Analysis of Prevention of XSS Attacks at Client Side

Teena Hadpawat
Department of CSE
CTAE, Udaipur

Dipesh Vaya
Department of CSE
SSCE, Udaipur

ABSTRACT
The web has become paramount part of our lives. Unfortunately, as our dependency on the web increases, so does the interest of attackers in enslaving web applications and web-base information systems. Previous work in the field of web application security has mainly focused on the mitigation of Cross Site Scripting and SQL injection attacks. XSS, or Cross Site Scripting, allows an attacker to execute code on the target website from user's browser of ten causing side effects such as data compromise, or the stealing of a user session. This can allow an attacker to impersonate a user to steal their details, or act in their place without consent. It is caused by scripts, which do not sanitize user input. In general, XSS attack is easy to execute, but difficult to detect and prevent. It can be prevented at both client and server. Several server side solutions of XSS attacks do exist, but such techniques have not been universally applied, because of their deployment overhead. In this paper analyzing of client side solution to detect attack and which technique is appropriate is done. In this paper focus is on the analysis of most of the client side solution presented yet and provides a comparative view of the solutions.

Keywords
XSS attacks, SQL injection, Client side solution

1. INTRODUCTION
The rapid growth of internet resulted in feature rich, dynamic web applications. This increase resulted in the harmful impact of security flaws in such applications. Vulnerabilities leading to compromise of sensitive information are being reported continuously, resulting in ever increasing financial damages. Notably Facebook, MySpace and Orkut have all been hit by these attacks. XSS attacks can be self-propagating [1], and have the potential to rapidly victimize millions of people. The JavaScript language [2] is widely used to enhance the client-side display of web pages. Usually, JavaScript code is downloaded into browsers and executed on-the-fly by an embedded interpreter. In this scenario, the attacker sends a specially crafted e-mail message to a victim containing malicious link scripting such as one shown below:

<A
HREF=http://educane.com/registration.cgi?clientprofile=<SC
RIPT>malicious code</SCRIPT>>Click here</A>

When an unsuspecting user clicks on this link, the URL is sent to educane.com including the malicious code. If the legitimate server sends a page back to the user including the value of client profile, the malicious code will be executed on the client web browser, comparative view of two proposed client side techniques, client side optimization and noxes[5] is shown in Fig 1.

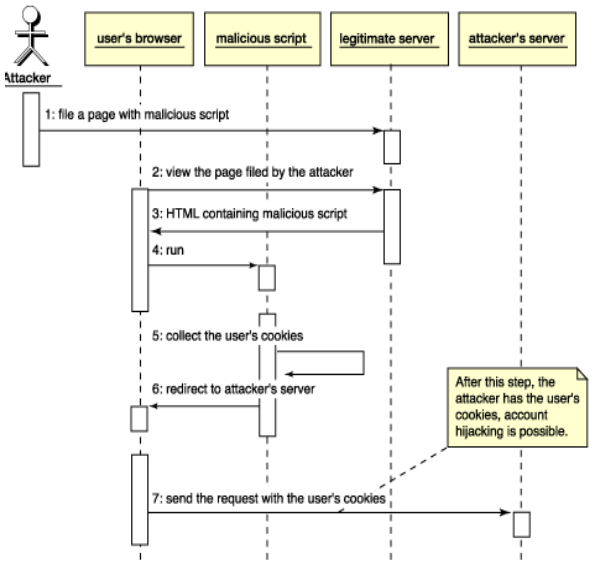

Fig 1: Email Scenario of Cross Site Scripting

2. TYPES OF XSS ATTACKS

2.1 The persistent (or stored)
It is a more devastating variant of a cross-site scripting flaw, it occurs when the data provided by the attacker is saved by the server, and then permanently display on "normal" pages returned to other users in the course of regular browsing, without proper HTML escaping. A classic example of this is with online message boards where users are allowed to post HTML formatted message for other users to read.

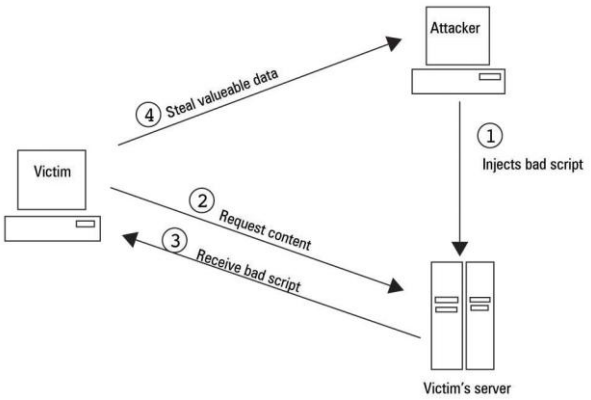

Figure 2: Persistent Attack

2.2 Non persistent
The non-persistent (or reflected) cross-site scripting vulnerability is by far the most common type. These holes show up when the data provided by a web client, most commonly in HTTP query parameters or in HTML form submissions, is used immediately by server-side scripts to generate a page of results for that user, without properly

sanitizing the request.

```
<?php
if(!array_key_exists("name",$_GET) | |$_GET['name'] ==
NULL || $_GET['name']==""){
$isempty=true;
} else{
echo '<pre>';
echo 'Hello' . $_GET['name'];
echo '</pre>';
}?>
```

AS it can be seen that the “name” parameter doesn't sanitized and echo back to the user, so when the user inject a malicious JS code, It will execute. Now attacker will inject its malicious js Code, for demonstration <script>alert (/xss/) </script> is injected.

2.3 DOM based

DOM-based vulnerabilities occur in the content processing stages performed by the client, mostly in client-side JavaScript. The name refers to the standard model for representing HTML or XML contents which is called the Document Object Model (DOM) JavaScript programs manipulate the state of web page and populate it with dynamically-computed data primarily by acting upon the DOM. This type occurs on the java script code itself that the developer use in client side for example "A typical example is a piece of JavaScript accessing and extracting data from the URL via the location.* DOM, or receiving raw non-HTML data from the server via XML HttpRequest, and then using this information to write dynamic HTML without proper escaping, entirely on client side.

...

Select your language:

```
<select><script>
document.write("<OPTION
value=1>"+document.location.href.substring(document.
location.href.indexOf("default=")+8)+"</OPTION>");
document.write("<OPTION
value=2>English</OPTION>");
 </script></select>
```

...

If the page is loaded with the 'default' parameter set to '<script>alert(“xss”)</script>' instead of the intended language string, then the extra script will be added into the page's DOM and executed as the page is loaded.

3. RELATED WORK

3.1 Personal Firewall

Personal firewall is subject to its ability to effectively control and block incoming and outgoing traffic. In addition software firewalls operate through a learning process in which program and processes may be allows and denied access to the internet. If a new process executed on the selected workstation which requires internet access, the firewall should continually block access until the end user clearly permits the traffic to flow. Unfortunately software firewalls do not clearly and legibly present information on program and processes which are attempting to access the internet. As a result numerous end user tend to remove and uninstall the firewall as the question the firewall presents are often not aimed at an individual with little or no expertise in computer and network security.

3.2 Blacklisting vs. Whitelisting

To help mitigate XSS attacks, two basic techniques are used to sanitize data. Blacklisting uses a list of known bad data to block illegal content from being executed. Whitelisting uses a list of known good data to allow only that content to be executed. Blacklisting mode is faster to set up, but can be bypassed more easily by a skilled attacker. Whitelisting allows for a much stronger security solution but comes with a steep learning curve. Once mastered, though, whitelisting is very effective at stopping XSS attacks. OWASP [7] example of white testing OWASP Enterprise Security API.

3.3 Swap

SWAP operates on a reverse proxy, which relays all traffic between the web server that should be protected and its server visitors. The proxy forwards each web response, before sending it back to the client browser, to a JavaScript detection component, in order to identify embedded JavaScript content. In the JavaScript detection component, SWAP puts to work a fully functional, modified Web browser that notifies the proxy of whether any scripts are contained in the inspected content. If no scripts are found, the proxy decodes all script IDs, effectively restoring all legitimate scripts, and delivers the response to the client. If the JavaScript detection component, on the other hand, detects a script, SWAP refrains from delivering the response, but instead notifies the client of the attempted XSS attack. Solutions on server side result in considerable degradation of web application and are often unreliable, whereas the client side solutions result in a poor web browsing experience, there is need of an efficient client side solution which does not degrade the performance.

4. COMPARISON

Firstly, optimized client side solution is based on modified client side web browser, this solution was implemented using open source Mozilla Firefox 1.5 web browser from Mozilla foundation. The modified web browser was successfully built with the help of the build documentation provided on Mozilla website on Microsoft’s Windows XP using Visual Studio .Net 2003 and Cygwin, where as noxes [5] is based on personal web firewall application which runs on the background service of user’s desktop. The development of Noxes [5] was inspired by Windows personal firewalls that are widely used on PCs and notebooks today. Popular examples of such firewalls are Tiny, ZoneAlarm, Kerio and Norton Personal Firewall through Noxes [5] and can either be blocked or allowed based on the current security policy in other components of the web browser. Some Data structures were created, and others were modified according to the need where as in Noxes [5] operates as a web proxy that fetches HTTP requests on behalf of the user’s browser. Hence, all web connections of the browser pass through Noxes [5] and can either be blocked or allowed based on the current security policy

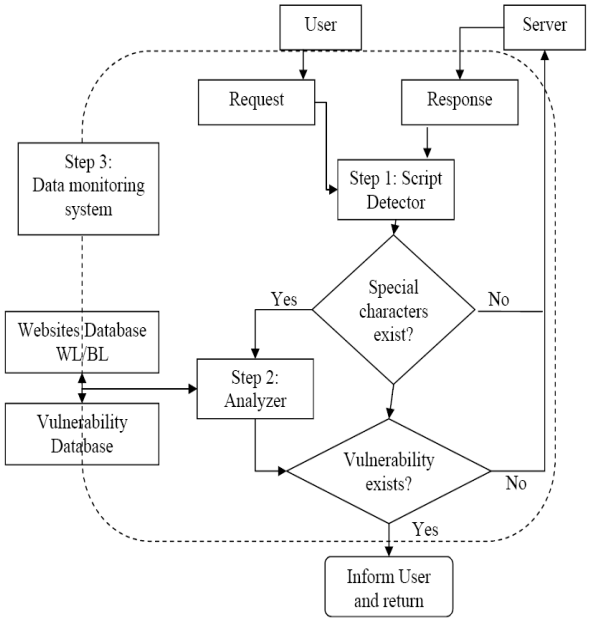

Fig 3. Proposed Solution: a three step process to detect XSS

Third, client side fig. 3, checks for the maximum number of characters, and if the input exceeds the number of characters, then the input is rejected without processing the input further. The second condition is checked by the analyzer is the existence of special characters. This is because the scripts can only be executed when it is embedded using the tags and special characters. If special character exists in the input, then the input is passed to the parser, otherwise the request is forwarded to the web application. It is not enough to use a blacklist of special characters to detect XSS in input or to encode output. Searching for and replacing just a few characters or phrases is weak and has been attacked successfully. Even an unchecked "<b>" tag is unsafe in some contexts. XSS has a surprising number of variants that make it easy to bypass blacklist validation. A whitelist and a blacklist of sites is maintained and synchronized with server of security sites like CERT or other advisory, so that the decision can be taken on the basis of previous record of the website. Also a list of potentially vulnerable script tags is maintained as a small fast database. The second step is performed by an analyzer which uses both these databases to detect vulnerability, and decision is made by user. The third step is above the whole system, which is performed by a data monitoring system. The flow of data is passively monitored by the system. The operations processing sensitive information are marked along with the results of those operations. If the marked data is about to be transferred over the network, user is asked to al- low or disallow the transfer based on the information in the dialogue box provided. The information is in layman language, and it teaches user about the consequences, so that any kind of user can take a good decision. Where as in noxes [5] personal firewall that will have a set of filter rules that do not change over a long period of time, a personal web firewall has to deal with filter rule sets that are flexible; a result of the highly dynamic nature of the web. In a traditional firewall, a connection being opened to an unknown port by a previously unknown application is clearly a suspicious action. On the web, however, pages are linked to each other and it is perfectly normal for a web page to have links to web pages in domains that are unknown to the user. Hence, a personal web firewall that should be useful in practice must support some optimization to reduce the need to create rules. At the same time, the firewall has to ensure that security is not undermined. An important observation is that all links that are statically embedded in a web page can be considered safe with respect to XSS attacks. That is, the attacker cannot directly use static links to encode sensitive user data. The reason is that all static links are composed by the server before any malicious code at the client can be executed. An XSS attack, on the other side, can only succeed after the page has been completely retrieved by the browser and the script interpreter is invoked to execute malicious code on that page. In addition, all local links can implicitly be considered safe as well. An adversary, after all, cannot use a local link to transfer sensitive information to another domain; external links have to be used to leak information to other domains. Based on these observations, system is extended with the capability to analyze all web pages for embedded links. That is, every time Noxes [5] fetches a web page on behalf of the user, it analyzes the page and extracts all external links embedded in that page. Then, temporary rules are inserted into the firewall that allows the user to follow each of these external links once without being prompted. Because each statically embedded link can be followed without receiving a connection alert, the impact of Noxes [5] on the user is significantly reduced. Links that are extracted from the web page include HTML elements with the href and src attributes and the url identifier in Cascading Style Sheet (CSS) files. The filter rules are stored with a time stamp and if the rule is not used for a certain period of time, it is deleted from the list by a garbage collector. Using the previously described technique, all XSS attacks can be prevented in which a malicious script is used to dynamically encode sensitive information in a web request to the attacker's server. The reason is that there exists no temporary rule for this request because no corresponding static link is present in the web page. Note that the attacker could still initiate a denial-of-service (DOS) XSS attack that does not transfer any sensitive information. For example, the attack could simply force the browser window to close. Such denial-of-service attacks, however, are beyond the scope of our work as Noxes [5] solely focuses on the mitigation of the more subtle and dangerous class of XSS attacks that aim to steal information from the user. It is also possible to launch an XSS attack and inject HTML code instead an XSS attack and inject HTML code instead of JavaScript. Since such attacks pose no threat to cookies and session IDs, they are no issue for Noxes [5]. Figure 4 shows an example page. When this page is analyzed by Noxes [5], temporary rules are created for the URLs http://example.com/1.html (line 4), http://example2.com/-2.html (line 6) and http://external.com/image.jpg (line 8). The local links /index.html and /services.html (lines 11 and 12) are ignored

```
1. <html>
2. <body>
3. <h2>This is an example page.</h2>
4. <a href="http://example.com/1.html">
5. First link </a>
6. <a href="http://example2.com/2.html">
7. Second link </a>
8. <img src="http://external.com/image.jpg"
9. alt="Some image">
10. This is followed by a local link: <br>
11. <a href="/index.html">Home</a>
12. <a href="/services.html">Services</a>
13.
```

14. </body>
15. </html>

Fig 4. An example HTML page

When Noxes [5] receives a request to fetch a page, then it goes through several steps to decide if the request should be allowed. It first uses a simple technique to determine if a request for a resource is a local link. This is achieved by checking the Referrer HTTP header and comparing the domain in the header to the domain of the requested web page. Domain information is determined by splitting and parsing URLs. For example, the hosts client1.tucows.com and www.tucows.com will both be identified by Noxes [5] as being in the domain tucows.com. If the domains are found to be identical, the request is allowed. Although the referrer header is optional according to the HTTP specification, all popular browsers such as the Internet Explorer, Opera and Mozilla make use of this header. Note that using the Referer HTTP header is safe because the attacker has no means of spoofing or changing this header. The reason is that JavaScript does not allow the Referrer HTTP header to be modified (e.g. JavaScript error messages are generated in Internet Explorer, Mozilla and Opera).Also, the code that the attacker can inject only runs on the victim's browser and has no direct access to the network. If a request being fetched is not in the local domain, Noxes [5] then checks to see if there is a temporary filter rule for the request. If there is a temporary rule, request is allowed. If not, Noxes [5] checks its list of permanent rules to find a matching rule. If no rules are found matching the request, the user is prompted for action and can decide manually if the request should be allowed or blocked.

5. CONCLUSION

XSS vulnerabilities are being discovered and disclosed at an alarming rate. XSS attacks are generally simple, but difficult to prevent because of the high flexibility that HTML encoding schemes provide to the attacker for circumventing server-side input filters. In [3], the author describes an automated script-based XSS attack and predicts that semi automated techniques will eventually begin to emerge for targeting and hijacking web applications using XSS, thus eliminating the need for active human exploitation.

Large amount of websites are vulnerable to XSS attacks. The client side solution is found to be very effective. The solution is platform independent and has been implemented on a platform independent browser, so it can be used with other operating systems with a few changes. Cross site scripting vulnerability exists on all the platforms, so it is a big advantage over other solutions it uses a step by step approach instead of performing all the tests at the same time.

In noxes [5] is that it is the first client-side solution that provides XSS protection without relying on the web application providers. Noxes [5] supports an XSS mitigation mode that significantly reduces the number of connection alert prompts while at the same time providing protection against XSS attacks where the attackers may target sensitive information such as cookies and session IDs. Thanks to the experts who have contributed towards development of the template.

6. REFERENCES

[1] Kamkar, S. I'm popular, 2005, description and technical explanation of the JS. Spacehero (a.k.a. "Samy") MySpace worm.

[2] Flanagan,D. JavaScript: The Definitive Guide. December 2001. 4th Edition.

[3] Masri, W., Beirut, L., Podgurski, A. Using dynamic information flow analysis to detect attacks against applications, ACM SIGSOFT Software Engineering Notes Volume 30, Issue 4 July 2005

[4] Jovanovic, N, Kruegel, C., and Kirda., E. Pixy: A Static Analysis tool for Detecting web application vulnerabilities, Proceedings of the 2006 IEEE Symposium on Security and Privacy(S&P'06).

[5] Kirda, E., Kruegel, C., Vigna,G., and Jovanovic., N. Noxes: A Client-Side Solution for Mitigating Cross-Site Scripting Attacks. In The 21st ACM Symposium on Applied Computing (SAC 2006), Pages: 330 - 337, April 23-27, 2006.

[6] Ismaill,, O., and .Youki, M.E., A proposal and Implementation of Automatic Detection/Collection system for Cross-Site Scripting Vulnerability"=. Proceeding of the 18th International conference on Advanced Information Networking and Application (AINA'04).

[7] Kavado, Inc. "InterDo Version 3.0." Kavado Whitepaper, 2003.