Volume 4, Issue 3, March 2014 ISSN: 2277 128X

International Journal of Advanced Research in Computer Science and Software Engineering

Research Paper

Available online at: www.ijarcsse.com

A Survey on XSS web-attack and Defense Mechanisms

Amit Singh, S Sathappan

Department of Computer Science and Engineering

LNCTS, Bhopal, India

Abstract— Cross-site scripting (XSS) is a scripting attack on web pages and accounted as one of the most dangerous vulnerability found in web applications. Security researchers investigated several issues and found XSS vulnerability in most of the popular websites. Once the vulnerability is exploited, an attacker gains a voluntary access of the legitimate user's web-browser and may perform cookie-stealing, malware-spreading, session-hijacking, and malicious redirection. As a preventive measure against such attacks, it is imperative to adopt security measures that inevitably block the third party interference. The most recent attack on existing websites is DOM based Cross Site Scripting attack, it is much harder to detect and therefore it must be prevented. This attack can harm millions of people in a few seconds. Vulnerabilities of websites are mostly exploited through HTTP GET submission method and HTTP POST submission method. To prevent the existing websites from the XSS attacks, a methodology of two way detector & filter is developed.

Keywords— Cross-Site Scripting (XSS); SQL Injection; Static Taint analysis; symbolic execution ; DOM based XSS.

I. INTRODUCTION

According to security experts, cross-site scripting is amongst the most serious and common threats in Web applications today, surpassing buffer overflow, it has become the number one vulnerability for the past decade. XSS is the result of a weakness inherent in many Web applications security mechanism, absence or insufficient sanitization of user inputs. XSS flaws exist in Web applications written in various programming languages such as PHP, Java, and .NET where web pages processes unrestricted user inputs. Attackers inject malicious code via these inputs, thereby causing unintended script executions by client's browsers. Researchers have proposed multiple XSS solutions ranging from simple static analysis to complex runtime protection mechanisms. However, vulnerabilities continue to exist in many Web applications due to developer's lack of understanding of the problem and their unfamiliarity with current defenses, strengths and limitations.

A. *XSS Exploits*

XSS exploits are similar to SQL injection, an original form of code injection. This type of attack exploits an application's output function that references poorly sanitized user input. However, SQL injection targets the query function that interacts with the database, whereas XSS exploits target the HTML output function that sends data to the browser. The basic idea of XSS injection is to use special characters to cause Web browser interpreters to switch from a data context to a code context. For example, when an HTML page references a user input as data, an attacker might include the tag <script>, which can invoke the Java- Script interpreter. If the application does not filter such special characters, XSS injection is successful, and the attacker can perform exploits such as account hijacking, cookie poisoning, denial of service (DoS), and Web content manipulation. Typical input sources that attackers manipulate include HTML forms, cookies, URLs, and external files. Attackers often favor JavaScript and also other kinds of client-side accessed user input in the outgoing web- page. This type of XSS exploit is common in error messages and search results.

The XSS project recently reported multiple reflected XSS holes in McAfee that attackers could exploit to trick users into downloading viruses. Stored or persistent XSS holes exist when a server program stores user input containing injected code in a persistent data store such as a database and then references it in a webpage. Attack against social networking sites commonly exploits this type of XSS flaw. An example is the Samy worm, which, within less than 24 hours after its release on 4 October 2005, caused an exponential growth of friend lists for 1 million Myspace users, effectively creating a DoS attack. Both reflected and stored XSS holes result from improper handling of user inputs in server-side scripts. In contrast, DOM-based XSS holes appear in the Web application when client-side scripts reference user inputs, dynamically obtained from the Document Object Model structure, without proper validation.

B. *Example XSS Exploits*

For a Web application that lets travelers share tips about the places they have visited. The program contains four input fields—"Action," "Place," "Tip," and "User"—that attacker can manipulate. An attacker could send a seemingly innocuous URL link to a victim via e-mail or a social networking site. The script in bold will execute on the victim's browser if the victim follows the link to traveling Forum.

C. *XSS Defenses*

XSS defenses can be broadly classified into four types: defensive coding practices, XSS testing, vulnerability detection.

1) Defensive Coding

Because XSS arises from the improper handling of inputs, using defensive coding practices that validate and sanitize inputs is the best way to eliminate XSS input validation as it ensures that user inputs conform to a required input format. There are four basic input sanitization options. Replacement and removal methods search for known bad characters (blacklist comparison); the former replaces them with non malicious characters, whereas the latter simply removes them. Escaping methods search for characters that have special meanings for client-side interpreters and remove those meanings. Restriction techniques limit inputs to known good inputs (white list comparison). Checking blacklisted characters in the inputs is more scalable, but blacklist comparisons often fail as it is difficult to anticipate every attack signature variant. White list comparisons are considered more secure, but they can result in the rejection of many unlisted valid inputs. OWASP has issued rules that define proper escaping schemes for inputs referenced in different HTML output.

2) XSS Testing

Input validation testing could uncover XSS vulnerabilities in Web applications. Specification-based IVT methods generate test cases with the aim of exercising various combinations of valid/invalid input conditions stated, to avoid the sole dependency on specifications, Nuo Li and colleagues attempted to infer valid input conditions by analyzing input fields and their surrounding texts in client-side scripts. Code-based IVT methods apply static analysis to extract valid/invalid input conditions from server-side scripts. In general, the effectiveness of both specifications, the code-based approaches relies largely on the completeness of specifications or the adequacy of generated test suites for discovering XSS vulnerabilities in source code. Only test cases containing adequate XSS attack vectors can induce original and mutated programs to behave differently. Hossain Shahriar and Mohammad Zulkernine developed MUTEC, a fault-based XSS testing tool that creates mutated programs by changing sensitive program statements, or sinks, with mutation operators. Cross site scripting (XSS) vulnerability is caused by the failure of web application in sanitizing user inputs embedded in HTML output pages. Through such inputs, there is a possibility that an attacker injects malicious scripts in the applications. Furthermore, the attacker's purpose may be served when a client subsequently visits an exploited web page causing the injected scripts to be executed by the client's browser. Thus, XSS attack is a type of code injection attack. In addition, injected scripts are written in any type of client-side scripts such as JavaScript, Action Script and VBScript. XSS has been ranked amongst the top two common and serious security laws. In the past, even giant web sites such as HSBC, Google Search Engine, Facebook, MySpace and Vodafone have been reported to contain this type of vulnerability.

Code-based extraction of XSS defense artifacts in this concept, it presents the concepts on extracting the program artifacts, which serves the purpose for securing the program from input manipulation attacks. These concepts are built on modeling the possible code patterns of defensive coding methods. Through the empirical studies on many web applications, they observed that the following methods are generally implemented to prevent XSS: (a) input validation; (b) escaping; (c) filtering. These methods are also addressed as sanitization methods in the literature.

a) Input validation

It is a traditional approach for handling external data in web applications. This method could reject invalid input immediately. Originally, it is used to ensure input data correctness but in today's web applications, data accuracy could also ensure data security; therefore nodes in a CFG which implement this defense method should be extracted and checked for adequacy in defending against XSS. Let G be a CFG of a given web program. Let k be a pv-out node in G and vk be a tainted variable referenced at k. There may be more than one tainted variable referenced at k. It is common that the program uses predicate nodes or exception nodes to allow vk to be operated at k only if the value of vk satisfies the user interface specification or some required conditions. It provides a definition that characterizes such node pattern.

b) Characterizing XSS Defense through Escaping

Although input validation may be used as a primary defense against all kinds of input manipulation attacks, validation methods may not defend against all XSS attacks. Therefore for absolute prevention of code injection attacks such as XSS, escaping (also called encoding) is often used to complement input validation. Escaping is a technique that ensures any special characters significant to a certain interpreter are just treated as data not as code. To prevent XSS, proper escaping methods, such as HTML entity escaping, URL escaping, and JavaScript escaping, need to be used according to the context in which the tainted data are referenced (i.e. according to the type of client-side interpreter interpreting the tainted data) . Hence, this method will not work if the escaping scheme is inappropriate. For example, a developer may use SQL escaping scheme (i.e. special characters significant to SQL parser are escaped) on the tainted data assuming that the data are only to be referenced in SQL statements. However, if the data are also used in HTML outputs, the adopted SQL escaping scheme will not escape the special characters significant to the HTML interpreters, thus causing XSS vulnerability. Therefore nodes implementing such defense method need to be extracted and examined for adequacy.

c) Characterizing XSS Defense through Filtering

Although escaping could completely prevent XSS, it is required that the correct escaping method is applied depending on the context in which the tainted data are referenced. As the use of a standard escaping library is also required, some web applications may not prefer this method. Instead they may apply filtering method to prevent XSS. Filtering is a technique that either removes or replaces malicious characters with non-malicious ones. Among the discussed three defense methods the filtering method is equally important in many web applications as the flow of tainted data through the filtering methods and into the HTML outputs can be very seamless. Such filtering techniques had implemented in a web application is generally carried out by nodes in G that influence the tainted variables of pv-out node.

3) Vulnerability Detection

Cross-site scripting (XSS) vulnerabilities can be classified into two types:

- Non-persistent (or reflected) cross-site scripting is a most commonly exploited XSS vulnerability. In this type of attack the malicious data is reflected immediately on the page by the server without proper sanitization.

- Persistent (or stored) cross-site scripting vulnerability occur when the attacker injects the malicious code as user input into the server and the code is saved permanently by the server. Thereafter, it is displayed every time as a page of result to the users visiting the webpage in the course of regular browsing. For this reason stored XSS is much more devastating than the reflected XSS. By exploiting the stored XSS vulnerability, attacker may replicate large amount of malicious data to the users (For example the Samy XSS worm that affected Myspace a few years ago).

II. LITERATURE SURVEY

Researchers have proposed multiple solutions to cross-site scripting, but vulnerabilities continue to exist in many Web application's due to developers' lack of understanding of the problem and their unfamiliarity with current defenses, strengths and limitations. Existing techniques for defending against XSS exploits suffer from various weaknesses: inherent limitations, incomplete implementations, complex frameworks, runtime overhead, and intensive manual-work requirements. Security researchers can address these weaknesses from two different perspectives. From a development perspective, researchers need to craft simpler, better, and more flexible security defenses. They need to look beyond current techniques by incorporating more effective input validation and sanitization features. In time, development tools will incorporate security frameworks such as ESAPI that implement state-of-the-art technology [1].

Author Kieyzun et al. advised an automatic technique for creating inputs that expose SQLI and XSS vulnerabilities. The technique generates sample inputs, symbolically tracks tainted data through execution (including through database accesses), and mutates the inputs to produce concrete exploits. This technique creates real attack vectors, has few false positives, incurs no runtime overhead for the deployed application, works without requiring modification of application code, and handles dynamic programming-language constructs. The author also implemented the technique for PHP, in a tool Ardilla. This approach was implemented in a tool called BLUEPRINT that was integrated with several popular web applications. The authors evaluated BLUEPRINT against a barrage of stress tests that demonstrate strong resistance to attacks, excellent compatibility with web browsers and reasonable performance overheads [2].

Scott & Sharp investigated new tools and techniques which address the problem of application-level Web security. They 1) described a scalable structuring mechanism facilitating the abstraction of security policies from large Web-applications developed in heterogeneous multiplatform environments; 2) presented a set of tools which assist programmers in developing secure applications which are resilient to a wide range of common attacks. 3) Proposed signature based misuse detection approach. It expresses a security layer on top of the web application, so that the existing web application remain unchanged whenever a new threat is introduced that demands new security mechanisms. They claim that this approach is very effective as it addresses the vulnerabilities at a granular level of tags and attributes, in addition to addressing the XSS vulnerabilities [3].

In this paper, on the basis of the possible implementation patterns of defensive coding methods author extracts all such defenses implemented for securing each potentially vulnerable HTML output. Author also introduces a variant of control flow graph, called tainted information flow graph, as a model to audit the adequacy of XSS defense artifacts. The author evaluated the proposed method based on the experiments on seven Java-based web applications. In the auditing experiments, there approach was effective in recovering all the XSS defense features implemented in the test subjects. The extracted artifacts were also shown to be useful for filtering the false-positive cases reported by a vulnerability detection method and helpful in fixing the vulnerable code sections.

Cross site scripting (XSS) vulnerability is mainly caused by the failure of web applications in sanitizing user inputs embedded in web pages. Even though state-of-the-art defensive coding methods and vulnerability detection methods are often used by developers and security auditors, XSS flaws still remain in many applications because of (i) the difficulty of adopting these methods, (ii) the inadequate implementation of these methods, and/or (iii) the lack of understanding of XSS problem. To address this issue, this study proposes a code-auditing approach that recovers the defense model implemented in program source code and suggests guidelines for checking the adequacy of recovered model against XSS attacks [4].

In this paper, they have analyzed all the techniques those have been used to detect XSS and arrange a number of analyses to evaluate performances of those methodologies. Cross-Site Scripting is one of the main problems of any Web-based service. Since Web browsers support the execution of commands embedded in Web pages to enable dynamic Web pages, attackers can make use of this feature to enforce the execution of malicious code in a user's Web browser. To augment the user's experience many web applications are using client side scripting languages such as JavaScript but the growing of JavaScript is increasing serious security vulnerabilities in web application too, such as cross-site scripting (XSS) [5].

In this paper, author proposes a passive detection system to identify successful XSS attacks. Based on a prototypical implementation, they examine accuracy of approach and verify its detection capabilities. They compiled a data-set of HTTP request/response from 20 popular web applications for this, in combination with both real word and manually crafted XSS exploits; detection approach results in a total of zero false negatives for all tests, while maintaining an excellent false positive rate for more than 80 percent of the examined web applications.

In this world of networking where people around the globe are connected, Cross-site Scripting (XSS) has emerged to one of the most prevalent growing threat. XSS attacks are those in which attackers inject malicious codes, most often client-side scripts, into web applications from outside sources. Because of the number of possible injection location and

techniques, many applications are vulnerable to this attack method. Even though the main reason for the vulnerability primarily lies on the server side, the actual exploitation is within the victim's web browser on the client side [6].

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.

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 generates 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 [7].

In this paper, initially they have tried out the experiments on the exploitation of XSS vulnerabilities using local host server (i.e. XAMPP). After this, they have investigated for the XSS vulnerabilities on social networking sites (like Facebook, Orkut, Blogs, Twitter etc.) and tried to exploit the same on blogs. Finally, on the basis of some analysis and results, they have discussed a novel technique of mitigating this XSS vulnerability by introducing a Sandbox environment on the web browser.

Attacks on web applications are growing rapidly with the opening of new technologies, HTML tags and JavaScript functions. Cross-Site Scripting (XSS) vulnerabilities are being exploited by the attackers to steal web browser's resources (cookies, credentials etc.) by injecting the malicious JavaScript code on the victim's web applications. The existing techniques like filtering of tags and special characters, maintaining a list of vulnerable sites etc. cannot eliminate the XSS vulnerabilities completely [8].

In this paper, a novel technique called Dynamic Hash Generation Technique is introduced whose aim is to make cookies worthless for the attackers. This technique is implemented on the server side and its main task is to generate a hash value of name attribute in the cookie and send this hash value to the web browser. With this technique, the hash value of name attribute in the cookie which is stored on the browser's database is not valid for the attackers to exploit the vulnerabilities of XSS attacks.

Cookies are a means to provide state full communication over the HTTP. In the World Wide Web (WWW), once the user using web browser has been successfully authenticated by the web server of the web application, then the web server will generate and transfer the cookie to the web browser. Now each time, if the user wants to send a request to the web server as a part of the active connection, the user has to include the corresponding cookie in its request, so that the web server associates the cookie to the corresponding user. Cookies are the mechanisms that maintain an authentication state between the user and web application. Therefore cookies are the possible targets for the attackers. Cross Site Scripting (XSS) attack is one of such attacks against the web applications in which a user has to compromise its browser's resources [9].

The attack specially focuses on Cross Site Scripting attacks. The author further discusses types and several counter measures. The major problem faced by the web application is the parameter manipulation, through which the attackers are aiming to access the database. Generally web applications maintain same structure and value. In that, required information is being accessed by the identical variables and keywords through web parameters. Parameter manipulation is the major issue in the web application used by the attacker to manipulate the parameter being sent by the browser and executed by the server. These vulnerabilities occur after the string gets returned to the user's web browser by a susceptible web application. Therefore, to prevent XSS vulnerabilities, it is obligatory to prepare preventative measures to protect the parsing processing in the web browser so that there is no influence even from the effect of the string prepared by the attacker [10].

An XSS vulnerability are found in those Web applications that accepts data from users and dynamically include it in servers then on web-pages without properly validating the data. In this process, if an attacker found XSS vulnerability then it allows him to execute arbitrary commands and display content in a victim's browser. In a successful XSS attack, an attacker controls the victim's browser or account on the vulnerable Web application. The reliability of an XSS vulnerability lies in the fact that the malicious code executes in the context of the victim's session, allowing the attacker to bypass normal security restrictions. . "A classic example of this is with online message boards where users are allowed to post HTML formatted messages for other users to read" Complete Cross-site Scripting Walkthrough [11].

In December 2006, Stefano Di Paola and Giorgio Fedon described a universal XSS attack against the Acrobat PDF plugin. When the client clicks the link and the data is processed by the page (typically by a client side HTML-embedded script such as JavaScript), the malicious JavaScript payload gets embedded into the page at runtime. [12].

III. PROBLEM STATEMENT

An attacker can lure the client to render the page containing the URL (the location and/or the referrer) partly controlled by the attacker. When the client clicks the link and the data is processed by the page (typically by a client side HTML-embedded script such as JavaScript), the malicious JavaScript payload gets embedded into the page at runtime. Input validation using HTTP POST submission method and HTTP GET submission method are vulnerable to XSS attacks as it allows the user to manipulate the website controls using code implementation. Although the HTTP GET can be exploited

more easily by an attacker because all it needs is to change the URL. The exploitation may change according to submission methods used by different web-browsers.

IV. CONCLUSIONS & FUTURE WORK

Patching XSS vulnerability is arduous as we can never be 100% sure that no-one can penetrate the filter. But attackers always find ways to breach websites filter and exploit the vulnerability. For this purpose, it is necessary to get updated with the latest XSS vectors. Therefore, considering the emergence of web technologies, a new proposal is introduced which will perhaps enhance the security of websites by thoroughly analyzing the input and URL validation process. To prevent the XSS attacks in existing websites, a two way detector and filter can be developed. The detector will identify any suspicious URL submitted or stored in the database of website and report to filter. Filter will sanitize that particular data to passive text before storing in the database of website. Filter can also be programmed to sanitize previously store data. This approach can effectively prevent XSS attacks on websites.

REFERENCES

[1] Lwin Khin Shar and Hee Beng Kuan Tan, "Defending against Cross-Site Scripting Attacks", IEEE Computer, Vol. 45(3), pp 55-62, March-2012.

[2] A.Kieyzun, P.J. Guo,K. Jayaraman, and M.D. Ernst, "Automatic Creation of SQL Injection And Cross-Site Scripting Attacks", ICSE '09 Proceedings of the 31st International Conference on Software Engineering, pp. 199-209, May 2009.

[3] D. Scott and R. Sharp, "Specifying and enforcing application-level Web security policies", IEEE Transactions on Knowledge and Data Engineering, vol. 15, no.4, pp. 771-783, July-Aug 2003.

[4] L.K. Shar and H.B.K. Tan, "Auditing the XSS defence features implemented in web application programs", Software, IET, Vol. 6, Iss. 4, pp. 377–390, 2012.

[5] Tejinder Singh, "Detecting and Prevention Cross–Site Scripting Techniques", IOSR Journal of Engineering, Vol. 2(4) pp. 854-857, April-2012.

[6] M. James Stephen, P.V.G.D. Prasad Reddy, Ch. Demudu Naidu and Ch. Rajesh, "Prevention of Cross Site Scripting with E-Guard Algorithm", International Journal of Computer Applications, Vol. 22(5), May-2011.

[7] S. Shalini and S. Usha, "Prevention Of Cross-Site Scripting Attacks (XSS) On Web Applications In The Client Side", IJCSI International Journal of Computer Science Issues, Vol. 8(4), July-2011

[8] Shashank Gupta and Lalitsen Sharma, "Exploitation of Cross-Site Scripting (XSS) Vulnerability on Real World Web Applications and its Defense", International Journal of Computer Applications, Vol. 60 (14), December-2012.

[9] Shashank Gupta, Lalitsen Sharma, Manu Gupta and Simi Gupta, "Prevention of Cross-Site Scripting Vulnerabilities using Dynamic Hash Generation Technique on the Server Side", International Journal of Advanced Computer Research, Vol. 2(3), September-2012.

[10] Vishwajit S. Patil, Dr. G. R. Bamnote and Sanil S. Nair, "Cross Site Scripting: An Overview", International Symposium on Devices MEMS, Intelligent Systems & Communication, Proceedings published by International Journal of Computer Applications (IJCA), 2011.

[11] Ahmed Elhady and Mohamed, "Complete Cross-site Scripting Walkthrough", 23rd CCC Conference, December 2006.

[12] Subverting Ajax Stefano Di Paola, Giorgio Fedonhttp://events.ccc.de/congress/2006/Fahrplan/attachments/1158-Subverting_Ajax.pdf.

