An Effective Technique of Detecting & Preventing Web Vulnerabilities

Swarnaprabha Patil, Nitin Agrawal

Abstract— Here in this paper a new and efficient technique for the detection and prevention of security in web is implemented here. Since Web contains a number of sql injection attacks through which information can be fetched. Hence a new technique is implemented which provides successful attacks detection and prevention using IDS. The experimental results show the performance of the proposed methodology.

Index Terms— Web Security, Vulnerability, Attack Injection, SQL Injection, False Positive, True Positive.

1 INTRODUCTION

The web is our primary gateway to many critical services and offers a powerful platform for emerging applications.

As the underlying execution platform for web applications grows in importance, its security has become a major concern. Web application vulnerabilities have become pervasive in web applications today, yet techniques for finding and defending against them are limited. With such a e-commerce transactions occurring, the security of those web transactions is a major concern to any Internet user. To complicate matters, the introduction of smart mobile devices and tablet computers is also having an impact on e-commerce and web transactions. In recent years there has been extensive research conducted related to e-commerce and web security; however, the problem of securing web transactions still exists. The problems with e-commerce and web security are often related to Structured Query Language (SQL) injection, cross-site scripting (XSS), cookie manipulation, and Uniform Resource Locator (URL) redirection [1, 4]. We tackle the problem of developing techniques to automatically find and prevent script injection or scripting vulnerabilities-a class of web vulnerabilities permissive in web applications today. Web applications are a fundamental part of our lives and culture. We use web applications in almost every facet of society: socializing, banking, health care, taxes, education, news, and entertainment, to name a few. These web applications are always available from anywhere with an Internet connection, and they enable us to communicate and collaborate at a speed that was unthinkable just a few decades ago by web languages.

Web languages, such as HTML, have evolved from lightweight mechanisms for static data markup to full-blown vehicles for supporting dynamic execution of web application logic. HTML allows inline constructs both to embed entrusted data and to invoke code in higher-order languages such as JavaScript. Web applications often embed data controlled by entrusted adversary's inline within the HTML code of the web application. For example, a blogging application often embeds entrusted user comments inline within the HTML content of the blog. HTML and other web languages lack principled mechanisms to separate trusted code from inline data and to further isolate entrusted data such as user-generated content from trusted application data. Script injection vulnerabilities arise when entrusted data controlled by an adversary is interpreted by the web browser as trusted application i.e. script code. This causes an attacker to gain higher privileges than intended by the web application, typically granting entrusted data the same authority as the web application's code. Well-known example categories of such attacks are cross-site scripting i.e. XSS [5] and cross-channel scripting i.e. or XCS [2] attacks. Scripting vulnerabilities are highly pervasive and have been recognized as a prominent category of computer security vulnerabilities. Software errors that result in script injection attacks are presently rated as the fourth most serious of software errors in the CWE's Top 25 list for the year 2011 [3]. OWASP's Top 10 vulnerabilities ranks scripting attacks as the second most dangerous of web vulnerabilities in 2010 [4] Web Application Security Consortium's XSS vulnerability report shows that over 30% of the web sites analyzed in 2007 were vulnerable to XSS attacks [6]. In addition, there exist so many publicly available repositories of real-world XSS vulnerabilities.

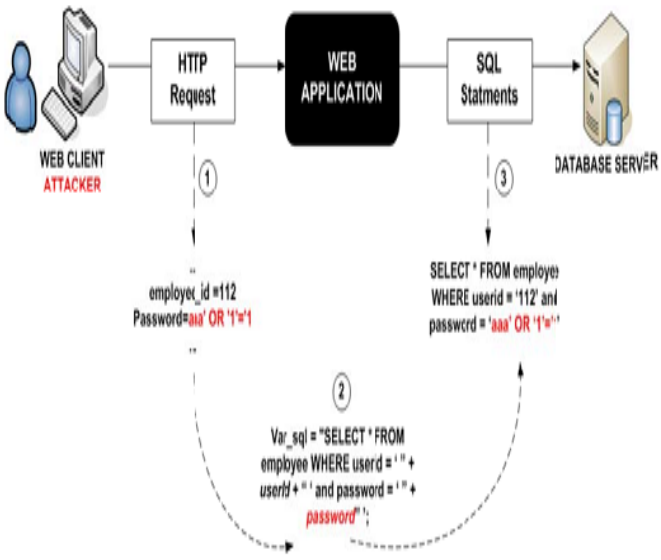

Figure 1. Example of a SQL Injection Attack

Server-side Script Injection Vulnerabilities

Script injection attacks in server-side applications have been investigated in depth by prior work. We provide an example of a typical scripting attack for exposition. All the HTTP data inputs to

the web application server are treated as entrusted data. In this application, the security policy forbids entrusted data to be executed as scripts or HTML markup when processed by the web browser. Script injection vulnerability is one that allows injection of entrusted data into a victim web page which is subsequently interpreted in a malicious way by the browser on behalf of the victim web site.

Client-side Script Injection Vulnerabilities

Much prior vulnerability research has focused primarily on the server-side components of web applications. Scripting vulnerabilities can arise in client-side components, such as those written in JavaScript, as well [7]. Here we present examples of client-side script injection vulnerabilities, a subclass of scripting vulnerabilities which result from bugs in the client-side code. In client-side script injection vulnerability, critical sinks are operations in the client-side code where data is used with special privilege, such as in a code evaluation construct.

SQL Injection

SQL injection vulnerabilities, while declining in the number reported compared to XSS vulnerabilities are still numerous and are incredibly critical when they occur.

```
$name = $_GET['name'];
$q = "select * from users where name = '" . $name . "';";
$result = mysql_query($q);
```

2 Literature Survey

The approach of Louw et al. [8] is the closest to our work. They develop the "Blueprint" tool to mitigate XSS attacks that first generates response pages without any JavaScript node at the server-side. To eradicated script is accomplished at the browser side based on the content generation provided by the server-side with code instrumentation. Consequently, Blueprint has to rely on a particularly intended external JavaScript library at the client-side. Absolutely not, but their approach depends on compassionate HTML and JavaScript features and removes suspected malicious contents from the server-side. Furthermore their come within reach of does not impose any external library dependency. Blueprint transforms untrusted contents (e.g., an attacker supplied inputs) when sending them to the browser. On the contrary, their approach transforms the server-side script code based on the possible injection places, but not the injected contents themselves.

In this paper author et al. Jose Fonseca [9] proposes a methodology to automatically inject realistic attacks in web applications and a prototype tool to evaluate web application security mechanisms. The proposed work analyzing the web application and generating a set of potential vulnerabilities on the idea that injecting realistic vulnerabilities in a web application and attacking them automatically can be used to support the assessment of existing security mechanisms and tools to find each vulnerability in custom setup scenarios is injected and various attacks are mounted over each one is automatically assessed and reported.

In this paper author has proposed [10] mechanism for assessing Web application security was constructed to analyze the design of Web application security assessment mechanisms in order to identify poor coding practices that render Web applications vulnerable to attacks such as SQL injection and cross-site scripting. Proposed technique come across on the real-world circumstances of web application security estimation of intended a crawler crossing point that incorporates and mimics Web browser functions with the intention of analysis Web applications.

Here et al. Halfond, W. G [11] present an extensive perform this evaluation, they first identified the various types of SQLIAs known to date. They then evaluated the considered techniques in terms of their capability to detect and/or prevent such type of attacks. For each type of attack, they make available explanations and examples of how attacks of that type could be performed.

Here author et al. Manisha A. Bhagat [12] paper proposed alerts the people who are related to database maintenance, DBA and other people who are introducing their sites on Internet. This paper gives idea on the subject of the hole which can be secured either by code or protection security like firewalls. It is essential to check the code before commencing the site. SQL Injection Attacks are dangerous to the applications on Internet.

In this paper [13] author has presented an enhanced Tainted Mode model that incorporates data flows which allows inter module vulnerabilities detection. They also commenced a new approach to involuntary penetration testing by leveraging it with knowledge from dynamic analysis. Convectional approaches based on the Tainted Mode vulnerability model which cannot hold inter-module vulnerabilities. So the author will work focus on two main approaches.

In this paper [14] authors presents an authentication scheme for preventing SQL Injection attack using Advance Encryption Standard (PSQLIA-AES). Here this scheme required encrypted user name and password are to improve the authentication process with minimum overhead. The server has to sustain three parameters of every user: user name, password, and user's secret key.

In this paper [15] author presents a new methodology by developing a framework that makes attacks on networked servers and discovers security vulnerabilities in software systems and allows security administrators to determine the problems. To show this model application is built that a new methodology that takes protocol requirement aspects from server and carry out various attacks on the server and discovers vulnerabilities.

In this paper [16], they present a detailed review on various types of Structured Query Language Injection attacks, Cross Site Scripting Attack, vulnerabilities, and prevention methods. From the survey of various papers it is found that SQL

Injection and Cross-site Scripting (XSS) Attacks are most powerful and easiest attack methods on the Web Application. This research presents a review of various current methods for protecting against SQL injection and XSS make use of it.

Fonseca, J. CISUC [17] proposed a methodology to inject realistic attacks in Web applications. The methodology is based on the idea that by injecting realistic vulnerabilities in a Web application and attacking them automatically we can assess existing security mechanisms. To provide true to life results, this methodology relies on field studies of a large number of vulnerabilities in Web applications. The paper also describes a set of tools implementing the proposed methodology. They allow the automation of the entire process, including gathering results and analysis.

3 Proposed Methodology

A novel method to detect SQL injection attacks based on static and dynamic analysis. This method removes the attribute values of SQL queries at runtime (dynamic method) and compares them with the SQL queries analyzed in advance (static method). Applying Table 1 to the example in the following:

If : admin, 1234, 1' OR '1=1'-- and 1234.

FQ : SELECT * FROM user WHERE id='$id' AND password='$password'.

DQt : SELECT * FROM user WHERE id='admin' AND password='1234',

DQf : SELECT * FROM user WHERE id='1' or '1=1'—' AND password='1111'.

Symbol	Description
It,f	User input data{t:normal input data, f :abnormal input data}
F	Function which drops the value of the SQL query
FQ	Fixed SQL query in web application
DQt,f	Generated dynamic SQL query with user input {t:normalSQLquery,f :abnormal SQL query}
FDQ	Attribute indicating which value was removed from the fixed SQL query
DDQt,f	Attribute indicating which value was removed from the dynamic SQL query{t:normalSQLQuery,f :abnormal SQL Query}

The detection method proposed in this article uses the function f which deletes the attribute values in the SQL queries. The function is shown in formula

FDQ=f (FQ), DDQ=f (DQ)……(1)

In algorithm 1, the function, f , removes only the string values surrounded by ' after ''=" or within parenthesis. The attribute value of an SQL query consists either of the form variable = 'string value' or variable = numeric value. In case that a function is used in an SQL query, the function head is either of the form "function name (numeric value)" or "function name ('string value')". The value of the string is surrounded by '. The 'which surrounds the string value is the operator' but the value of ' in the SQL query is preceded by \. So the case where ' is preceded by \ is not considered. The function of Get_Token in the algorithm extracts and removes the first character in the input string and then returns the character. Current_Quotation_State is changed to the appropriate status in the function Toggle Current_Quotation_Statein this algorithm.

Algorithm f(One SQL query)

```
Enumerate Quotation_Status = { Quot_Start, Quot_End}
Input String=One SQL query;
;Output_String=Null;
Current_Quotation_State=Quot_End;
Do while( not empty of Input String)
{
Char=Get_Token(Input_String);
If Char is a quotation character
{
Add Char to Output_String;
If the preceding character is not back slash
Toggle Current_Quotation_State;
}
Else
{
If Current_Status is Quota_End than
{
Add Char to Output_String;
}
Else
{
If the preceding character is \ (back slash) then
Add Char to Output_String;
}
}
}
Return Output_String;
```

Algorithm 1: Algorithm which removes the attribute value in a SQL query.

The following examples show the result of function f . Bold characters are deleted and "consists of two concatenated '.

DQ1 is a normal query and DQ2 is an abnormal query.

FQ = SELECT * FROM user WHERE id='$id' AND password='$password'

FDQ=f(DQ)= f(SELECT * FROM user WHERE id=' $id' AND password='$password') = SELECT * FROM user WHERE id=' ' AND password=' '

DQ1= SELECT * FROM user WHERE id='admin' AND \ password='1234'

DDQ1=f(DQ1) = f(SELECT * FROM user WHERE id='admin'

AND password='1234')= SELECT * FROM user WHERE id=' ' AND password=' 'DQ2= SELECT * FROM user WHERE id='1' or '1=1'—' AND password='1234'

DDQ2=f(DQ2) = f(SELECT * FROM user WHERE id='1 ' or' 1=1'—' AND password='1234')= SELECT * FROM user WHERE id=' ' or ' '—''1234'

Algorithm 2 is the generalization of the SQL injection attack detection algorithms proposed in this section. Lines 1–4 of this algorithm can be processed for the targeted web pages in advance.

N: Total number of fixed SQL queries in web application
FQi: i 'th fixed SQL query in web application
DQi: Dynamic SQL query generated from FQi
f : Function to delete value of attribute in SQL query
FQ = {FQ1, . . . , FQn},
FDQ = {FDQ1, . . . , FDQn},
// Static analysis
1. For i=1 to N
2. Get FQi
3. FDQi = f (FQi)
4. End {For}
5.// Dynamic analysis (running time)
6. While(Normal & k & N)
7. Get DQk from the web with I{t,f }
8. DDQk = f (DQk)
9. If(FDQk & DDQk) = 0 then
10. Result = Normal
11. Else
12. Result = Abnormal
13. End {If}
14. End {While}

4 Result Analysis

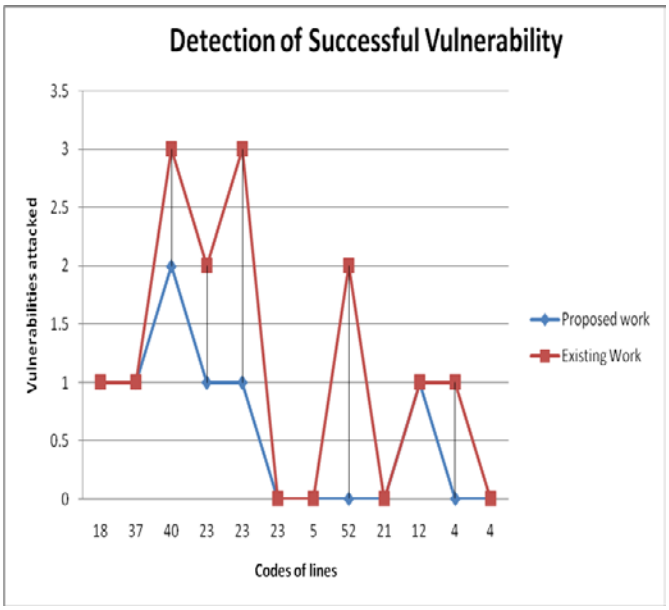

Figure 5.1 Comparison of Detection of Successful Vulnerabilities

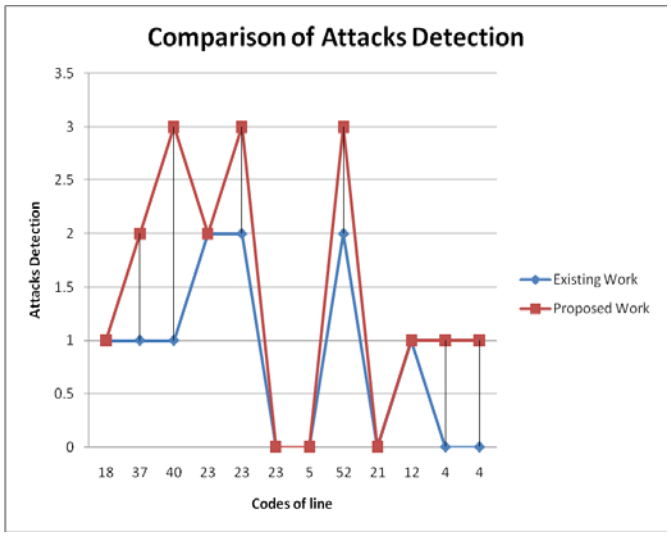

Figure 5.2 Comparison of Attacks Detection

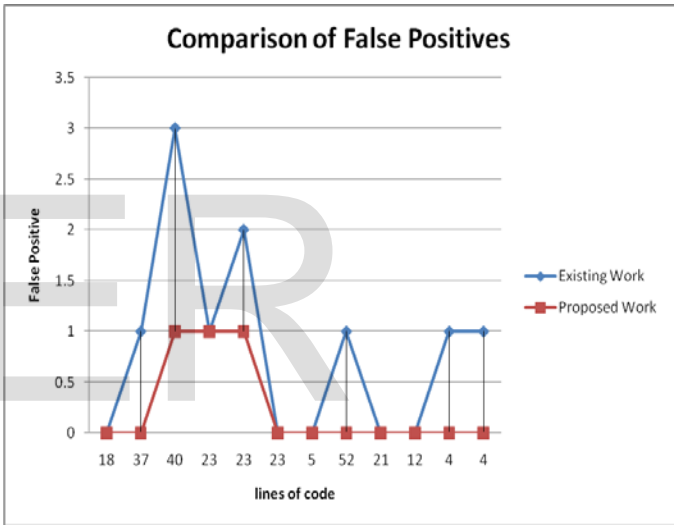

Figure 5.3 Comparisons of False Positives

4 Conclusion

Attacks to these vulnerabilities basically take advantage of improper coded applications due to unchecked input fields at user interface. This allows the attacker to change the SQL commands that are sent to the database (SQLi) or through the input of HTML and scripting languages (XSS). The use of fault injection techniques to assess security is actually a particular case of software fault injection, focused on software faults that represent security vulnerabilities or may cause the system to fail in avoiding a security attack. We are trying to automatically discover places in the web application code that can be used to inject vulnerabilities using fault injection techniques and smart fuzzing to seamlessly attack them. The proposed methodology implemented here for the security vulnerabilities from various attacks is efficient as compared to the existing technique.

REFERENCES

[1] W. D. Thomas Ian and D.Weidenhamer, "Quarterly Retail E-Commerce Sales 1st Quarter 2012," U.S. Census Bureau, Washington, DC, USA, Newsletter CB12-78, May 2012.

[2] Hristo Bojinov, Elie Bursztein, and Dan Boneh. "XCS: Cross Channel Scripting and its Impact on Web Applications" In: CCS. 2009.

[3] CWE. "2011 CWE/SANS Top 25 Most Dangerous Software Errors". http://cwe.mitre.org/top25/. 2011.

[4] OWASP. OWASP Top 10 - 2010, The Ten Most Critical Web Application Security Risks.Presentation https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project.

[5] RSnake. XSS Cheat Sheet for filter evasion. http://ha.ckers.org/xss.html.

[6] Web Application Security Consortium. "Web Application Security Statistics Project 2007". http://www.webappsec.org/projects/statistics/wasc_wass_2007.pdf.

[7] Amit Klein. DOM Based Cross Site Scripting or XSS of the Third Kind. Tech. rep.Web Application Security Consortium, 2005

[8] M. Louw and V. Venkatakrishnan, "BLUEPRINT: Robust Prevention of Cross-Site Scripting Attacks of Existing Browsers," IEEE Security and Privacy, Oakland, California, USA, May 2009, pp. 331-346.

[9] José Fonseca, Marco Vieira, and Henrique Madeira "Evaluation of Web Security Mechanisms using Vulnerability & Attack Injection", pp. 1- 12, 2013.

[10] Huang, Yao-Wen, Shih-Kun Huang, Tsung-Po Lin, and Chung-Hung Tsai. "Web application security assessment by fault injection and behavior monitoring." In Proceedings of the 12th international conference on World Wide Web, pp. 148-159. ACM, 2003.

[11] Halfond, W. G., Jeremy Viegas, and Alessandro Orso "A classification of SQL-injection attacks and countermeasures", In Proceedings of the IEEE International Symposium on Secure Software Engineering, Arlington, VA, USA, pp. 13-15. 2006.

[12] Manisha A. Bhagat and Vanita Mane "Protection Of Web Application Against Sql Injection Attack", International Journal of Scientific and Research Publications, ISSN 2250-3153, Volume 3, Issue 10, October 2013.

[13] Petukhov, Andrey, and Dmitry Kozlov "Detecting security vulnerabilities in web applications using dynamic analysis with penetration testing." Computing Systems Lab, Department of Computer Science, Moscow State University (2008).

[14] Indrani Balasundaram and E. Ramaraj "An Authentication Mechanism to prevent SQL Injection Attacks", International Journal of Computer Applications, ISSN: 0975 – 8887, Volume 19, No. 1, April 2011.

[15] Dr. B. Raveendranath Singh "Vulnerability Discovery with Attack Injection Software Vulnerability Discovery", International Journal of Advanced Research in Computer Science and Software Engineering, ISSN: 2277 128X, Volume 3, Issue 9, September 2013.

[16] Rahul Johari and Pankaj Sharma "A Survey On Web Application Vulnerabilities (SQLIA,XSS)Exploitation and Security Engine for SQL Injection", 2012 International Conference on Communication Systems and Network Technologies, pp. 453 – 458, 2012.

[1] Fonseca, J. CISUC, Univ. of Coimbra, Coimbra, Portugal Vieira, M. and Madeira, H. Vulnerability & attack injection for web applications. Dependable Systems & Networks, 2009 DSN '09. IEEE/IFIP International Conference on, 93-102, 2009.