Prevention Of Cross-Site Scripting Attacks (XSS) On Web Applications In The Client Side

***S.SHALINI*[1], *S.USHA*[2]**

[1]**Department of Computer and Communication, Sri Sairam Engineering College, Chennai- 44, Tamilnadu, India.**

[2]**Department of Information Technology, Sri Sairam Engineering College, Chennai- 44, Tamilnadu, India.**

Abstract* — Cross Site Scripting (XSS) Attacks are currently the most popular security problems in modern web applications. These Attacks make use of vulnerabilities in the code of web-applications, resulting in serious consequences, such as theft of cookies, passwords and other personal credentials. Cross-Site scripting (XSS) Attacks occur when accessing information in intermediate trusted sites. Client side solution acts as a web proxy to mitigate Cross Site Scripting Attacks which manually generated rules to mitigate Cross Site Scripting attempts. Client side solution effectively protects against information leakage from the user's environment. Cross Site Scripting (XSS) Attacks are easy to execute, but difficult to detect and prevent. This paper provides client-side solution to mitigate cross-site scripting Attacks. The existing client-side solutions degrade the performance of client's system resulting in a poor web surfing experience. In this project provides a client side solution that uses a step by step approach to protect cross site scripting, without degrading much the user's web browsing experience.

Keywords -- Cross Site Scripting; web proxy; Software Protection; Code Injection Attacks; Security Policies.

1. INTRODUCTION

Cross-Site Scripting, commonly known as XSS, is a type of attack that gathers malicious information about a u ser; typically in the form of a s pecially crafted hyperlink that will save the users credentials. Cross-site scripting, or XSS is a web security vulnerability where the attacker injects malicious client-side script into a web page. When a user visits a web page, the script code is downloaded and transparently run by the web browser. The malicious script inherits the user's rights, authentication, and so on. XSS represents the majority of web based security vulnerabilities

One reason for the popularity of XSS vulnerabilities is that developers of web-based applications often have little or no security background. The result is that poorly developed code, riddled with security flaws, is deployed and made accessible to the whole Internet. Currently, XSS attacks are dealt with by fixing the server-side vulnerability, which is usually the result of improper input validation routines.

XSS protection can be configured for multiple types of request and response data – URL query parameters – URL encoded input ("POST data") – HTTP headers – Cookies.

The possibilities to manipulate HTML documents displayed by the browser with JavaScript or to influence the operation of the browser itself are dangerous features if misused. The misuse potential directly relates to the functions available for a malicious programmer. Unfortunately JavaScript provides full access to HTML documents using the document object model (DOM). A script therefore can modify at least the document it is residing in arbitrarily: it is also possible to completely delete the document and create a t otally different document. From an attackers point of view two things are of special interest: cookies associated to a document and access credentials. JavaScript also provides access possibilities to these information. The cookies associated to a document can be accessed using the function call document.cookie and application level access credentials are often acquired using form based login. Here the credentials are input into input fields residing in a form environment. Since the form is part of the document a script can access all information in all fields or can simply modify the target URL of the form. Then the credentials are sent to the new target, which is under the control of the attacker. JavaScript is a powerful tool for developing rich Web applications. Without client-side execution of code embedded in HTML and XHTML pages, the dynamic nature of Web applications like Google Maps, Try Ruby! and Zoho Office would not be possible. Unfortunately, any time you add complexity to a s ystem, you increase the potential for security issues -- and adding JavaScript to a Web page is no exception.

Among the problems introduced by JavaScript are:

- A malicious Web site might employ JavaScript to make changes to the local system, such as copying or deleting files.
- A malicious Web site might employ JavaScript to monitor activity on the local system, such as with keystroke logging.
- A malicious Web site might employ JavaScript to interact with other Web sites the user has open in other browser windows or tabs.

The first and second problems in the above list can be mitigated by turning the browser into a sort of "sandbox" that limits the way JavaScript is allowed to behave so that it only works within the browser's little world. The third can be limited somewhat as well, but it is all too easy to get around that limitation because whether a p articular Web page can interact with another Web page in a given manner may not be something that can be controlled by the software employed by the end user. Sometimes, the ability of one Web site's JavaScript to steal data meant for another Web site can only be limited by the due diligence of the other Web site's developers.
The key to defining cross-site scripting is in the fact that vulnerabilities in a given Web site's use of dynamic Web design elements may give someone the opportunity to use JavaScript for security compromises. It's called "cross-site" because it involves interactions between two separate Web sites to achieve its goals. In many cases, however, even though the exploit involves the use of JavaScript, the Web site that's vulnerable to cross-site scripting exploits does not have to employ JavaScript itself at all. Only in the case of local cross-site scripting exploits does the vulnerability have to exist in JavaScript sent to the browser by a legitimate Web site.

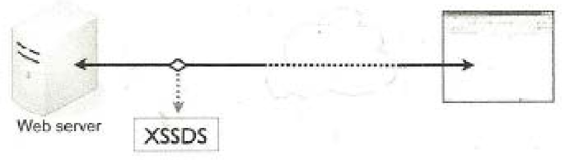


Fig 1: Overview of XSS Attack

2. TYPES OF CROSS-SITE SCRIPTING

To prevent the script code contained in a document loaded from some Web site accesses documents loaded from some other Web site, browsers do n ot allow access between documents loaded from different sites (i.e. cross-site access). Therefore attackers use other techniques to implement a cr oss-site attack. In general there are currently three major categories of cross-site scripting. Others may be discovered in the future, however, so don't think this sort of misuse of Web page vulnerability is necessarily limited to these three types.

- Reflected Cross-Site Scripting attacks

- Stored Cross-Site Scripting attacks

- DOM based Cross-Site Scripting attacks

• **Reflected XSS:** Probably the most common type of cross-site scripting exploit is the reflected exploit. It targets vulnerabilities that occur in some Web sites when data submitted by the client is immediately processed by the server to generate results that are then sent back to the browser on the client system. An exploit is successful if it can send code to the server that is included in the Web page results sent back to the browser, and when those results are sent the code is not encoded using HTML special character encoding thus being interpreted by the browser rather than being displayed as inert visible text.
The most common way to make use of this exploit probably involves a link using a malformed URL, such that a variable passed in a URL to be displayed on the page contains malicious code. Something as simple as another URL used by the server-side code to produce links on the page, or even a user's name to be included in the text page so that the user can be greeted by name, can become a vulnerability employed in a reflected cross-site scripting exploit.

• **Stored XSS:** Also known as HTML injection attacks, stored cross-site scripting exploits are those where some data sent to the server is stored (typically in a database) to be used in the creation of pages that will be served to other users later. This form of cross-site scripting exploit can affect any visitor to your Web site, if your site is subject to a stored cross-site scripting vulnerability. The classic example of this sort of vulnerability is content management software such as forums and bulletin boards where users are allowed to use raw HTML and XHTML to format their posts.
As with preventing reflected exploits, the key to securing your site against stored exploits is ensuring that all submitted data is translated to display entities before display so that it will not be interpreted by the browser as code.
.
• **DOM-based XSS:** It is a special variant of reflected XSS, where logic errors in legitimate JavaScript and careless usage of client-side data result in XSS conditions.
Application developers and owners need to understand DOM Based XSS, as it represents a threat to the web application, which has different preconditions. As such, there are many web applications on the Internet that are vulnerable to DOM Based XSS, yet when tested for standard XSS, are demonstrated to be "not vulnerable". Developers and site maintainers need to familiarize themselves with techniques to detect DOM Based XSS vulnerabilities, as well as with techniques to defend against them.

3. RELATED WORK

By now there have been a variety of defensive techniques to prevent XSS, including the following aspects: static analysis, dynamic analysis, black-box testing, white-box testing, anomaly detection, etc. Generally, these approaches are deployed on the client-side or server-side to protect web users from XSS injection attack. To remedy the shortcomings of server- side protection, there have been several defensive strategies which are deployed on the client side. In a client-side mechanism for detecting malicious JavaScript is proposed. The system consists of a browser-embedded script auditing component, and an IDS that processes the audit logs and compares them to signatures of known malicious behavior or attacks. With this system, it is possible to detect various kinds of malicious scripts, not only XSS attacks.
However, the system has significant weakness: it can only detect the XSS attacks whose behavior haven been known. Attacks that do not anticipated by the signature authors are left unprotected by the scheme.
The two main aims of XSS attacks are stealing the victim user's sensitive information and invoking malicious acts on the user's behalf. Noxes provides a client-side web proxy to block URL requests by malicious content using manual and automatic rules. Reference presents another approach: tracking the flow of sensitive information in the browser to prevent malicious content from leaking such information. Both of these projects focus on ensuring confidentiality of sensitive data (e.g., cookies) by analyzing the flow of data through the browser, rather than preventing unauthorized script execution. They can defeat only the first goal of XSS attacks. It would be defeated by attacks that do not violate same-origin policies. By contrast, our approach is based on analyzing function-call sequences of JavaScript to detect unauthorized scripts; we can defeat both objectives of XSS attacks.
One of the most important discussions related to aspects of code injection is by CERT. The paper describes the source of code injection: invalidated input from untrustworthy sources. It also proposes solutions that may be carried out directly by users. On the client side, the most effective solution is to disable all scripting language support in user's browsers and e-mail readers. 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. 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.
Scott and Sharp used an application proxy to abstract Web application protection; the proxy validates user input to protect against XSS attacks. Commercial products such as Appshield and InterDo use a similar approach. However, even though it provides immediate assurance of Web application security, it requires the correct identification of and validation policy for each individual entry point to a Web application. Another limitation is that this approach protects Web applications at the deployment phase instead of trying to eliminate bugs during the development phase. One thing should not be avoided when discussing the server side solution, the performance.
Most existing browsers are capable of interpreting and executing scripts created in such scripting languages as JavaScript, JScript, VBScript that are embedded in the Web-page downloads from the Web server. When an attacker introduces a malicious

script to a dynamic form submitted by the user, a cross-site scripting (XSS) attack then occurs.

4. PROPOSED MODEL

The 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. The proposed system is designed in order to provide effective security against the Cross Site Scripting attack, keeping the concept of usable security with optimized web browsing. This approach uses a three step process:

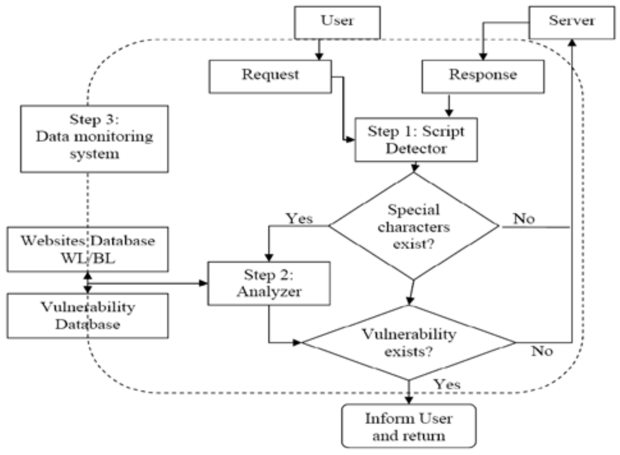


Fig 2: Proposed Solution: a three step process to detect XSS

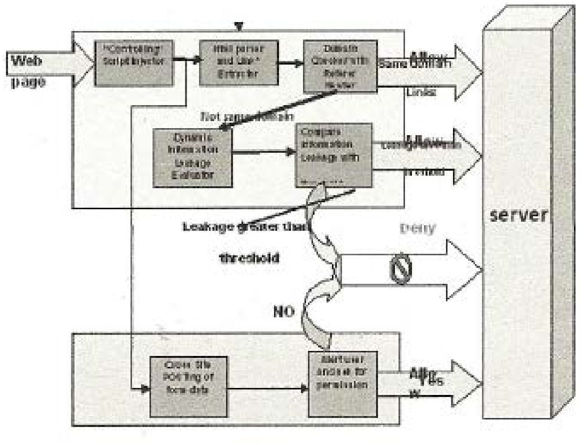


Fig 3: Block Diagram to detect XSS

THREAT MODEL

Attacker Abilities. Client-side XSS _lters are designed to mitigate XSS vulnerabilities in web sites without requiring the web site operator to modify the web site. We assume the attacker has the following abilities:

- The attacker owns and operates a web site.
- The user visits the attacker's web site.
- The target web site lets the attacker inject an arbitrary sequence of bytes into the entity-body of one of its HTTP responses.

Vulnerability Coverage. Ideally, a client-side XSS would prevent all attacks against all vulnerabilities. However, implementation is infeasible. Instead, we focus our attention on a narrower threat model that covers a certain class of vulnerabilities. For example, we consider only rejected XSS vulnerabilities, where the byte sequence chosen by the attacker appears in the HTTP request that retrieved the resource.

Attacker Goals: We assume the attacker's goal is to run arbitrary script in the user's browser with the privileges of the target web site. Typically, an attacker will run script as a stepping stone to disrupting the confidentiality or integrity of the user's session with the target web site. In the limit, the attacker can always inject script into a web site if the attacker can induce the user into taking arbitrary actions. In this paper, we consider attackers who seek to achieve their goals with zero interaction or a single-click interaction with the user.

5. IMPLEMENTATION & EXPERIMENTAL RESULTS

This solution was implemented using open source Mozilla Firefox 1.5 web browser from Mozilla foundation.
The Mozilla Firefox web browser executes JavaScript programs included in web pages with the help of the JavaScript engine called SpiderMonkey. The engine, written in C, is an important part of the web browser. It is used to execute JavaScript programs included in web pages as well as for the Gecko rendering engine that is used to display HTML, CSS, and XUL (Mozilla’s XML-based User interface language), and run JavaScript programs. The solution needed some major changes in the JavaScript engine and some minor changes in other components of the web browser. Some Data structures were created, and others were modified according to the need.

5.1. Security Evaluation

The proposed solution has been tested with thousands of malicious inputs, non vulnerable input with white listed tags and vulnerable websites. Fig. 4 shows comparison of the proposed browser with Firefox without security implemented, Microsoft's Internet Explorer, Apple's Safari Web Browser and other available web browsers on same platform and environment. It has been observed that there are more than 100 variants of XSS attacks exist and the approach is tested with the data collected from various research sites, white hat and black hat sites.

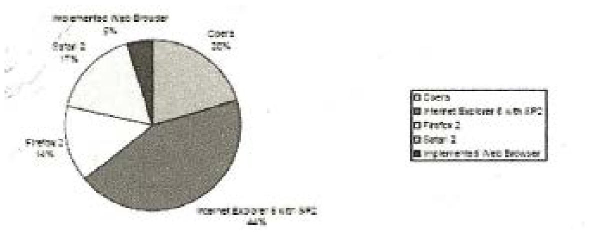
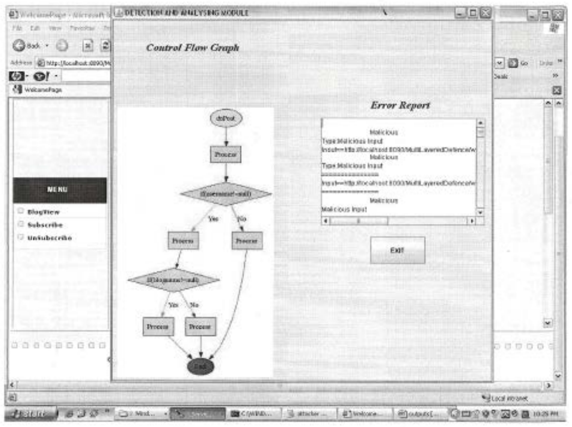


Fig 4: Security Evaluation of the proposed web browser

5.2. Performance Evaluation

The proposed solution has been tested with thousands of malicious inputs, non vulnerable input with white listed tags and vulnerable websites. Fig. 5 shows how the attacker can inject the malicious script code into a trusted website with the help of Control flow graph. This control flow graph intimates to the Client (Administrator) about when the attacker can hack the information, what are all the information that can be hacked. The vast of majority of XSS attacks can be prevented by identifying the user input locations within the web application and ensuring the source code handling these has proper measures in place. From a developer's perspective, this means ensuring all data inputted from a user is properly encoded to remove HTML and script markup to be replaced with text that all browsers can process.



```
Malicious
Type:Malicious Input
Input==http://localhost:8090/MultiLayeredDefence/w
Malicious
Type:Malicious Input
================
Input==http://localhost:8090/MultiLayeredDefence/w
================
Malicious
Malicious Input
```

Fig 5: Control Flow Graph displays how the attacker can inject the code

6. CONCLUSION

The proposed solution is found to be very effective by the experimental results. The solution is platform independent so we block suspected attacks by preventing the injected script from being passed to the JavaScript engine rather than performing risky transformations on the HTML. Cross-site scripting attacks are among the most common classes of web security vulnerabilities. Every browser should include a client-side XSS to help mitigate unpatched XSS vulnerabilities. Cross-site scripting is a Web-based attack technique used to gain information from a victim machine or leverage other vulnerabilities for additional attacks. These practices employ policy, people, and technology countermeasures to protect against XSS and other Web attacks.
In general, the system successfully prohibits and removes a variety of XSS attacks, maximizing the protection of web applications.

REFERENCES

[1] Engin Kirda, Christopher Kruegel, Giovanni Vigna, and Nenad Jovanovic. Noxes: A client-side solution for mitigating cross site scripting attacks. In Proceedings of the 21st ACM Symposium on Applied Computing (SAC), 2006.

[2] CERT. Advisory CA-2000-02: malicious HTML tags embedded in client web requests, <http://www.cert.org/advisories/CA- 2000-02.html>; 2000.

[3] CERT. Understanding malicious content mitigation for web developers, <http://www.cert.org/tech_tips/malicious_code_mitigation.html>; 2005

[4] D. Scott and R. Sharp. Abstracting Application-Level Web Security. In Proceedings of the 11th International World Wide Web Conference May 2002.

[5]Open Web Application Security Project, "The ten most critical web application security vulnerabilities",2007,ww.owasp.org/index.php/OWASP_Top_Ten_Project

[6] K. Fernandez and D. Pagkalos. Xssed.com - xss (cross-site scripting) information and vulnerabile websites archive. [online], http://xssed.com (03/20/08).