International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 3 Issue 1, January - 2014

Identification and Defense Mechanisms for XSS Attack

Nency Patel
Department of Computer Engineering
D.J.Sanghavi College of engineering
Mumbai, India

Narendra Shekokar
Department of Computer Engineering
D.J.Sanghavi College of engineering
Mumbai, India

***Abstract*— XSS vulnerabilities are growing with each day and at the same time their disclosure rate is alarming. XSS attacks are generally simple, since HTML encoding is known for its simplicity and flexibility providing the attacker means for bypassing server-side input filters. Several approaches have been proposed to mitigate XSS attacks. The different techniques for prevention of XSS such as NOXES, NoMoXSS and ARDILLA have been studie. Based on the comparative analysis we have proposed a solution to eliminate the dependency on the browser's parser to build parse trees for untrusted HTML tags. This proposed solution converts untrusted contents to atrusted model, which is generated for each comment in the HTML page which has malicious code. Thus the application's output is modified by replacing each instance of untrusted HTML with its corresponding model and leaving trusted content unaltered.**

Keywords—Cross site scripting, Noxes, NoMoXSS, ARDILLA.

I. INTRODUCTION

Cross-site Scripting (XSS) is the name of a class of security loopholes that can occur in Web applications. These are all vulnerabilities that allow an attacker to inject HTML Markup or JavaScript into the affected Web application's front-end client [6].

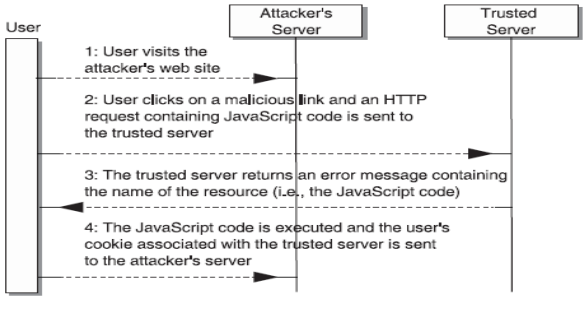

Figure 1: Typical XSS Scenario

Usually the attacker will encode the malicious portion of the link to the site in HEX (or other encoding methods) so the request is less suspicious-looking to the user. After the data is collected by the web application, it creates an output page for the user containing the malicious data that was originally sent to it, but in a manner to make it appear as valid content from the website. Figure 1 shows the typical cross site scripting scenario.

A. Types of Cross Site Scripting

- Persistent or Stored XSS:

A persistent cross-site scripting vulnerability is when the attacker provides malicious data to the web application and is stored permanently on a database or some other similar storage. This malicious data can at a later date be accessed and executed by the victims without filtering or sanitizing it [8].

- Non Persistent or Reflected XSS:

Reflected XSS takes place when the data provided by the attacker is used immediately by the web application right away in some response. This is what happens in website error messages and search results.

- DOM based XSS:

The Document Object Model (DOM) is a method for representing and interacting with objects in HTML/XML. Browsers work with the DOM; when a page is loaded the browser parses the page into an object structure. DOM-based XSS occurs in the content processing stages performed by the client [3].

The remaining part of this paper is organized as follows. Three different cross site scripting prevention techniques are discussed and compared in Section 2, followed by the proposed solution based on disadvantages of three techniques in Section 3 and we conclude in Section 4.

II. LITERATURE SURVEY

One reason for the popularity of XSS vulnerabilities is that developers of web-based applications often have little or no security background. Moreover, business pressure forces these developers to focus on the functionality for the end-user and to work under strict time constraints, without the resources (or the knowledge) necessary to perform a thorough security analysis of the applications being developed. 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. While being the obvious course of action, this approach leaves the user completely open to abuse if the vulnerable web site is not willing or able to fix the security issue.

E. Kirda et al, have proposed a solution called NOXES which is the first client-side solution to alleviate cross-site scripting attacks. The Noxes was inspired by windows personal firewalls which are widely used on PCs and notebooks. Although personal firewalls play an important role in protecting users from a wide range of intimidation, they are nearly helpless against web based client-side attacks, like the XSS attacks. Noxes provides an additional layer of protection that existing personal firewalls do not support. The main idea is to allow the user to endeavor control over the connections that the browser is making, just as personal firewalls allow a user to control the internet connections received by or originating from processes running on the local machine [1]. Therefore it would be safe to say Noxes 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 and can either be blocked or allowed based on the current security policy. Comparable to personal firewalls, Noxes allows the user to create filter rules (i.e., firewall rules) for web requests.

Philipp Vogt et al, have proposed a solution that uses dynamic data tainting. The goal is to ensure that a JavaScript program can send sensitive information only to the site from which it was loaded. To this end, the information flow of sensitive data is tracked inside the JavaScript engine of the browser. Whenever an attempt to relay such information to a third party (i.e., the adversary) is detected, the user is warned and given the possibility to stop the transfer [2]. The authors have done:

- A dynamic taint analysis together with a integral static analysis can prevent XSS attacks by observation of the flow of sensitive information in the web browser.
- The assimilation of the analyses into the popular Firefox web browser.
- The development of a Firefox-based web crawler capable of simulating user actions. This can allow us to perform a large-scale empirical validation of their techniques based on the automatic browsing of more than one million web pages.

Adam Kiezun et al, have proposed an automatic technique for creating inputs that expose SQLI and XSS vulnerabilities. The technique while generating sample inputs, symbolically track taints through execution (also those by database accesses), at the same time mutating the inputs to produce concrete exploits. This is the first analysis of precisely addresses second-order XSS attacks [3].

Table 1: Comparisons of NOXES, NoMoXSS and ARDILLA

Techniques	**NOXES**	**NoMoXSS**	**Ardilla**
Symbolic Executer	C#	phpBB, myBB, webCal	Appolo
Platform	.NET	PHP	PHP
Focus on	XSS, Advanced XSS	XSS	SQLI, XSS1, XSS2
Detect/Exploit	Exploit	Exploit	Exploit
Detection method	Personal firewall	Dynamic data tainting	Taint propagation

Table 1 represents the comparisons of NOXES, NoMoXSS and ARDILLA based on different criteria.

Noxes is the first client side prevention technique. But its main drawback is it just bothers about sensitive data on web application; it doesn't take care about JavaScript code if it contains any malicious content. NOMOXSS has additional protection layer when surfing the web without depending on the security of the web application. But same as NOXES it also doesn't take care about JavaScript code if it contains any malicious content [17] [2]. The third technique discussed above is ARDILLA tool has advantage that it is the first analysis of precisely addressed second order XSS [3]. It has disadvantages like Input generations cannot simulate sessions and If any improvement in input generation is likely to improve ARDILLA's effectiveness.

III. Proposed Solution

Many web applications are built on top of frameworks that APIs providing extensive defense mechanisms against everyday attacks such as cross-site scripting (XSS) and cross-site request forgery (CSRF). The three techniques discussed in literature survey prevent sensitive data from being corrupted by XSS attack. This technique cannot prevent invasion by malicious JavaScript. JavaScript is used as a vehicle to infect websites because it's a programming language that underpins today's web. It's primarily used in the form of client-side JavaScript, implemented as part of a web browser in order to provide enhanced user interfaces and dynamic websites.

Figure 2 presents an abstract description of how HTML input coming from path A flows through a web browser as it is parsed and interpreted. Whenever HTML page is processed the HTML lexer and parser generates HTML parse tree. Then HTML parse tree is given as input to Document generator that generates JavaScript code. This JS code goes to JavaScript lexer and parser and generates JS parse tree. This tree goes to JS runtime environment to generate instructions of HTML page that comes through the entire path to perform specific task and goes to DOM API. This way HTML page is accessed and

most of the HTML page contains JavaScript. Attacker can add malicious code in JavaScript and can inject the Javascript by putting malicious code in comment. Malicious JavaScript generates high-traffic and redirect users to malicious web pages without the victim's knowledge. This process starts initiating vulnerabilities and when people visit these malicious sites, further scripts exploit client-side vulnerabilities. To mitigate this vulnerability we have proposed a solution to eliminate dependence on the browser's parser for building unpatriotic HTML parse trees. The proposed solution generates and encapsulates a model for each comment in the HTML page which can contain malicious code. So whenever we come across a malicious piece of code the application's output is converted into new alternative by replacing each of unpatriotic HTML with its corresponding model and not making any changes to the trusted content.

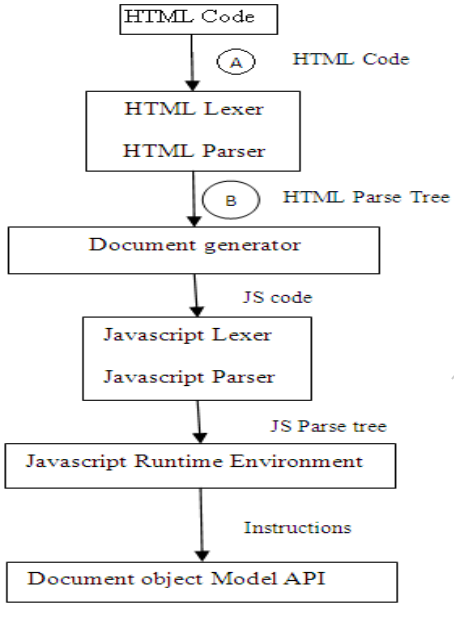

Figure 2: Generalized HTML interpretation process

In our approach, we build the parse tree at the first level only. The parse tree is generated for unpatriotic content programmatically using a small set of low-level Document Object Model (DOM) primitives that are well documented [19] and supported on all JavaScript-enabled browsers. In this solution we have modeled web application output along with a short, trusted script that calls upon the client side JavaScript library, which decodes and safely reconstructs the parse tree within the browser.

IV. Conclusion

XSS is one of the most common types of attack on web services. XSS attacks can be carried out very easily and the attacks are generally simple, but preventing them is very difficult because HTML is very flexible encoding schemes which provides to the attacker capabilities for circumventing server-side input filters. Many different approaches have been proposed to overcome XSS attacks.

The different techniques for prevention of XSS, Noxes, NoMoXSS and ARDILLA, have been studied. Noxes operates as a web proxy that fetches HTTP requests on behalf of the user's browser. NoMoXSS is a solution that uses dynamic data tainting. ARDILLA is an automatic technique for creating inputs that expose SQLI and XSS vulnerabilities. Based on the comparative study these techniques cannot prevent invasion by malicious JavaScript. To mitigate this problem we have proposed a solution that eliminates the use of HTML parse tree. This proposed solution automatically generates and embeds an alternative code for each comment in the HTML page which has malicious code. In the proposed solution all the comments or instance of malicious code is replaced by model which contains trusted code.

References

[1] E. Kirda, C. Kruegel, G. Vigna, and N. Jovanovic, "Noxes: A client-side solution for mitigating cross-site scripting attacks," in 21st Annual ACM Symposium on Applied Computing, Dijon, France, Apr. 2006.

[2] P. Vogt, F. Nentwich, N. Jovanovic, E. Kirda, C. Kruegel, and G. Vigna, "Cross-site scripting prevention with dynamic data tainting and static analysis," in 14th Annual Network & Distributed System Security Symposium, San Diego, CA, USA, Feb. 2007.

[3] Adam Kiezun, Philip J. Guo, Karthick Jayaraman, Michael D. Ernst, "Automatic creation of SQL Injection and Cross site scripting Attacks" in 31st International Conference on Software Engineering IEEE computer society Washington, DC, USA 2009.

[4] S. Artzi, A. Kie˙zun, J. Dolby, F. Tip, D. Dig, A. Paradkar, and M. Ernst. Finding bugs in dynamic Web applications. In ISSTA, 2008.

[5] D. Balzarotti, M. Cova, V. Felmetsger, N. Jovanovic, E. Kirda, C. Kruegel, and G. Vigna. Saner: Composing static and dynamic analysis to validate sanitization in Web applications. In S&P, 2008.

[6] W. Halfond, A. Orso, and P. Manolios. WASP: Protecting Web applications using positive tainting and syntax-aware evaluation. IEEE TSE, 34(1):65, 2008.

[7] Victor Mehai Christiansenn SecPoint - Best IT Security products [online]. Available: http://www.secpoint.com/what-is-cross-site-scripting.html

[8] W. G. Halfond and A. Orso. AMNESIA: Analysis and Monitoring for NEutralizing SQL-Injection Attacks. In ASE, 2005.

[9] Y.-W. Huang, F. Yu, C. Hang, C.-H. Tsai, D.-T. Lee, and S.- Y. Kuo. Securing Web application code by static analysis and runtime protection. In WWW, 2004.

[10] M. Lam, M. Martin, B. Livshits, and J. Whaley. Securing Web applications with static and dynamic information flow tracking. In PEPM, 2008.

[11] D. Balzarotti, M. Cova, V. Felmetsger, N. Jovanovic, E. Kirda, C. Kruegel, and G. Vigna. Saner: Composing static and dynamic analysis to validate sanitization in Web applications. In S&P, 2008.

[12] M. Emmi, R. Majumdar, and K. Sen. Dynamic test input generation for database applications. In ISSTA, 2007.

[13] M. Martin and M. Lam. Automatic generation of XSS and SQL injection attacks with goal-directed model checking. In USENIX Security, 2008.

[14] J. Newsome and D. Song. Dynamic taint analysis for automatic detection, analysis, and signature generation of exploits on commodity software. In NDSS, 2005.

[15] G.Wassermann, D. Yu, A. Chander, D. Dhurjati, H. Inamura, and Z. Su. Dynamic test input generation for Web applications. In ISSTA, 2008.

[16] S. Cook. A web developer's guide to cross-site scripting. Technical report, SANS Institute, 2003.

[17] Y.W. Huang, S.K. Huang, T.P. Lin, and C.H. Tsai. Web application security assessment by fault injection and behavior monitoring. In: Proceedings of the 12th International World Wide Web Conference (WWW 2003), May 2003.

[18] Y.W. Huang, S.K. Huang, T.P. Lin, and C.H. Tsai, F. Yu, C. Hang, D. Lee, and S.Y. Kuo. Securing Web Application Code by Static Analysis and Runtime Protection. In: Proceedings of the 13th International World Wide Web Conference (WWW 2004) May 2004.

[19] World Wide Web Consortium, "Document object model (DOM) level 2 core specification," Nov. 2000. [Online]. Available: http://www.w3.org/TR/DOM-Level-2-Core/