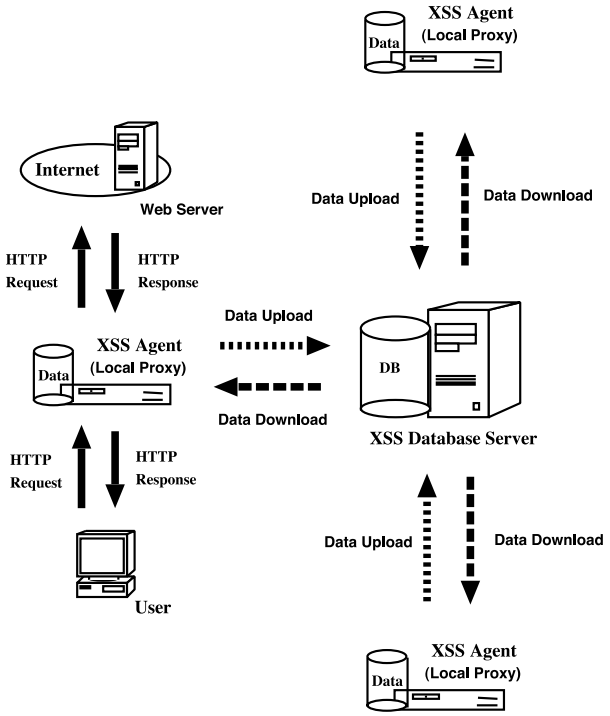

Fig. 4.1. Automatic Detection/Collection System for XSS

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 be able to identify which parameters have XSS scripts such as, in this case, Parameter 1 and 2. Thus, it avoids the false alert in the parameter 3, 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 *indentity* (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.

4.3 The Information Collection for XSS Vulnerability

Fig. 4.1 presents the system overview of the Automatic Detection/Collection system for XSS vulnerability. After the proxy server detects the vulnerabilities, it sends those collected information such as host names, the parameter name, the path name etc. (Table. 4.2) to the collection database server and such that the collected information can be shared among the proxy servers.

As illustrated in Fig. 4.2, 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

Table 4.2. Information Collected in Database

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

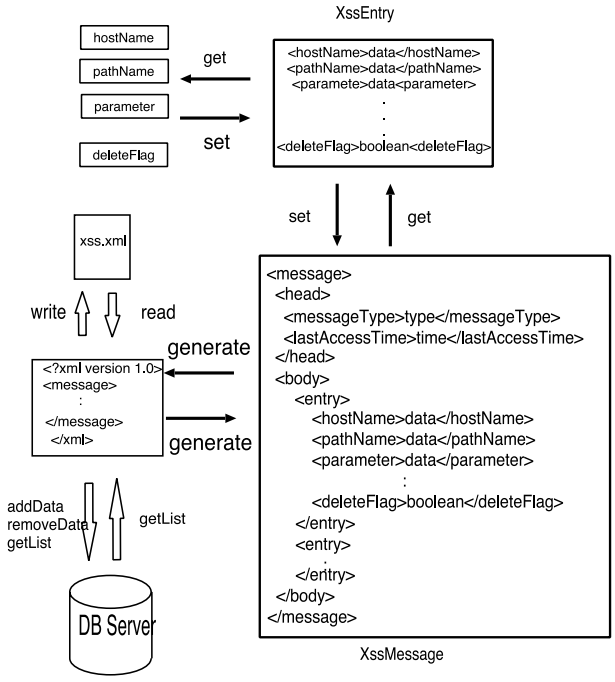

Fig. 4.2. Data Format for XSS Vulnerabilities

upgrade the data, as well as share the data in the database respectively.

第 5 章 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)[102]. As the detection part, two different detection mechanisms, response change mode and request change mode, are implemented. These will be discussed later.

5.1 Response Change Mode

Fig. 5.1 shows how the response change mode works.

1. Request Check

   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 forwards the request or response between

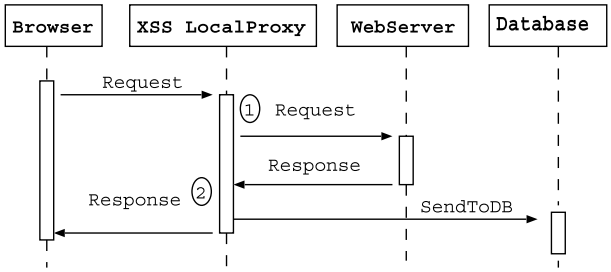

Fig. 5.1. Response Change Mode

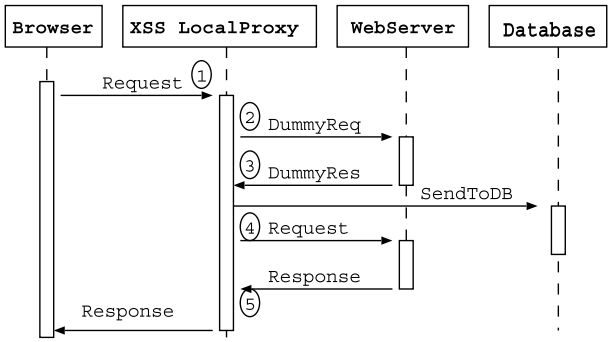

Fig. 5.2. Request Change Mode

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 sends the alert messages to the client. Meanwhile, the escape encoded response message will be sent to the client.

5.2 Request Change Mode

Fig. 5.2 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

   Same with the description at response check mode. To 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 numbers are inserted to every parameter in random number plus one order 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 server XSS vulnerable. Then send the result 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 the alert message by embedding HTML in the response page.

5.3 XSS Collection Database Server

- addData

  Add the XSS vulnerability information with the timestamp to the database.
- removeData

  Update the database with renewed vulnerability information.
- getList

  Update the XSS vulnerable web site list information from the database in detection/collection server.

第 6 章 System Evaluation

We measure the performance of our proposed approach on the number of detected XSS vulnerable Web sites. We evaluated this system at the WIDE Camp (Sept, 2003). During the whole experiment period, the system investigated more than 8000 HTTP requests (HTTP GET requests) and the corresponding response messages. Through the detection mechanism, the system detected XSS vulnerable 13 Web sites (note that those aren't the XSS attacks, but the vulnerable Web sites that could be used to launch the XSS attacks).

We also tested the system by using real-world examples. By real-world, we mean that the XSS vulnerable web applications for evaluating our system are actual web sites, not emulated ones. We gathered a number of XSS vulnerable websites[230] and manually inserted emulated scripts to the HTTP requests before sending to the vulnerable websites. The emulated 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 6.1 shows the result in the response change mode and Table 6.2 presents the result for the request change mode. In order to test the XSS detection/collection system, we gathered some XSS vulnerable websites and manually insert emulated scripts to the HTTP requests before sending to the requested website. The system requirement is to detect the malicious scripts embedded in the request/response message and the XSS vulnerabilities at the website and to collect XSS vulnerabilities information, as well as to protect the user/client from XSS attacks by encoding those special characters in the request/response messages. Table 6.1 shows the result in the response change mode and Table 6.2 presents the result for the request change mode. 'o' means the system requirement is fulfilled while '×' 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 chapter 4, this can be solved by using the request change mode. Table 6.2 shows that the request change mode is very effective when encountered with the multiple parameters problem.

At Table 6.3, we categorized 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

Table 6.1. Result for Response Change Mode

	Number of Parameters	Detection	Collection	Encoding
GET	1	○	○	○
	2	×	×	○
POST	1	○	○	○
	2	×	×	○

Table 6.2. Result for Request Change Mode

	Number of Parameters	Detection	Collection	Encoding
GET	1	○	○	○
	2	○	○	○
POST	1	○	○	○
	2	○	○	○

Table 6.3. Evaluation of Our System Using Real-world Links

Website Category	Total	Response Change		Request Change	
		○	×	○	×
News Media					
Government Agencies	3	3	0	3	0
Online Stores	9	9	0	8	1
Search Engines	3	3	0	3	0
Technology	6	6	0	6	0
Miscellaneous	4	4	0	3	1
Total	32	32	0	30	2

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.

第 7 章　Conclusion and Future Work

From the participants' point of view, the system protected them against XSS attacks but it failed to provide a completely stable Web environment. For example, the system generated some false alerts, and as a result, the corresponding responses were blocked. Also, since every request is required to be checked, the participants felt that the system violated their privacy. Deploying the system directly in user's computer by using local proxy might avoid those problems but this approach needs to be tested.

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 information in the central database, and how to make the system deployment for *universal*. In the report, we have presented a user side forwarding 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 vulnerablilities. The proposed approach and techniques described in this report is useful in indentifying web application security problems. The Migrating 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. Meanwhile, this will reduce the configuration overhead both in administration side and client side. Those real-world examples (Table 6.3) show us the exciting future of this approach and justifying those claims to real-life case studies is high priority for future work.