Web Services Based SQL Injection Detection and Prevention System for Web Applications

Monali R. Borade¹, Neeta A. Deshpande ²

¹PG Students, ²Assistant Professor, Matoshri College of Engineering & Research Centre, Pune University Nashik, Maharashtra, India

Abstract—Web applications form an integral part of our day to day life. The number of attacks on websites and the compromise of many individuals secure data are increasing at an alarming rate. Hence, providing increased amount of security for the users and their data becomes essential. Most important vulnerability as described in top 10 web security issues by Open Web Application Security Project is SQL Injection attack (SQLIA). SQLIA’s occur when attacker modifies the intended effects of query to gain unauthorized access of web applications and their underlying databases. The Proposed system focuses on how the advantages of randomization can be employed to prevent SQL injection attacks in web based applications. The two most important advantages of the proposed approach against existing analogous mechanisms that are, first, it prevents almost all forms of SQL injection attacks using X_PAATH validation technique (i.e. active guard and service detector components) by scanning user inputs for malicious attacks with the advantage of web services; second, use of randomization encryption algorithm provides enhanced security while detecting and preventing SQL injection attacks in database. The result clearly indicate enhancement in the performance of the system in terms of improved SQL injection detection and prevention ability and improved response time.

Keywords--Randomization, SQL injection, Vulnerability; Web Application Security, Runtime Monitoring, Service Detector, Active Guard.

I. INTRODUCTION

SQL injection is a major concern belongs to code injection problem categories. [1] Unsecured web applications allow injection attacks to perform unwanted and malicious operations on backend databases and theft of data. Hence security of web applications has become a necessity these days. In these types of attacks, input given by the user is included in an SQL query in such a way that part of the user’s input is treated as SQL code and taking advantage of these vulnerabilities, an attacker gain access to web applications and their underlying databases by submitting SQL Commands. The main reason of SQL injection vulnerabilities is insufficient validation of inputs.

To address these problems developers have proposed a variety of detection and prevention strategies [2] which provide defensive approaches such as encoding user inputs, performing input validations and many more. A rigorous and systematic combination of these approaches makes an effective solution for detecting and Preventing SQLIA’s. To address these issues this paper presents a comprehensive approach to detect and prevent SQLIA’s.

The further paper is organized as follows: Section 2 describes different types of SQL injection attacks. Section 3 focuses on current techniques for detection and prevention of SQLIA’s. Section 4 describes proposed system with its implementation details and algorithms used. Section 5 describes system evaluation and comparison of proposed system with other techniques. Finally in section 6 this paper is concluded and there will be a discussion on future trends and directions.

II. BACKGROUND

2.1 Types Of SQLIA’S

2.1.1 Tautologies

Tautology attacks are used to inject code in one or more Conditional statements so that they will always evaluate to true. This technique is most commonly used to bypass authentication pages and extract data. If the attack is Successful, the code either displays all of the returned records or performs some action if at least one record is returned. Example: In this example attack, an attacker submits “’ or 1=1 - ”.

The Query for Login mode is:

SELECT * FROM user info WHERE loginID='” or 1=1 -- AND pass1='” References: [3] [4] [5] [6] [1]

2.1.2 Illegal/Logically Incorrect Queries

This type of queries uses the error message sent from databases on being sending wrong SQL queries which may contain some useful debugging information helpful to attacker in finding parameters vulnerable to web application and their underlying databases.

Example- The error message for sending a wrong password may be like:
Select * from <tablename> where userId = <id> and password = <wrongPassword> or 1=1;
References: [7] [8] [9] [1]

2.13 Union Query
In these types of attacks, an attacker uses vulnerable parameters to change the data set returned for a given query. The Attackers do this by injecting a statement of the form: UNION SELECT <rest of injected query> because the attackers can completely control the second/injected query they can use original query to retrieve information from a specified table. In output the database returns a dataset that is the union of the results of the original first query and the results of the injected second query. [6] [10] [2] [1]

Example- SELECT accounts FROM users WHERE login='' UNION SELECT cardNo from CreditCards where acctNo=11123 -- AND pass='' AND pin=

2.1.3 Alternate Encodings
This technique performs injection by modifying query, means replacing characters in query by some other characters or symbols like – Unicode, ASCII, hexadecimal. In this way attacker can escape the filter for “wrong characters” [1]. This attack could be extremely harmful for web applications if used in combination with other techniques because it can target different layers of a web application. This method can be used to hide all kinds of SQL injection attack

Example- SELECT name FROM <tablename> WHERE id='' and password=’O; exec (char (O x232456789j776e))

2.1.4 Stored Procedures
These are the routines stored in the database and executed by the database engine. These procedures can be either user-defined or procedures provided by the database by default. Depending on the type of stored procedure there are different ways to attack. [1] It’s vulnerable to web applications. Moreover all the types of SQL injection attacks applicable to web application are also going to work at this level.

Example: CREATE PROCEDURE DBO.isAuthenticated @userName varchar2, @pass varchar2, @pin int EXEC("SELECT accounts FROM users WHERE login='' +@userName+ '' and pass='' +@password+ '' and pin='' +@pin");
GO

2.2 Remote Code Execution
Remote Code Injection consists of injecting code which is then interpreted/executed by the web application. This type of attack exploits poor handling of untrusted data. These types of attacks are usually happens because of lack of proper input/output data validation.

2.3 Cross Site Scripting
In Cross-Site Scripting attacks malicious scripts are injected into the otherwise benign and trusted web sites. XSS occurs when an attacker make use a web application to send malicious code, usually in the form of a browser side script, to a different end user.

2.4 Local File Inclusion (LFI)
Local File Inclusion (LFI) is the process of including local files on a server through the web browsers. This attack occurs when a page include is not properly sanitized, and allows directory traversal characters (such as dot-dot-slash) to be injected.

III. LITERATURE SURVEY
Until now Researchers have proposed a range of techniques to address the issue of SQL injection attacks. This technique varies from developing best coding practices to fully automated frameworks for detecting and preventing SQLIAs. This section summarizes some of these techniques with their features and limitations.

3.1 Defensive Coding Practices
The main reason of SQL injection vulnerabilities is improper input validation. Therefore, the perfect solution application of suitable defensive coding Practices like input type checking, encoding user inputs, positive pattern matching, and identification of all input sources to these vulnerabilities. Although defensive coding practices are the best way to prevent SQL injection vulnerabilities, but their application in reality is problematic. Defensive coding practices are prone to human errors and cannot completely and rigorously apply as automated techniques [11] [12] [13] [1].
3.2 Static Code Checkers

Tautology checker is a technique proposed by Wassermann and Su which provides an analysis framework for providing security in web application. It combines static analysis with automated reasoning to verify whether the SQL queries generated in the application layer contains a tautology or not [14]. This technique has limited scope of detecting and preventing tautologies and cannot detect other type of vulnerabilities [1].

3.3 Instruction Set Randomization

SQLRand is a technique proposed by S. W. Boyd and A. D. Keromytis for preventing SQL injection attacks by instruction set randomization. This technique attaches SQL keywords with the key generated by the randomization Algorithm [15]. When an attacker having no knowledge of the key, performs the attack will fail because the query constructed by the attacker will not match with the query containing randomly generated key. The keywords in both the queries will differ, therefore prevents SQL injection attack. As it’s a static analysis technique, the security of server’s web databases is not compromised due to attack. However, implementation of a proxy server for randomization and de-randomization adds to the performance overhead [1].

3.4 Static and Dynamic Analysis Techniques

Amnesia [16] [17] [18] is the technique proposed by W. G. Hal fond and A. Orso, which a combination of static analysis and run-time monitoring used for prevention of SQL injection. It works in two phases. It builds a model of all the queries generated by the application in static phase. To do this, the tool needs to access the entire source code. The query built during run-time is validated against the model built during the Static phase in dynamic phase. But the limitation is it needs to access whole source code and response time [1].

3.5 Taint Based Approach

Web App. Hardening [19] proposed by Nguyen-Tuong and colleagues and CSSE [20] proposed by Pietraszek and Berghé tracks precise per-character taint information by modifying a PHP interpreter. The approach uses a context sensitive analysis to detect and reject queries to check for untrusted input used to create certain types of SQL tokens. A common limitation of these two approaches is that they require modifications of the runtime environment, which greatly affects portability.

3.6 Framework For Sql Retrieval On Web Applications

This technique is proposed by Haeng Kon Kim [21]. It proposes a framework for web application security. The technique has two parts - Pattern Creation Module (PCM) and Attack Detection Module (ADM). PCM creates a model of attacks based on the patterns observed from previous attacks and ADM checks whether the query fired by an application matches an existing pattern [1].

3.7 Intrusion Detection Systems

This technique is proposed by Valeur and colleagues [36] to detect SQLIAs. It uses machine learning technique to train a set of typical application queries. It builds models of the typical queries and then monitors the application at runtime to identify unmatched queries [1].

3.8 Random4 Encryption Algorithm

It is a technique proposed by Aviredy s. [5] which converts the user’s plaintext inputs into cipher text inputs by using random values from lookup table to detect and prevent SQL injection. This is just to provide the additional security to the user inputs. But the technique has limited scope.

Outcomes of literature survey: By studying these existing technique and algorithms, we concluded with the aim of removing limitations of existing techniques such as,

1. Some methods only provide SQLIAs detection abilities.
2. The response time is critical issue in many techniques.
3. The scope is limited to very few attacks detection and prevention.

IV. IMPLEMENTATION DETAILS

This section focuses on working methodology of proposed system with exploration of its different components.

4.1 System Architecture

Fig.1 describes the proposed system architecture and its components. This technique uses runtime monitoring to detect and prevent SQLIAs. The solutions can be applied to each kind of application. In the proposed solution, when the login page will be redirected to checking page where it will detect and prevent SQL injection attacks without stopping legitimate access. This proposed system will be using web services which have functions Db-to-XML generator and XPATH-validator which is an XML query language used to select specific part of an XML document?
XPAA TH is used to traverse nodes from XML and collect information. The copy of original database is maintained in the form of XML documents which is a temporary storage of sensitive data in the database.

Firstly the active guardian module detects and prevents SQL injection attacks. Second module service detector & locator authenticate and allow valid user to access the web applications. This technique monitors both statically and dynamically generated queries at runtime and checks them for malicious inputs. If the comparison does not satisfies the rules proposed in the model then it is treated as a potential SQLIA’s and blocked from gaining access to the database.

The proposed system has 4 modules: two filtrations modules, one encryption module and web services layer to detect and prevent SQL injections.

4.1.1 Active Guardian Filtration module:

This module works at application layer which builds a susceptibility detector for Detecting and preventing the malicious characters or Meta characters from queries in order to prevent the malicious attacks from gaining access to the user sensitive data stored in database. Active guard accepts query as an input and removes special characters, Meta characters, and special symbols. It stores the filtered input into array lists. Encryption algorithm then encrypts the filtered inputs using lookup table. Then it finds out the attack type using SQL_Attack function. The procedure then compares the filtered inputs with the original inputs and if inputs are valid, active guard provides inputs to service detector and locator.

4.1.2 Random Encryption algorithm:

This algorithm converts the user’s plaintext inputs into cipher text inputs by using random values from lookup table. This is just to provide the additional security to the user inputs because as we know Cipher text is difficult and time consuming to crack.

4.1.3 Service detector & locator module:

This model also works at application layer which validates user inputs from XPAATH-validator where user’s sensitive data from database stored in XML format. This is the second level of filtration of queries. The user inputs are compared with the existing inputs from XPAATH-Validator, if they matched the access is granted to user, if not the access is blocked.

4.1.4 Web service layer:

Service detector & locator module calls web services using SOAP protocol. Web services execute two types of processes that are DB-to-XML generator and XPAATH-validator. DB-to-XML generator creates temporary copy of user sensitive data from database in an XML format. The user inputs from service detector & locator are then compare with the data existed in XML documents in XPAATH-validator, if the data is matched XPAATH-validator sends a flag with the count value=1 to service detector & locator module by conveying that the user data is valid.

4.1.5 Identifying vulnerabilities or hotspots:

This module perform simple step of scanning web applications in order to identify vulnerabilities or hotspots. Each vulnerability will be verified with the active server by removing the malicious characters from the code using specific methods like system.data.sqlclient namespaces.1 and sqlcommand.executereader (string). Using these methods dynamically generated queries are categorized as malicious or non-malicious.

Fig 1. Architecture Diagram
4.2 Procedure of Runtime Data Monitoring

When a Web application fails to properly sanitize the parameters, which are passed to, dynamically created SQL statements (even when using parameterization techniques). It is possible for an attacker to alter the construction of back-end SQL statements. When an attacker is able to modify an SQL statement, the statement will execute with the same rights as the application user; when using the SQL server to execute commands that interact with the operating system, the process will run with the same permissions as the component that executed the command (e.g., database server, application server, or Web server), which is often highly privileged. Current technique as shown in Fig 1 appends With Active Guard, to validate the user input fields to detect the Meta character and prevent the malicious attacker. Transact-SQL statements will be prohibited directly from user input.

For each hotspot, statically build a Susceptibility detector in Active Guard to check any malicious strings or characters append SQL tokens (SQL keywords and operators), delimiters, or string tokens to the legitimate command. Concurrently in Web service the DB_2_Xml Generator generates a XML document from database and stored in X_PAATH Validator. Service Detector receive the validated user input from active guard and send through the protocol SOAP (Simple Object Access Protocol) to the web service from the web service the user input data compare with XML Validator if it is identical the XML Validator send a flag as a count Value = 1 to Service Detector then the user consider as Legitimate user and allowed to access the web application. If the data mismatches the XML Validator send a flag as a count Value = 0 to Service Detector through the SOAP protocol then the illegitimate/invalid user is not authenticated to access the web application. If the script builds an SQL query by concatenating hard-coded strings together with a string entered by the user, As long as injected SQL code is syntactically correct, tampering cannot be detected programmatically. String concatenation is the primary point of entry for script injection Therefore, we compare all user input carefully with Service Detector (Second filtration model). If the user input and Sensitive data's are identical then executes constructed SQL commands in the Application server. Existing techniques directly allows accessing the database in database server after the query validation. Web Service oriented X_PAATH authentication technique does not allow directly to access database in database server.

V. EXPERIMENTAL RESULTS

5.1 Experimental Setup

The proposed technique is deployed and tried few trial runs on the web server. The evaluation parameters are Execution time/Response time to Queries from web application and number of attack types detected and prevented. Figure 2 shows the results of proposed technique on different web application and its prevention capacity. It shows the result for two web applications based on windows operating system another based on open source operating system. The graph in Figure 2 along with Table 1 shows the number of applied attacks on the web application along with prevented number of attacks and successful attacks.

Table 1: Results of proposed System on Different web Applications

<table>
<thead>
<tr>
<th>Web application</th>
<th>Total attacks</th>
<th>Successful attacks</th>
<th>Prevented attacks</th>
</tr>
</thead>
<tbody>
<tr>
<td>online student entry system</td>
<td>13</td>
<td>0</td>
<td>13</td>
</tr>
<tr>
<td>social networking application</td>
<td>14</td>
<td>0</td>
<td>14</td>
</tr>
</tbody>
</table>
5.2 Evaluation Parameters / Performance Measures

5.2.1. Execution Time/ Response Time

It’s the time taken by server to execute the query submitted by user & respond them back. For the proposed system the response time will also be dependent on the web application using the system.

*Execution Time Comparison of Proposed System With Existing System at runtime validation:* The runtime validation incurs some overhead in terms of execution time at both the Web Service Oriented X_PAATH Authentication Technique (Proposed Technique) and SQL-Query based Validation Technique (Existing Technique) as depicted in Table 2.

*Analysis:* Taken a sample web site we measured the extra computation time at the query validation, this delay has been amplified in the graph (Figure 2) to distinguish between the Time delays using bar chart shows that the data validation in XML Validator performs better than query validation due to following differences:

i. In Query validation the user input is generated as a query in script engine then it gets parsed in to separate tokens then the user input is compared with the statistical generated data if it is malicious generates error reporting.

   ii. In Web Service Oriented X_PAATH Authentication Technique user input is generated as a query in script engine then it gets parsed in to separate tokens, and send through the protocol SOAP to Susceptibility Detector, then the validated user data is sequentially send to Service Detector through the protocol SOAP then the user input is compared with the sensitive data, which is temporarily stored in data set. If it is malicious data, it will be prevented otherwise the legitimate data is allowed to access the Web application.

### 5.2.2 SQLIA Detection and Prevention Ability of Proposed System

Both the protected and unprotected web Applications are tested using different types of SQLIA’s; namely use of Tautologies, Union, Piggy-Backed Queries, stored procedures and different authentication bypass attacks like remote code execution, malicious file upload and many other.

<table>
<thead>
<tr>
<th>Table 2: Execution time Comparison of Existing and Proposed System</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Total number of entries in web application</strong></td>
</tr>
<tr>
<td>10</td>
</tr>
<tr>
<td>20</td>
</tr>
<tr>
<td>30</td>
</tr>
<tr>
<td>40</td>
</tr>
<tr>
<td>50</td>
</tr>
<tr>
<td>60</td>
</tr>
</tbody>
</table>

Figure 3: Execution time comparisons for proposed technique

Table 3 shows that the proposed technique prevented all types of SQLIA’s and authentication bypass attacks in almost all cases. So the prevention ability of proposed technique is observed to be approximately 100% against existing technique predefined set of attacks. The proposed technique is thus a secure and robust solution to defend against SQLIA’s and authentication bypass attacks. The Figure 4(a) and Figure 4(b) show the comparison of prevention ability of Existing techniques with proposed system. The graph clearly indicates the detection and prevention ability of implemented system.
5.3 Comparisons With Other Tools

The Figure 5 and Figure 6 show comparison of proposed system with other existing tools. The comparison is made on whether any encoding was part of the technique and the automation in detecting and preventing SQLIA’s.

5.4 Discussion

The response rate of existing system was in a range of 40000 -120000 millisecond which was directly proportional to the number of inputs to web application. As number of inputs increases the performance of the system were degraded. The response time /execution time for earlier work were more because of direct accesses provided to the database for query execution. Also the previous system’s attack detection and prevention ability was limited to number of attacks. System was static system and was able to detect and prevent only few numbers of attacks.
This is illustrated in Table 4. There is clearly an enhancement in the results of proposed system against earlier work in terms of Query execution time/ response rate and Detection and Prevention ability. The response time was reduced from range of 40000-12000 to 10000-20000 for same number of inputs as a result of X_PAATH validation technique which checks the user input with valid database which is stored separately in X_PAATH and do not affect database directly so the time delay occurred in database access was reduced and hence response time is improved as shown in Table 5. Also the detection and prevention ability was increased by approximately 60% than the earlier work for same set of predefined attacks.

Table 4:
Results of Earlier Work

<table>
<thead>
<tr>
<th>Technique</th>
<th>Response Time/ Execution Time In Milliseconds</th>
<th>Detection and Prevention Ability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random4 Encryption Algorithm</td>
<td>40000-120000 for 100-600 entries</td>
<td>Limited to 4 type of attacks</td>
</tr>
</tbody>
</table>

Table 5:
Results of Proposed Work

<table>
<thead>
<tr>
<th>Technique</th>
<th>Response Time/ Execution Time In Milliseconds</th>
<th>Detection and Prevention Ability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Web Services Based SQLIA's Detection and Prevention</td>
<td>10000-20000 for 100-600 entries</td>
<td>10 attacks successfully detected and prevented. Can be extended to more number of attacks</td>
</tr>
</tbody>
</table>

VI. CONCLUSION AND FUTURE WORK

SQL Injection Attacks attempts to modify the parameters of a Web-based application in order to alter the SQL statements and can gain unauthorized access to web applications and their underlying databases. The proposed technique is used to detect and prevent the SQLI flaw (Susceptibility characters & exploiting SQL commands) in susceptibility detector and prevent the susceptibility attacker. Web Service Oriented X_PAATH Authentication Technique checks the user input with valid database which is stored separately in X_PAATH and does not affect database directly then the validated user input field is allowed to access the web application as well as used to improve the performance of the server side validation.

The proposed technique focused on 10 types of attacks mainly which are SQL injection attacks like Tautology, Union Query etc. and authentication bypass attacks like Remote Code Execution, Cross-Site Scripting etc. This proposed technique was able to suitably classify the predefined set of attacks that performed on the applications without blocking legitimate accesses to the web application. The result of proposed work clearly indicates increased performance in terms of attack types detected and prevented and improved response time/execution time to query validation.

These observed results show that proposed technique represents a promising approach to countering SQLIA's as it uses web services which can be used by any web application to authenticate their users and provide enhanced security and also motivate further work in this direction.

VII. FUTURE SCOPE

The proposed work can be extended further to detect more number of attacks and further performance can be improved.

1. System can be modified for detection and prevention of XML attacks and attacks on web services.
2. Can be extended for detection and prevention of more number of attacks.

REFERENCES


Abhishek Kumar Baranwal, “Approaches to detect SQL injection and XSS in web applications”. EECE 571B, TERM SURVEY PAPER, APRIL 2012


Subodh Raikar, "SQL Injection Prevention using Runtime Query Modeling and Keyword Randomization".


