Dynamic Removal Of Cross Site Scripting Vulnerabilities In Web Application

P. Umasankari, E. Uma, A. K Kannan

Abstract - Cross Site Scripting (XSS) Vulnerability is the top web application vulnerabilities according to current surveys. This vulnerability occurs when a web application uses inputs received web pages without properly validating them. This allows an attacker to inject malicious script when a client visits tapped web pages. Such an attack may cause serious security violations such as account hijacking and cookies theft. A current approach to mitigate these problems mainly focuses on effective detection of XSS vulnerabilities in the programs or prevention of XSS attacks. As more convoluted attack vectors are being detected. Vulnerabilities if not removed could be tapped anytime. In this proposed system to address this issue and presents an approach for removing XSS vulnerabilities in web application.

Keywords - Cross Site Scripting, Web security, Injection vulnerability, automated bug fixing, Encoding and removal.

I. INTRODUCTION

Recent reports about web applications to reveal that cross site scripting (XSS) is one of the most common and severe web security defects. It is a type of code injection vulnerability that enables attackers to send venomous scripts to the web clients. It occurs when the web application references the user input in its HTML pages without properly validating the web pages. An attacker may inject the malicious scripts via such inputs in the web application’s HTML pages. When a client visits a tapped web page, the client’s browser not being aware of the presence of malicious scripts shall execute all scripts sent by application resulting in a successful XSS attack. XSS attacks may be the reason for severe security violations. To mitigate the threats posed by the XSS attacks, several solutions have been proposed. They can be classified into defensive coding practices, input validation and XSS testing techniques, vulnerability detection techniques and attack prevention techniques. However, these methods if performed manually are prone to human errors and hard to enforce in existing web applications. Therefore, automation of this task would be beneficial. This paper proposes an automated approach that statically removes the XSSVs from the program source code. The proposed approach consists of two methods: (1) XSSV Detection and XSSV Removal. XSSV detection method identifies the potential XSSVs in the program source code using static analysis and pattern matching techniques. XSSV removal method identifies the HTML context of each user input referenced in the potential XSSV.

It then secures the potential XSSVs by applying the appropriate escaping methods using escaping library provided by ESAPI. Results show that the approach was effective in securing all the XSSVs found in the subjects by using encoding facilities.

II. RELATED WORK

Based on the way, the XSS threat is mitigated; related approaches can be classified into three types- input validation and its testing, vulnerability detection and attack prevention. These existing approaches focus on finding vulnerabilities present in the applications or preventing XSS attacks with runtime monitors. By contrast, our approach focuses on removing XSSVs by using escaping mechanisms that prevent the special characters contained in the user input from invoking client-side interpreters.

A. Input validation and XSS testing

Jane Huffman Hayes and Jeff Offutt [1] addressed the problem of statically analyzing input command syntax as defined in interface and requirements specifications and then generating test cases for dynamic input validation testing. In this effort the IVAT (Input Validation Analyzing and Testing) techniques to be used to validate the inputs and generate the test cases. Input validation analyzed by the Errors and generates the Test Cases. Gary Wassermann, et al., [4] has been proposed an automated input test generation algorithm that uses runtime values to analyze dynamic code, models the semantics of string operations, and handles operations whose argument and return values may not share a common type. As in the standard concolic testing framework, algorithm gathers constraints during symbolic execution.

B. Vulnerability Detection

These approaches are mainly based on static analysis techniques. Static approaches could in prove the absence of vulnerabilities. However, as they tend to generate many false alerts, later approaches incorporate dynamic analysis techniques to improve the accuracy. It is based on the static or dynamic analysis and prevention techniques.

1) Static Analysis

Dynamic Removal Of Cross Site Scripting Vulnerabilities In Web Application 37
Adam Kiezun, et al.[5] has been proposed an automatic Static analysis technique for creating inputs that expose SQLI and XSS vulnerabilities. The technique generates consumed inputs, symbolically tracks taints through execution (including usage of database accesses), and changes the inputs to produce concrete exploits. Analysis of which they are aware that precisely addresses XSS attacks. Their technique creates real attack transmitters, has few false positives, incurs no runtime overhead for the positioned application, works without requiring modification of application’s source code, and handles dynamic programming-language constructs.

2) Dynamic Taint Analysis

US-CERT, OWASP (Open Web Application Security Project) proposed the Detection of XSS vulnerabilities and its detailed description and way of possibilities of XSS attacks are described [12] [13]. It describes the possibilities of XSS attacks and contaminated encoding mechanism and escaping mechanism are described and implemented in script level project by deriving the same constraint rules and using encoding rules. A challenge for these security mechanisms is enabling web applications to accept complex input from users, while disallowing poisonous script content.

C. Attack Prevention

These techniques use the automatic monitoring systems, which positioned on either client side or server side, to prevent the actual time XSS attacks. William Robertson and Giovanni Vigna proposed a different approach to web application security [7]. In this work, they present a web application framework that leverages existing work on strong type systems to statically enforce a separation between the structure and content of both web documents and database queries generated by a web application, and displays, how this approach can automatically prevent the introduction of both server-side cross-site scripting and SQL injection vulnerabilities.

III. PROPOSED APPROACH

The proposed approach consists of two major parts: XSS vulnerability detection and XSS vulnerability removal. The overall system architecture is represented by figure1. The first part identifies the potential XSS vulnerabilities in server side programs. The second part first identifies the code locations where the malicious data can be escaped, second determines the required escaping mechanisms using the ESAPI’s API. Our approach follows the OWASP’s XSS prevention rules [12]. The proposed approach provides two options to the user in order to secure the implicated statement: (1) Lenient Mode-it requests the user to input an appropriate sanitization method (2) Strict mode- it unconditionally removes the untrusted data from the code location it is referred. Therefore XSSV removing mechanism is effective and complete in terms of removing all XSS vulnerability in the client and server programs. In the following we provide the details of the proposed approach.

![Figure 1: Overall System architecture](image-url)
Based on the above definitions, we have implemented the identification of pv-outs by tracking the flow of untrusted data between input data and HTML output data. Note that the term “potentially vulnerable” is used because the HTML output statement may not be actually vulnerable to XSS attacks if the untrusted data referenced is not controlled by external users.

B. XSS vulnerability removal

This part contains two major steps-HTML context discoveries and secures source code replacement. The first step identifies the statements at which the untrusted data referenced in an HTML output statement can be escaped without compromising intended HTML outputs and security aspects. Then it extracts the HTML document structure surrounding each untrusted data from the source code and identifies the HTML context using pattern matching. The second step generates secure code structures using ESAPI’s escaping APIs as replacements for original code structures. The methods are based on data flow analysis and pattern matching techniques.

1) HTML context discovery

When a HTML output statement is identified as vulnerable, the untrusted data referenced in that statement must be escaped according to the HTML context the referenced data is in. However, in some scenarios, escaping should not be done in the vulnerable statement itself because the variable containing the untrusted data may also contain programmer-defined HTML document structures. Therefore, for each potential vulnerable output, the method finds the statements at which the untrusted data can be properly escaped. Then the method identifies the HTML context by analyzing the HTML document structure surrounding the escaping statements.

2) Secure Source code replacement

As an input from the previous section, this method receives the information of escape statements, to_be_escaped’s escaping rules and corresponding escaping methods. Firstly, it declares the required ESAPI packages for class declaration of the input program. For each escape_stmt in each potential vulnerable output o, the method wraps the untrusted data referenced in escape_stmt with appropriate escaping APIs using the following steps:

(1) Identify the appropriate escaping API escape_api from EAPI library that corresponds to the required escaping mechanism of the variable or the method marked as to_be_escaped in the escape_stmt.

(2) Modified escape_stmt by wrapping the object marked as to_be_escaped with escape_api.

(3) Remove the original statement and insert the modified statement into the same code location.

Hence, removal of all XSSVs from input programs is fully automated. And code modification required is very minimal because only objects containing untrusted data are wrapped with escaping API calls.

IV. EVALUATION AND RESULTS

We developed a mechanism to implement the proposed approach. Using the mechanism, we evaluated the proposed approach on web applications.

![Table 1: Test Subjects](image)

This proposed approach consists of three modules: Program Analyzer XSSV Detection, and XSSV Removal. The prototype mechanism was developed through the use of program analysis by manual shows the architecture of proposed approach in Fig. 1. XSSV Detection includes two major modules: data trace and identifier. Two modules combined together to implement the XSSV detection part discussed in section 3.1. The XSSV Remover consists of two modules: context finder and code wrapper. These two modules implement the XSSV removal as discussed in section 3.2. Code wrapper provides a user interface for user to set lenient or strict mode. As a final step user has to install ESAPI into the application and recompile the modified programs.

Table 1 shows the Real XSS vulnerabilities found in the test subjects and potential XSS vulnerabilities reported by proposed mechanism, which correspond to the each HTML context type.

CONCLUSION

In this paper we present two methods for finding and removing potential XSSVs in server and client...
programs. The first method adopts the static analysis approach to track the flow of user inputs into HTML output statements and identifies the potentially vulnerable statements. The second method uses pattern matching and data dependency analysis to identify the HTML contexts in which the user inputs are referenced and the required escaping mechanisms that prevent code injection. Then it performs source code generation and replacement to source potentially vulnerable statements with proper escaping APIs. As more and more sophisticated attack patterns are discovered, vulnerabilities if not removed could be exploited anytime. We also proposed approach is fully focused on removing XSSVs with minimum user intervention. We also developed the mechanism that automated the proposed approach. In our evaluation, the mechanism was successful in removing all the real XSSVs found in the tested subjects.

In future work, we intend to enhance both current approach and mechanism (1) add client side script analysis script in our proposed approach to prevent DOM-based XSS; (2) develop an analysis technique that tracks the flow of user inputs stored into persistent data structures and across the web pages to accurately detect the influence of user input in HTML output statements; Furthermore, we would like to explore the applicability of proposed approach in other security issues by making full use of ESAPI’s capabilities.

REFERENCES

(3) H.Liu, H.B.K.Tan, Covering code behavior on input validation in functional testing, Information security and software technology 51(2). (2009). PP 546-553
(8) M.Martin, M.S.Lam, Automatic generation of XSS and SQL injection attacks with goal-directed model checking, in : Proceeding of the 17th USENIX security Conference (USENIX Security’09),2009 pp.31-43.
(9) Mike Ter Louw, V.N Venkatakrishnan Blueprint: Robust Prevention Of Cross-Site Scripting Attacks For Existing Browsers in proceedings of the 5th International Conference on Detection of Intrusions and Malware and Vulnerability Assessment (DIMVA’08),2008. PP. 23-43.

★★★