Testing for Tautology based SQL Injection Attack using Runtime Monitors

Ramya Dharam and Sajjan G. Shiva
Department of Computer Science
University of Memphis
Memphis, TN-38152, USA
{rdharam, sshiva}@memphis.edu

***Abstract* - Today, all commercial and business applications (e-commerce, banking, blogs, web mail, etc.,) are built as web-based database applications. Increasing prominence and usage of these applications has made them more susceptible to attacks because they store huge amount of sensitive user information. Traditional security mechanisms like network firewalls, intrusion detection systems, and use of encryption can protect the network, but cannot mitigate attacks targeted towards web applications. Hence hackers are moving their focus from network to web applications. SQL Injection Attacks (SQLIAs) are one of the most popular and widely performed attacks on web applications. Various security testing techniques exist to detect vulnerabilities in web applications. Vulnerability Scanning performed using vulnerability scanners is one of the widely used security testing techniques, but the rate of false positives and false negatives obtained limit their usage to efficiently detect all the vulnerabilities present in web applications. In this paper, we present and evaluate a security testing technique called Runtime Monitoring to detect and prevent tautology based SQL Injection Attack. For evaluation, we targeted a web application with large number of both legitimate inputs and illegitimate tautology based attack inputs, and measured the performance of the proposed technique.**

Keywords - Security Testing; Runtime Monitoring; Data-flow Testing; Basis-path Testing; Web Applications; SQL Injection Attacks (SQLIAs); Tautology.

I. INTRODUCTION

Web applications have become popular means of modern information retrieval and interaction. This increase in demand and usage of web applications has made them vulnerable to attacks, as they contain confidential user data. In June 2013, the Open Web Application Security Project (OWASP) officially released the Top 10 attacks performed on web applications [1]. Once again, SQLIAs have been ranked as the most widely performed attack on web applications. SC Magazine [2] and ZDNet [3] discuss the recent increase in SQLIAs during the year 2013 and 2012 respectively. Popular websites like Sony [4], LinkedIn [5], Nvidia [6] and Gamigo [7] were hacked using SQLIAs, in June and July 2012, and sensitive information like passwords, account details, etc., about millions of users were leaked online. In spite of various methodologies and testing techniques proposed and implemented by researchers and organizations to handle SQLIAs, they still allow breaches, and hence the security of web applications is not completely accomplished.

A web application is structured as a three-tiered architecture consisting of a web browser, an application server and a back-end database server. Such an application will accept input from external users via forms, dynamically construct the database queries using the inputs provided by the user, dispatch them to the underlying database for execution, and finally retrieve and present the data to the user.

SQLIAs are a class of code injection attacks which are directed towards the database residing at the back-end of the web applications and the attacker tries to gain unauthorized access to the confidential user information residing in the database. SQLIAs occur when the input provided by a malicious user consisting of SQL keywords or operators is not properly validated and is included directly as part of the query. This causes the web application to generate and send a query that in turn results in unintended behavior of the web application causing the loss of confidential user information [8]. Different kinds of SQLIAs known to date are discussed in [9, 10] which include the use of SQL tautologies, illegal queries, union queries, piggy-backed queries, etc.

In this paper, we discuss the Tautology based SQL Injection Attack which is the most simple and popular type of SQLIAs. This attack is performed on web applications by injecting code that consists of SQL tokens into one or more conditional statements so that they always evaluate to true. Bypassing the authentication page in web applications, and extracting data is the most common usage of tautology based SQLIA. In this type of attack, a vulnerable input field that is used in the query's WHERE condition is exploited by the attacker. As the database scans each record in the table, the conditional logic is evaluated and the database returns all the records in the table, if the evaluated conditional logic represents a tautology, and returns true. The attack is successful when the web application either displays all of the returned records or performs some action if at least one record is returned.

For example, if a database contains usernames and passwords, the application may contain code such as the following:

*Query = "SELECT * FROM employeeinfo WHERE name = ' "+ request.getParameter ("name") +" ' AND password = ' "+ request.getParameter ("password") +" ' ";*

This code generates a query intended to be used to authenticate a user who tries to login to a web site. If a malicious user enters " ' OR 1 = 1 -- ' " and " ' ' " instead of a legitimate username and password into their respective fields, the query string becomes as follows:

*SELECT * FROM employeeinfo WHERE name = ' ' OR 1 = 1 -- ' 'AND password = ' ' ' ';*

Any website that uses this code would be vulnerable to tautology based SQLIA. The character "--" indicates the beginning of a comment, and everything following the comment is ignored. The database interprets everything after the WHERE token as a conditional statement, and the inclusion of "OR 1=1" clause turns this conditional into a tautology whose condition always evaluates to true. Thus, when the above query is executed, the user will bypass the authentication logic, and more than one record is returned by the database. As a result, the information about all the users will be displayed by the application and the attack succeeds.

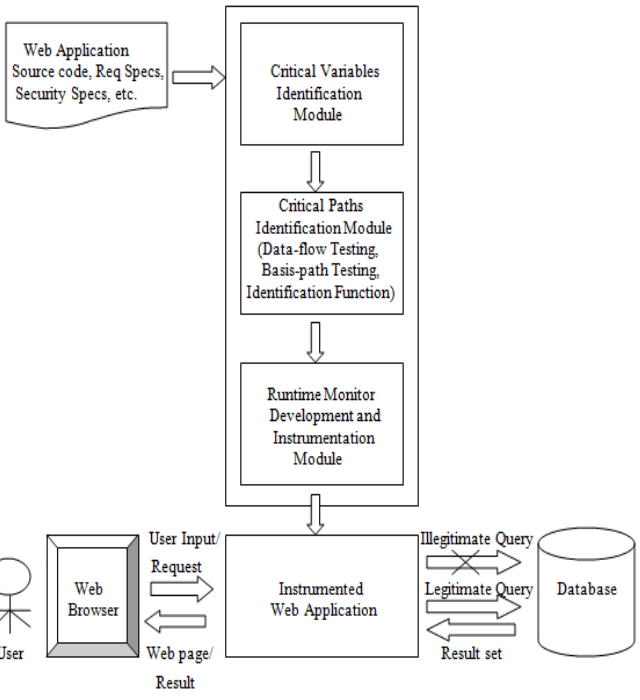
To detect such attacks and other types of security vulnerabilities, security testing is the most widely used testing technique. The overall goal of security testing is to reduce vulnerabilities within software systems. There are two major aspects of security testing: testing security functionality to ensure that it works, and testing the subsystem in light of malicious attack [11]. Different types of security testing techniques and tools are discussed in [12]. The two main approaches to perform security testing on web applications include White Box Testing and Black Box Testing [13]. White Box Testing consists of the source code analysis of web applications to detect vulnerabilities in them, and is performed manually or by using Source Code Analysis Tools. Due to the complexity of the code, exhaustive source code analysis may be difficult and might not be able to find all security vulnerabilities. Black Box Testing consists of analysis of the execution of application to detect vulnerabilities, and this approach is also known as Penetration Testing. It is performed by Black Box Testing Tools also called as Penetration Testing Tools or Vulnerability Scanners. The usage of vulnerability scanners is regarded as an easy way to test the security of web applications, and to detect critical vulnerabilities like SQL Injection Attacks. The rate of false positives (vulnerabilities detected that did not exist) and false negatives (vulnerabilities existed and were not detected) obtained limit their usage to efficiently detect all the vulnerabilities present in web applications.

Due to the limitations of the techniques mentioned above, web applications are still vulnerable to attacks. In this paper, we present and evaluate a security testing technique called Runtime Monitoring to handle tautology based SQL Injection Attack. For evaluation, we targeted an application with a large number of both legitimate inputs and illegitimate tautology based inputs, and measured the performance of the proposed technique. The results obtained were promising and our technique was successfully able to detect and prevent the tautology based SQLIA on web application, without causing any false positive or false negative. Our proposed technique also imposed a low runtime overhead on the subject application.

The paper is organized as follows. In Section 2, we discuss in detail the Runtime Monitoring Technique to handle Tautology based SQLIA. Evaluation and results obtained are discussed in Section 3. Finally, we conclude in Section 4.

II. Runtime Monitoring Technique

Runtime Monitoring involves examining the behavior of the program under test, determining whether this observed behavior is symptomatic of vulnerability in the software [14], and is usually performed by runtime monitors. In this section, we discuss our framework called Runtime Monitoring Framework to develop and integrate runtime monitors into the web application. The instrumented application obtained performs runtime monitoring of the application to detect and prevent tautology based SQLIA. Fig. 1 shows the high-level view of our framework consisting of the following modules: i) Critical Variables Identification Module ii) Critical Paths Identification Module iii) Runtime Monitor Development and Instrumentation Module.



Web Application Source code, Req Specs, Security Specs, etc.
Critical Variables Identification Module
Critical Paths Identification Module (Data-flow Testing, Basis-path Testing, Identification Function)
Runtime Monitor Development and Instrumentation Module
User
Web Browser
User Input/ Request
Web page/ Result
Instrumented Web Application
Illegitimate Query
Legitimate Query
Result set
Database

Fig. 1 High Level View of Runtime Monitoring Framework.

Critical Variables Identification Module: identifies all the critical variables, i.e. variables that are initialized with the input provided by external user and those that become a part of SQL query. Input to this module is a Java web application, and it outputs the critical variables.

Critical Paths Identification Module: identifies the critical paths generated by data-flow and basis-path testing techniques. This module takes the identified critical variables as input and returns the paths that need to be monitored. Data-flow testing [15] of the critical variables helps in identification of all the legal sub-paths that can be taken by critical variables during execution. Basis-path testing [16] is performed to identify the minimum number of legal execution paths of the application. The path identification function as discussed in [17] builds the set of critical paths to be monitored in the application to detect and prevent tautology based SQLIA.

Runtime Monitor Development and Instrumentation Module: develops the runtime monitor for the identified critical paths and instruments it into the respective module of the application. Runtime Monitor is developed using AspectJ [18] which is an aspect-oriented extension to the Java language. A special complier provided by AspectJ called the AspectJ Compiler (ajc is the command to be used to invoke this complier) is used to integrate the runtime monitor into the respective module of the application. Henceforth, on every query execution, the runtime monitor tracks the identified critical variables by monitoring their execution path. When a critical variable follows an invalid path, the runtime monitor immediately detects the abnormal behavior of the application as a possible tautology based SQLIA and halts the execution of the application.

III. EVALUATION

To evaluate our proposed approach, we performed experiments to assess the false positive, false negative, and runtime overhead imposed by our technique.

We used a web application named Employee Directory from the SQL Injection Application Testbed [19] which has been used to evaluate other techniques [20, 21, 22, and 23] to detect and prevent SQLIAs. The subject web application that we used for our experiment is the Employee Directory web application which is an online employee directory consisting of about 5000 lines of code developed using JSPs.

By surveying various sources which included government security websites such as NVD (http://www.nvd.nist.gov/), OWASP (https://owasp.org/), Build Security In (https://buildsecurityin.us-cert.gov/), US-CERT (http://www.us-cert.gov/), security related mailing lists, and research papers, etc., we generated both legitimate and attack set inputs to the Employee Directory web application. In total, we used 10 tautology attack inputs and 30 legitimate inputs.

We first test the instrumented Employee Directory web application with the collected attack inputs. The runtime monitor detects the abnormal behavior of the application displaying all the records present in the table as tautology based SQLIA, and halts its execution. Then, we ran the legitimate inputs on the same instrumented Employee Directory web application, and all the legitimate inputs were executed successfully.

The experimentation performed clearly demonstrates the success of our developed runtime monitor to handle tautology based SQLIA on the Employee Directory application. The monitor successfully allowed the legitimate queries to be executed on the application and detected the tautology based SQLIA performed on the application i.e. both false positive and false negative were handled effectively. Table I and Table II below show the false negative and false positive results obtained respectively.

TABLE I. FALSE NEGATIVE RESULTS

Subject	**Total # of Tautology based Attack Inputs**	**Total # of attacks detected on instrumented web application**
Employee Directory	10	10

TABLE II. FALSE POSITIVE RESULTS

Subject	**Total # of Legitimate Inputs**	**Total # of legitimate inputs successful on instrumented web application**
Employee Directory	30	30

To determine the runtime overhead imposed by our proposed approach, we ran the legitimate inputs on the Employee Directory application which is not instrumented, and measured the response time of the application. We then ran the same legitimate inputs on the instrumented version of the Employee Directory application and recorded the response time. The difference in the response time obtained from the two versions of the application is determined as the overhead imposed by our proposed technique.

Only the legitimate test set is used for the overhead calculation, because the attack set would cause different paths of execution between the two versions, where the attacks would be successful in the original application, but be prevented in the instrumented application, leading to incorrect timing comparisons [23]. We ran our experiments five times and recorded the average time to ensure accuracy. We found that the runtime overhead imposed by our proposed technique on the Employee Directory application is no more than 4% which is comparatively less than the average overhead of WASP [24] listed as 6%. Table III below shows the comparison results obtained.

TABLE III. COMPARISION WITH WASP

Technique	**% Overhead**
Runtime Monitoring	4%
WASP	6%

IV. Conclusion

We presented a novel security testing technique called runtime monitoring to detect and prevent tautology based SQLIA on web applications, based on the development of runtime monitors using our runtime monitoring framework. The results of our evaluation showed that the runtime monitor was successfully able to detect and prevent the tautology based SQLIA performed on the target web application, and allowed legitimate inputs to access the database. Also, our technique imposed low overhead on the application. Thus, our technique ensures that the security of the application is accomplished during its post-deployment.

V. References

[1] https://www.owasp.org/index.php/Top_10_2013-Top_10

[2] http://www.scmagazine.com/sql-injection-attacks-still-enable-breaches-all-these-years-later/article/305433/#

[3] http://www.zdnet.com/sql-injection-attacks-up-69-7000001742/

[4]http://www.zdnet.com/sony-hacked-again-in-lulzsec-breach-4010022607/

[5]http://www.zdnet.com/blog/btl/6-46-million-linkedin-passwords-leaked-online/79290

[6] http://www.zdnet.com/nvidia-confirms-hackers-swiped-up-to-400000-user-accounts-7000000903/

[7] http://www.zdnet.com/8-24-million-gamigo-passwords-leaked-after-hack-7000001403/

[8] W. G. J. Halfond, A. Orso and P. Manolios, "Using Positive Tainting and Syntax-aware Evaluation to Counter SQL Injection Attacks", Proceedings of the 14th ACM SIGSOFT International Symposium on Foundations of Software Engineering, 2006.

[9] W. G. J. Halfond, J. Viegas, and A.Orso, "A Classification of SQL - Injection Attacks and Countermeasures", Proceedings of the IEEE International Symposium on Secure Software Engineering, 2006.

[10] A. Tajpour and M. Massrum, "Comparison of SQL Injection Detection and Prevention Techniques", Second International Conference on Education Technology and Computer, 2010.

[11]https://buildsecurityin.us-cert.gov/articles/best-practices/security-testing/risk-based-and-functional-security-testing

[12] https://buildsecurityin.us-cert.gov/articles/tools

[13] M. Vieira, N. Antunes, and H. Madeira, "Using web security scanners to detect vulnerabilities in web services", International Conference on Dependable Systems & Networks, 2009

[14]https://buildsecurityin.us-cert.gov/articles/tools/black-box-testing/black-box-security-testing-tools#techs

[15] K. Saleh, A. S. Boujarwah, and J. Al-Dallal, "Anomaly Detection in Concurrent Java Programs Using Dynamic Data Flow Analysis", Information and Software Technology, vol. 43, no. 15, pp. 973-981, 2001.

[16] M. E. Khan, "Different Approaches to White Box Testing Technique for finding Errors" , International Journal of Software Engineering and Its Applications, vol. 5, no. 3, 2011.

[17] R. Dharam and S. G. Shiva, "Runtime Monitoring-A Post-deployment Security Testing Technique", International Research Workshop on Advances and Innovation in Software Testing, 2012.

[18] AspectJ Cookbook, Russ Miles, December 27, 2004.

[19] SQL Injection Application Testbed. http://www-bcf.usc.edu/~halfond/testbed.html

[20] W. G. J. Halfond and A. Orso, "AMNESIA: Analysis and Monitoring for Neutralizing SQL-Injection Attacks", Proceedings of the IEEE and ACM International Conference on Automated Software Engineering, 2005.

[21] P. Bisht and P. Madhusudan, "CANDID: Dynamic Candidate Evaluations for Automatic Prevention of SQL Injection Attacks", Proceedings of the 14th ACM Conference on Computer and Communications Security, 2007.

[22] Y. Kosuga, K. Kono, M. Hanaoka, M. Hishiyama and Y. Takahama "Sania: Syntactic and Semantic Analysis for Automated Testing against SQL Injection", 23rd Annual Computer Security Applications Conference, 2007.

[23] R. Mui and P. Frankl, "Preventing SQL Injection through Automatic Query Sanitization with ASSIST", Fourth International Workshop on Testing, Analysis and Verification of Web Software, 2010.

[24] W. G. J. Halfond, A. Orso and P. Manolios, "WASP: Protecting Web Applications Using Positive Tainting and Syntax-aware Evaluation", In IEEE Transactions on Software Engineering, vol. 34, no. 1, pp. 65-81, 2008.