Protection against Client-Side Cross Side Scripting (XSS/CSS)

Utkarsh Kumar & Sujeet Kumar

Final year Students, Information Technology, Dronacharya College of Engineering, Gurgaon,

Maharshi Dayanand University, Rohtak, Haryana, India.

Email: utkarsh.kumar10@gmail.com , sujeetkumar0110@gmail.com

Abstract:

Cross Side Scripting are the most popular security problems nowadays. These Attacks make use of vulnerabilities in the code of web-applications, which result 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 moderate cross-site scripting Attacks. The existing client-side solutions degrade the performance of client's system resulting in a poor web surfing experience. But, this paper provides a solution to XSS, which will not degrade the browsing quality and will facilitate the user with a good surfing experience and at the very same time provide security against XSS.

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 user; typically in the form of a specially 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.

Unfortunately, any time one adds complexity to a system, 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.

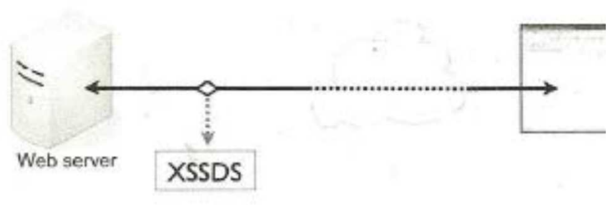


Fig 1: An Overview of XSS

2. Types of XSS

There are three types of XSS:

- Reflected Cross-Site Scripting attacks
- Stored Cross-Site Scripting attacks
- DOM based Cross-Site Scripting attacks

- **Reflected CSS:** This is the most common kind of CSS. 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. 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 CSS:** This kind of CSS is also known as HTML injection attack. These are scripting exploits 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 the Web site, if the 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.
- **DOM based CSS:** 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. 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. The Model we Propose

We assume the following:

Attacker Abilities: 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.

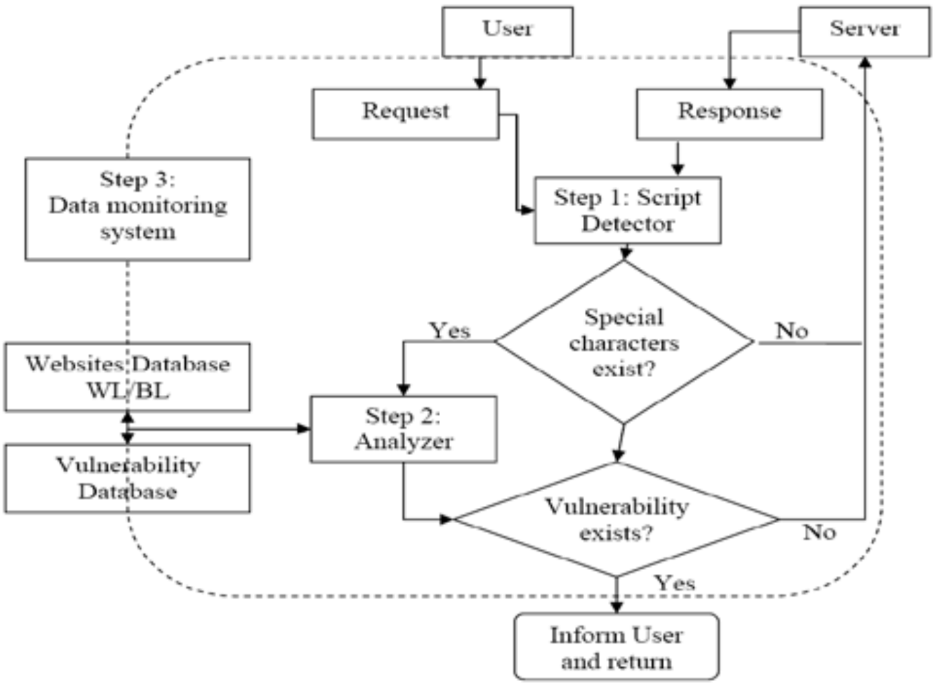


Fig 2: Proposed Solution, A three step process to detect CSS

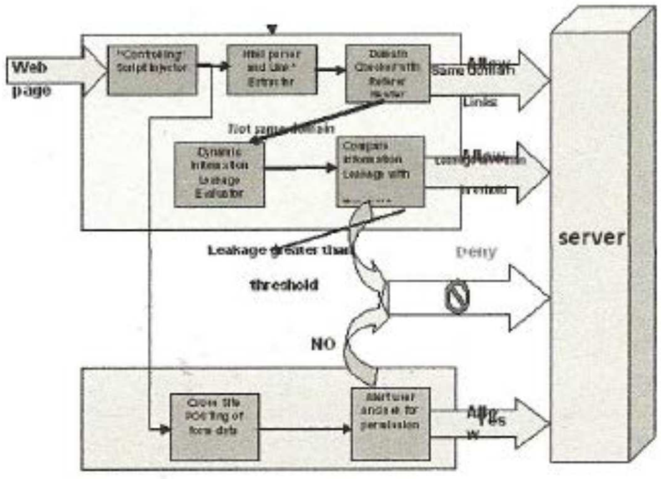

Web page
"Controlling" Script Injector
Html parser and Link Extractor
Domain Checker with Referer Header
Allow
Same domain Links
Not same domain
Dynamic Information Leakage Evaluator
Compare Information Leakage with
Allow
threshold
Leakage greater than threshold
Deny
NO
Cross Site POSTing of form data
Alert user and ask for permission
Allow
Yes
server

Fig 3: Block Diagram to detect CSS

4. Performance and Security Evaluation

The proposed solution has been tested with many malicious inputs, non-vulnerable input with white listed tags and vulnerable websites. The Figure 4 shows the 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.

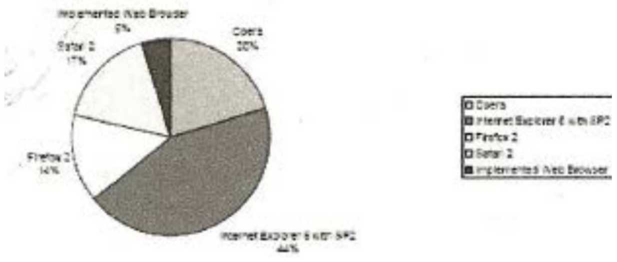
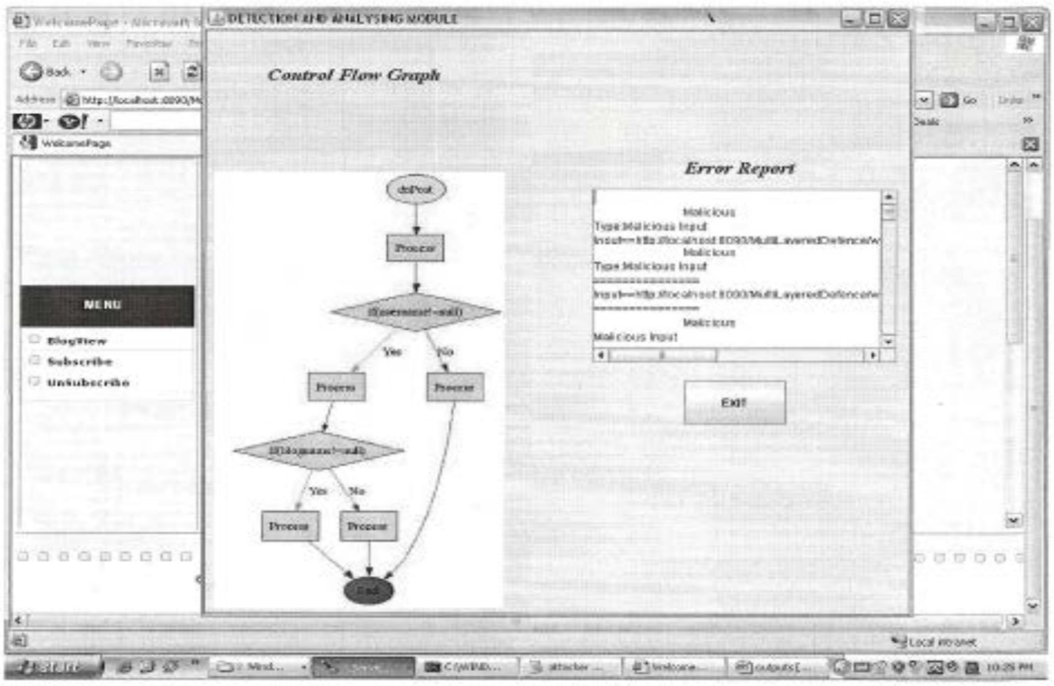


Fig 4: Security Evaluation of the proposed Web Browser

The proposed solution has been tested with many malicious inputs, non-vulnerable input with white listed tags and vulnerable websites. Figure 5 shows how the attacker can inject the malicious script code into a trusted website with the help of Control flow graph.



```
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: Security Evaluation of the proposed Web Browser\

5. Conclusions

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.

6. 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).

[7] IJCSI International Journal of Computer Science Issues, Vol. 8, Issue 4, No 1, July 2011 ISSN (Online): 1694-0814 www.IJCSI.org 654.

[8] Prevention of Cross-Site Scripting Attacks (XSS) on Web Applications in The Client Side, S.Shalini Usha, S.Usha, Department of Computer and Communication, Sri Sairam Engineering College, Chennai- 44, Tamilnadu, India. Department of Information Technology, Sri Sairam Engineering College, Chennai- 44, Tamilnadu, India, IJCSI International Journal of Computer Science Issues, Vol. 8, Issue 4, No 1, July 2011 ISSN (Online): 1694-0814, www.IJCSI.org

[9] P. Bisht, and V.N. Venkatakrishnan, "XSS-GUARD: Precise dynamic prevention of Cross-Site Scripting Attacks," In Proceeding of 5th Conference on Detection of Intrusions and Malware & Vulnerability Assessment, LNCS 5137, 2008, pp. 23-43.