PROXY BASED SOLUTION FOR MITIGATING CROSS-SITE SCRIPTING ATTACK IN CLIENT SIDE

A. Catherine Esther Karunya
PG Scholor
Karunya University
E-Mail: catherineesther@karunya.edu.in

C. Priyadharsini
Asst. Professor
Karunya University
E-Mail: priyadharsini@karunya.edu

D. Daniel
PG Scholor
Karunya University
E-Mail: Daniel_joen@yahoo.com

P. Priya
PG Scholor
Karunya University
E-Mail: priyapalaniswamy@gmail.com

ABSTRACT

The number and the importance of Web applications have increased rapidly over the last years. Along with the increased importance of Web applications, the negative impact of security flaws in such applications has grown as well. Cross-site scripting holes are web application vulnerabilities that allow attackers to bypass client-side security mechanisms normally imposed on web content by modern browsers. By finding ways of injecting malicious scripts into web pages, an attacker can gain elevated access privileges to sensitive page content, session cookies, and a variety of other information maintained by the browser on behalf of the user. Cross-site scripting attacks are therefore a special case of code injection. In this paper, a proxy based solution is proposed for detecting and preventing cross-site scripting attacks in the client side without degrading the user’s browsing experience and also provides additional security by making use of SSL support.
Index Terms: Application-level web Security, Cross-site scripting, Malicious Scripts, SSL and Security vulnerabilities.

I. INTRODUCTION

Web Applications have become one of the most important communication channels between various kinds of service providers and clients on the Internet. Along with the increased importance of Web applications, the negative impact of security flaws in such applications has grown as well. Vulnerabilities that may lead to the compromise of sensitive information are being reported continuously, and the costs of the resulting damages are increasing. Cross-site scripting (XSS) is an attack against web applications in which scripting code is injected into the output of an application that is then sent to a user’s web browser. In the browser, this scripting code is executed and used to transfer sensitive data to a third party, where the malicious script is given full access to all resources. A successful XSS attack can lead to compromised authentication information, privilege escalation, or disclosure of confidential data. Such situations arise due to lacking security awareness on part of the developers, or due to programming mistakes induced by financial and time constraints. One reason is the high flexibility of HTML encoding schemes, offering the attacker many possibilities for circumventing server-side input filters that should prevent malicious scripts from being injected into trusted sites. This allows an attacker to circumvent the same-origin policy which states that scripts loaded from a certain domain cannot access data belonging to any other domain and sand boxing mechanism which allows the code to perform a restricted set of operations.

CATEGORIES OF XSS ATTACKS

There are currently three major categories of Cross- Site Scripting flaws.

- Non-Persistent XSS attacks
- Persistent XSS attacks
- Local XSS attacks

Non-Persistent XSS attacks

Non-persistent XSS vulnerabilities could allow malicious sites to attack the users who visit the particular website while logged in. An attack 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.[8] Since HTML documents have a flat, serial structure that mixes control statements, formatting, and the actual content, any non-validated user-supplied data included in the resulting page without proper HTML encoding, may lead to mark-up injection. The attack can be done by using a link using a malformed URL, such that a variable passed in a URL to be displayed on the page contains malicious code. Another Uniform Resource Locator (URL) used by the server-side code to produce links on the page, can also become a vulnerability employed in a reflected Cross-Site Scripting flaws.

**Persistent XSS attacks**

In a Persistent or Stored XSS attack, the malicious JavaScript code is permanently stored on the target server. This type of Cross-Site Scripting flaws can affect any user to our website, if our site is subject to persistent Cross-Site Scripting vulnerability. One of the examples of persistent or stored vulnerability is a persistent cross-zone scripting vulnerability coupled with a computer worm allowed execution of arbitrary code and listing of file System contents via a QuickTime movie on MySpace. Preventing reflected flaws, the key to securing our web site against stored flaws 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.

**Local XSS attacks**

A local or Document Object Model Cross-Site Scripting flaws targets vulnerabilities within the code of a web page itself. These types of vulnerabilities are the result of incautious use of the Document Object Model in JavaScript so that opening another web page with malicious JavaScript code in it at the same time might actually alter the code in the first page on the local system. DOM-based vulnerabilities occur in the content processing stages performed by the client, typically in client-side JavaScript.[6]

This paper is structured as follows. Section II gives an overview about some of the server side and client side techniques used in detecting the cross site scripting attack. Section III describes the proposed proxy based solution for mitigating the cross-site scripting attack in client side with the extension of SSL support which provides additional security, confidentiality and integrity to the users. Section IV summarizes the implementation part.
and shows the empirical results. Finally Section V briefly concludes the vulnerability of cross-site scripting attack and its proposed solution.

II. RELATED WORKS

Cross site scripting (XSS), is the most widespread and harmful web application security issue. So far, there have been many Server and client side solutions for detecting and preventing it. Noxes, [1] a client side solution is proposed to mitigate cross site scripting attacks. It is a web proxy which uses both manual and automatically generated rules to mitigate possible cross-site scripting attempts is used. It effectively protects against information leakage from the user’s environment while requiring minimal user interaction and customization effort. SWAP (Secure Web Application Proxy) [2], a server-side solution is used for detecting and preventing cross-site scripting attacks by utilizing a reverse proxy equipped with a Web browser in order to detect malicious JavaScript content. Scott and Sharp [3] states that application-level web security refers to vulnerabilities inherent in the code of a web-application itself. This paper investigates new tools and techniques which address the problem of application-level web security.

Web application security assessment by behavior monitoring [4] adopts software-engineering techniques to design a security assessment tool for Web applications. Using a “complete crawling” mechanism, a reverse engineering of a Web application is performed to identify all data entry points.

PIXY [5] addresses the problem of vulnerable Web applications by means of static source code analysis. Data flow analysis is used to discover vulnerable points in a program. In addition, alias and literal analysis are employed to improve the correctness and precision of the results.

CERT- Center of internet security expertise,[11] a federally funded research and development center states that none of the client side solutions prevent the vulnerabilities completely and it is up to the server to eliminate these issues.

XSSDS (Server-side Cross Site Scripting Detection System) [6] is a passive detection system to identify successful XSS attacks. [7]Two novel detection approaches are used to identify reflected XSS attacks and stored XSS attacks. This approach does not require any changes to the actual application or infrastructure. Both attack detection methods depend solely on access to the HTTP traffic. This paper focuses on a proxy
based solution for cross-site scripting attack with high security and minimal user interaction.[10] The main difference of our approach with respect to the existing solutions is the provision of SSL support to the proxy to mitigate XSS attacks.

III. PROXY BASED SOLUTION

To be useful in practice, a mitigation technique for XSS attacks has to satisfy two properties. First, it has to be effective in detecting and preventing XSS attacks with minimal user interaction. Second, it should provide security. The proposed architecture provides a proxy based solution which contains a web proxy which fetches web pages on behalf of the user’s web browser that satisfies the above properties.

The various steps involved in mitigating the XSS attacks are as follows

Web Proxy

This web proxy fetches HTTP requests on behalf of the user’s browser. Hence, all web connections of the browser pass through this proxy and can either be blocked or allowed based on the current security policy [1]. It provides an additional layer of protection that includes SSL support.

SSL Support

Secure Socket Layer (SSL) allows web browsers and web servers to communicate over a secure connection. In this secure connection, the data that is being sent is encrypted before being sent and then is decrypted upon receipt and before processing. Both the browser and the server encrypt all traffic before sending any data. SSL addresses the following important security considerations. Every connection is associated with one session. Session can be reused across multiple secure connections. SSL provides additional services such as peer entity authentication, data confidentiality, data authentication and integrity, compression/decompression, generation/distribution of session keys, integrated into protocol and security parameter negotiation. SSL notifies the users with alert messages.[3]

Java Script Controller

Java script controller injects script code to all webpages that it fetches. This script checks if the page that is being displayed is a pop up window or a frame. The injected code checks the top level domain to determine if the pop up window or the frame has a
A SSL alert message is generated when domains differ, informing the user that there is a potential security risk.

**Domain Information Retriever**

Domain Information Retriever extracts all the links and source from the main webpages and parses the frames recursively in this web page itself. It further checks the directory of the domain from the web pages links. Static links that are extracted from the web page include HTML elements with href, src.

**Top Level domain Checker**

Top Level domain Checker is used to check the extracted domain links with referrer header.

The attacker has no means of spoofing or changing the referrer header. It is used to determine whether a link is local or not. Finally it compares the domain links with the
referrer header. If the domain links are same they are sent to the server. If domain links differ then it is sent to the evaluator.

Evaluator

Evaluator is used to evaluate the domain links that does not match with the referrer header. [1] The total number of information leaked to the external domain is zero, if the number of request to any of the links is zero. If the number of requests to external links is greater than zero or less than or equal to the total number statically embedded external links in a page, the information leakage is calculated using the below formula,

\[ I = \begin{cases} \frac{n}{r} & \text{if } r \leq n \\ 0 & \text{if } r = 0 \end{cases} \]

Where,

- \( n \) = Total no of statically embedded external links
- \( r \) = no of requests to any of these links
- \( I \) = Amount of information that can be leaked

Threshold Value

The Information leakage value is compared with the customizable threshold. If leakage is less than the threshold then the server information can be accessed. If leakage is greater than the threshold access to the server information is denied on receiving an SSL alert.

IV. EVALUATION

We implemented a client side solution in Java and extracted information about some visited web pages. Analyzing the web page, we were able to determine how many static links each visited web page contained, how many of these links were pointing to external domains, how many external links were actually requested by the user browser. We used a Java utility called Html Parser to extract the static hyperlinks in the page by looking at HTML elements. We analysed that login pages, personal information pages, shopping cart checkouts, or any pages where credit card information is transmitted is made secure with SSL support. The table shows the statistical information of the analysed web pages.
From the results outlined from above, we can conclude that our enhanced XSS mitigation technique provides a better security against cross-site scripting attacks.

**CONCLUSION**

Web applications have become dominant and widespread interaction medium. Vulnerabilities that present a threat to the personal data of users are discovered regularly. A specific case of attack against web applications are studied and seen how the existence of cross-site scripting (XSS) vulnerabilities on web application can involve a great risk for both the application itself and its users. In this paper, we presented a proxy based solution for protecting the user’s of a web application from cross-site scripting attacks. This web proxy fetches HTTP requests on behalf of the user’s browser and also provides an additional layer of protection that includes Secure Socket layer (SSL) support. SSL allows web browsers and web servers to communicate over a secure connection. Pages that require sensitive information to be transmitted are made secure using SSL. A prototypical implementation demonstrated our approach’s capabilities to reliably detect and mitigate cross-site scripting attacks while maintaining a tolerable alarm rate.

**REFERENCES**


A.Catherine Esther Karunya received her B.E degree in Information Technology from Karunya University in 2009. She is currently doing Post graduate in Karunya University and now works on the project in Computing Security and continues her research in Networking

C.Priyadharsini, is working as an Asst Professor in the department of Computer Science and Engineering since 2008 and focuses on the area of embedded system and peer to peer systems.
D. Daniel received the B.E degree in Information Technology from Karunya University in 2009. He is currently doing Post graduate in Karunya University, now works on the project in parallel and distributed Systems, and Continues research on adaptive scheduling techniques in grid computing.

P. Priya received the B.E degree in Information Technology from Anna University in 2009. She is currently with the Post graduate in Karunya University, now works on the project in mesh networks and continues her research on networking.