SQL Injection Detection & Defeating Tools

Kartik Rai
Student (B.tech Viith sem) Department of Computer science
Dronacharya College Of Engineering,Gurgaon-12350

***Abstract* - SQL injection is a form of attack that takes advantage of applications that generate SQL queries using user-supplied data without first checking or pre-processing it to verify that it is valid. The objective is to deceive the database system into running malicious code that will reveal sensitive information or otherwise compromise the server. By modifying the expected Web application parameters, an attacker can submit SQL queries and pass commands directly to the database. Many webpages take input from users, such as search terms, feedback comments or username and password, and use them to build a SQL query which is passed to the database. If these inputs are not validated, there is nothing to stop an attacker inputting malicious code, for example, that could instead instruct the database to delete a specific table of client records. Getting the SQL syntax right is not necessarily so simple and may require a lot of trial and error, but by adding additional conditions to the SQL statement and evaluating the Web application's output, an attacker can eventually determine whether, and to what extent, an application is vulnerable to SQL injection. If the code achieves an immediate result, it is an example of a first-order attack. If the malicious input is stored in a database to be retrieved and used later, such as providing input to a dynamic SQL statement on a different page, it is referred to as a second-order attack. Second-order attacks can be very successful because once data is in a database it is often deemed to be clean and so is not revalidated prior to use.**

***Index Terms*- introduction, SQL Injection Attacks, URL filter, Web Application Vulnerability Scanner.**

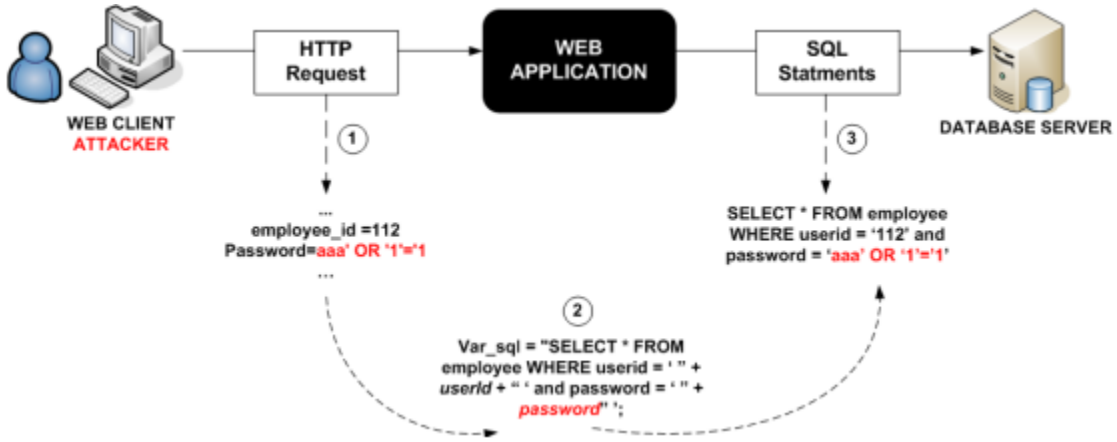
I. INTRODUCTION

According to OWASP, SQL injection vulnerabilities were reported in 2008, making up 25% of all reported vulnerabilities for web applications. An SQLIA occurs when an attacker changes the intended effect of an SQL query by inserting (or injecting) new SQL keywords or operators into the query thereby gaining unauthorized access to a database in order to view or manipulate restricted data. The root cause of SQLIA is insufficient user input validation. Although there is an increasing awareness about security, there are several significant factors that make securing web applications difficult. First web applications are growing at a frantic pace largely fuelled by the simplicity with which one can develop such applications using the numerous tools available. Secondly the developers and administrators do not have the requisite knowledge and experience in the area of security. A logical approach to tackle the problem of SQLIA is to scan the vulnerabilities present in a webpage and subsequently launch attack counter measure tools. There are a number of open-source as well as commercial tools called Web Application Vulnerability Scanners that perform security testing as well as assessment and finally report the vulnerabilities present. In spite of their continuous evolution, these automated scanners still have some problem with regard to the high number of undetected vulnerabilities and high percentage of false positives. A web access the security of web applications.

II. SQL INJECTION ATTACKS

SQLIA is a hacking technique which the attacker adds SQL statements through a web application's input fields or hidden parameters to access to resources. Lack of input validation in web applications causes hacker to be successful. For the following examples we will assume that a web application receives a HTTP request from a client as input and generates a SQL statement as output for the back end database server. For example an administrator will be authenticated after typing: employee id=112 and password=admin. Figure1 describes a login by a malicious user exploiting SQL Basically it is structured in three phases:

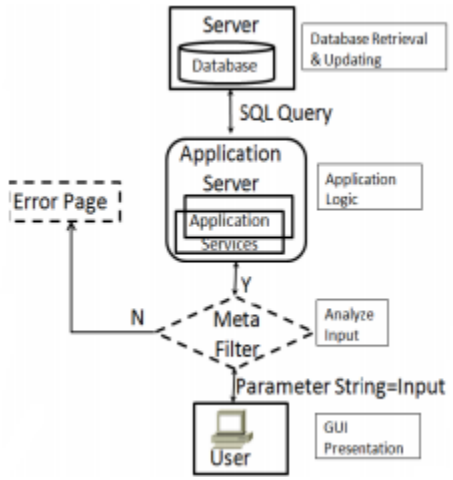
1. an attacker sends the malicious HTTP request to the web application
2. creates the SQL statement
3. submits the SQL statement to the back end database Example of a SQL Injection Attack



The above SQL statement is always true because of the Boolean tautology we appended (OR 1=1) so, we will access to the web application as an administrator without knowing the right password.

III. URL FILTER

An SQL code gets injected if an attacker manages to pass SQL Meta Characters (SQL expression) through the user input fields to change the behavior of predefined SQL queries. Thus if we can block the SQL commands in the request send to the Application Serverwe can prevent SQLIA. However, while blocking SQL commands we must ensure that legal queries and statements are not blocked. In this paper we propose to thwart SQLIA by using an URL filter . Every Request coming from the Client must pass through the URL filter first before being processed by the Application Server. If the request contains any of the attack signatures mentioned in the previous section it is denied access to the database. Our URL filter is different from a validator that blindly prohibits SQL meta characters in the input. The proposed filter prohibits a SQL Meta character if it occurs in combination with some other characters such that the database can be abused. In server-side architecture, a user invokes the services provided by the application server using a browser. The input provided by the user is usually sent to the application server in the form of a parameter string. The application server uses this input to generate a SQL query to retrieve information from the database or update it. Our proposed Meta filter is positioned between the user and application server. The filter intercepts the input from the user, parses it into SQL Meta character tokens. If the input from the user contains any attack signature then the injected input is treated as an attack and an error page is displayed , otherwise the input is processed by the application server normally.



Server-side Architecture and its interaction with our proposed Meta Filter , the SQL query and as such the SQL Meta characters are generated by the application server. Our proposed Meta filter checks for the presence of SQL Meta characters before the input is processed by the application server. Therefore, our proposed solution will not block legal inputs. Moreover, the attack patterns have been so designed so that it is robust to SQL Meta characters that can accidentally occurs in a legal input.

Comparison of Tools Based on Evaluation Parameter:

The authors of proposed tools have evaluated their tools in common parameters: efficiency, effectiveness and performance, flexibility and

stability. The results of this classification are summarized in Table 3. Definition of the measured parameters:

Efficiency

 False positive: is a false alarm. It is when the tool incorrectly categorizes a benign request being as a malicious attack.

 False negative: occurs when a malicious attack is not recognized, so the tool lets it pass normally. Effectiveness

 Attacks Detection: the percentage of real attacks, correctly detected.

 Attacks Prevention: the percentage of real attacks correctly blocked after being detected.

Flexibility

 Different Types of SQLIAs: the ability of the tool to detect/prevent different types of SQL Injection attacks such as those were presented in section II.

Performance

 Detection Overhead: is the time spent for a detection of a SQLIA once the tool is running.

 Prevention Overhead: is the time spent to detect and block (prevent) a SQLIA once the tool is running.

Stability

 Environment Independence

1. Web Applications: the possibility to test the tool on different types of web applications, such as open source/commercial, large/small.
2. Databases: testing on web applications that use different backend databases, such as open source (e.g. MySQL) commercial (e.g. Oracle).
3. Programming Languages: the ability of the tool to work on web applications written in different programming languages, such as J2EE, .NET, PHP and so On.
4. Operating Systems: the ability of the tool to run on different OS such as Windows and Linux.
5. Application Servers: the possibility to run the tool in a network using different type of Application Server such Tomcat.

IV. CONCLUSION

The tool crawls through all the web pages of a web application to discover vulnerable spots, performs a controlled exploit of the vulnerabilities at these vulnerable spots and finally verifies success of the attack and reports the result. The performance of the tool was measured by comparing it well-known SQLI scanners. Results show that our proposed scanner is able to cover more vulnerability in lesser time and has fewer false positives. The security framework proposed to defeat SQL Injection attack is based on an URL Meta filter. We analyzed well-known SQL Injection attacks and tried to identify a signature for each such attack. The filter works by checking the presence of these attack signatures in the userinput before it is processed by the application server. The proposed framework is generic and does not depend on the application server as well as the underlying database. The efficiency of the filter was tested by using the CSR Scanner. In our future work we will propose a framework for measuring effectiveness, efficiency, stability and performance of tools in common criteria to prove the strength and weakness of them.

REFERENCES

[1] W. G. Halfond, J. Viegas and A. Orso, "A Classification of SQL Injection Attacks and Countermeasures," College of Computing Georgia Institute of Technology IEEE, 2006.

[2] Sruthi Bandhakavi, Prithvi Bisht, P. Madhusudan, CANDID: Preventing SQL Injection Attacks using Dynamic Candidate Evaluation. Proceedings of the 14th ACM conference on Computer and communications security. ACM, Alexandria, Virginia, USA.page:12-24.

[3] Marco Cova, Davide Balzarotti. Swaddler: An Approach for the Anomaly-based Detection of State Violations in Web Applications. Recent Advances in Intrusion Detection, Proceedings, Volume: 4637 Pages: 63-86 Published: 2007.

[4] William G.J. Halfond, Jeremy Viegas and Alessandro Orso, "A Classification of SQL Injection Attacks and Countermeasures," College of Computing Georgia Institute of Technology IEEE, 2006.

[5] Z. Su and G. Wassermann. The Essence of Command Injection Attacks in Web Applications.

ACM SIGPLAN Notices. Volume: 41, pp: 372-382, 2006.
[6] Netsparker ofMavituna Security Ltd.: http://www.mavitunasecurity.com/netsparker/visited on January 2011
[7] Acunetix of Acunetix Ltd.:http://www.acunetix.com visited on January 2011.
[8] WebCruiser of Janus Security.:http://sec4app.com visited on January 2011.
[9] OWASP (Open Web Application Security Project) https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project visited on January 2011.