International Journal of Science and Research (IJSR)
ISSN (Online): 2319-7064
Impact Factor (2012): 3.358

Defense against SQL Injection and Cross Site Scripting Vulnerabilities

Kirti Randhe[1], Vishal Mogal[2]

[1]Department of Computer Engineering, RMD Sinhgad School of Engineering, Pune, Maharashtra, India

Abstract: ***As dependence on web applications is increasing very rapidly in various fields like social networks, online services, banking, etc. Access to web applications and ease of use make them more popular in offering online services instead of in person services. Due to the presence of security weakness in web applications malicious user can easily exploit various security vulnerabilities and becomes reason of their failure.SQL injection attacks and cross site scripting attacks are the two most common attacks in web application. Attack prevention techniques protect the applications from attack during their execution in actual environment. Prevention and detection of intrusion is made through a deployment of reverse proxy with the intrusion and prevention mechanism which are built in against web attacks specially SQLIA. In reverse proxy user input is sanitized which may transform into a database attack. Here data cleaning algorithm is used for sanitization application. Using this method SQL injection attack as well as cross site scripting attacks are detected.***

Keywords: SQL attacks, SQL injection, Cross site scripting, Sanitization, Vulnerabilities

1. Introduction

Due to internet evolution and expansion most of the applications have now migrated to web .Due to this effect ,the security risk associated with the web applications also increases. As most of the web applications have vulnerability in them due to which various attacks are feasible, and due to these attacks malicious hackers intend to access sensitive data in the databases.SQL injection attack is the most serious and topmost attack found among the top ten vulnerability list defined by OWASP.

SQL injection attack (SQLIA) is one of this techniques used to attack databases through the website. This attack tries to gain access to sensitive data directly by injecting malicious SQL codes through web application. SQLIA is the type of code injection attack in which an attacker uses specially crafted inputs to trick the database into executing attacker specified database commands. Cross site scripting is a type of code injection vulnerability that enables malicious user to send malicious script to web browsers. It occurs whenever a web application uses user inputs in response pages without properly validating them. When a user visits an exploited web page, the browser executes the malicious scripts sent by the application. This is called as XSS attack. Typical attack include installing malware, hijacking a users session or redirecting users to another site.
SQL injection classification:

SQL Injection
SQL Injection Attacks (SQLIA) refers to a class of code-injection attacks in which data provided by user is included in the SQL query in such a way that part of the user's input is treated as SQL code. These types of vulnerabilities come among the most serious threats for web applications. Web applications that are vulnerable to SQL injection allows an attacker to manipulate SQL queries, which are input to the database and provide them with complete access to the underlying databases. SQL Injection vulnerabilities occur because of nonexistent and/or incomplete validation of user input. As a result, an attacker can inject input that potentially alters the behavior of the script being executed [6, 8]. SQL Injection Attacks can be done in various ways like using UNION keyword, Tautology condition, Group by Having Clause etc. There are also various ways of performing such attacks which are discussed in [4], [5], and [7] by different authors.

Tautologies: The main goal of tautology-based attack is to inject code in conditional statements so that they are always evaluated as true. Using tautologies, the attacker wishes to either bypass user authentication or insert inject-able parameters or extract data from the database. A typical SQL tautology has the form, where the comparison expression uses one or more relational operators to compare operands and generate an always true condition. Bypassing authentication page and fetching data is the most common example of this kind of attack. In this type of injection, the attacker exploits an inject-able field contained in the WHERE clause of query. He transforms this conditional query into a tautology and hence causes all the rows in the database table targeted by the query to be returned. For example, SELECT * FROM user WHERE id='1' or '1=1'-'AND password='1234'; "or 1=1" is the most commonly known tautology. In this type of attack the injected code will always start with a string terminator (') followed by the conditional OR operator. The OR operator will be followed by a statement that always evaluates to true. The signature for such attacks is the string terminator (') and OR.

Logically incorrect query attacks: The main goal of the Illegal/Logically Incorrect Queries based SQL Attacks is to gather the information about the back end database of the Web Application. When a query is rejected, an error message is returned from the database including useful debugging information. This error messages help attacker to find vulnerable parameters in the application and consequently database of the application. In fact attacker injects junk input or SQL tokens in query to produce syntax error, type mismatches, or logical error by purpose. In this example attacker makes a type mismatch error by injecting the following text into the input field:

Licensed Under Creative Commons Attribution CC BY

1)Original URL: http://www.toolsmarket-al.com/veglat/?id_nav=2234
2)SQL Injection: http://www.toolsmarket-al/veglat/?id_nav=2234'
3) Error message showed: SELECT name FROM Employee WHERE id=2234\'. From the message error we can find out name of table and fields: name; Employee; id. By the gained information attacker can organize more strict attacks. The Illegal/Logically Incorrect Queries based SQL attack is considered as the basis step for all the other techniques. In this type of attack there are several ways to perform illegal or incorrect queries like incorrectly terminating the string ('), using AND operator to perform incorrect logics using order by, etc.

Piggy-Backed Queries: The main goal of the Piggy- Backed Query is to execute remote commands or add or modify data. In this attack type, an attacker tries to inject additional queries along with the original query, which are said to "piggy-back" onto the original query. As a result, the database receives multiple SQL queries for execution. Vulnerability of this kind of attack is dependent of the kind of database [5]. For example, if the attacker inputs [';drop table users--] into the password field, the application generates the query: SELECT Login_ID FROM users_ID WHERE login_ID='john' and password=''; DROP TABLE users-' AND ID=2345 After executing the first query, the database encounters the query delimiters (;) and execute the second query. The result of executing second query would result into dropping the table users, which would likely destroy valuable information. The signature of this attack is (;), the database line terminator.

Inference: The main goal of the inference is to change the behavior of a database or application. There are two well-known attack techniques that are based on inference: blind injection and timing attacks.

***Blind Injection*:** Sometimes developers hide the error details which help attackers to compromise the database. In this situation attacker face to a generic page provided by developer, instead of an error message. So the SQLIA would be more difficult but not impossible. An attacker can still steal data by asking a series of True/False questions through SQL statements. Consider two possible injections into the login field: For example, SELECT accounts FROM users WHERE id= '1111' and 1 =0 – AND pass = AND pin=0 SELECT accounts FROM users WHERE login= 'doe' and 1 = 1 -- AND pass = AND pin=0 If the application is secured, both queries would be unsuccessful, because of input validation. But if there is no input validation, the attacker can try the chance. First the attacker submits the first query and receives an error message because of "1=0 ". So the attacker does not understand the error is for input validation or for logical error in query. Then the attacker submits the second query which always true. If there is no login error message, then the attacker finds the login field vulnerable to injection. The possible guessing start with the AND operator and some time attacker also uses conditional operators.

Union Query: The main goal of the Union Query is to trick the database to return the results from a table different to the one intended. By this technique, attackers join injected query to the safe query by the word UNION and then can get data about other tables from the application. This technique is mainly used to bypass authentication and extract data. For example the query executed from the server is the following: SELECT Name, Phone FROM Users WHERE Id=$id. By injecting the following Id value: $id =1 UNION ALL SELECT credit Card Number, 1 FROM Credit sys Table. We will have the following query: SELECT Name, Phone FROM Users WHERE Id=1 UNION ALL SELECT credit card Number, 1 FROM Credit sys Table. This will join the result of the original query with all the credit card users. The signature of this attack is UNION character of SQL.

Cross Site Scripting
Cross-Site Scripting also known as XSS is another very harmful attack type of code injection attack . This flaw occurs mainly due to the lack of input validation and encoding. XSS allows attackers to execute script in the victim's browser, which can hijack user sessions, deface web sites, insert hostile content, and conduct phishing attacks . Any scripting language supported by the victim's browser can also be a potential target for this attack. All web application frameworks are vulnerable to XSS. Different types of XSS Attacks are discussed in and it also shows how such attacks are carried out.

2. Related Work

SQLrand [3] appends a random token to SQL keywords and operators in the application code. This approach was created randomize instances of the SQL query language, by randomizing the template query inside the CGI script and the database parser. The drawbacks of this approach are that the secret token could be guessed, thus making the approach ineffective, and that the approach requires the deployment of a special proxy server.

In SQL Guard and SQL Check [8, 9] queries are checked at runtime based on a model which is expressed as a grammar that only accepts legal queries. SQLCheck identifies SQLIAs by using an augmented grammar and distinguishing untrusted inputs from the rest of the strings by means of a marking mechanism. SQL Guard examines the structure of the query before and after the addition of user-input based on the model. In two approaches developer should to modify code to use a special intermediate library or manually insert special markers into the code where user input is added to a dynamically generated query. CANDID[5] modifies web applications written in Java through a program transformation. This tool dynamically mines the programmer-intended query structure on any input and detects attacks by comparing it against the structure of the actual query issued. CANDID's natural and simple approach turns out to be very powerful for detection of SQL injection attacks.

AMNESIA[7] combines static analysis and runtime monitoring. In static phase, it builds models of the different types of queries which an application can legally generate at each point of access to the database. Queries are intercepted before they are sent to the database and are checked against

Licensed Under Creative Commons Attribution CC BY

the statically built models, in dynamic phase. Queries that violate the model are prevented from accessing to the database. The primary limitation of this tool is that its success is dependent on the accuracy of its static analysis for building query models.

SQLIDS [11] approach utilizes security specifications that describe the intended syntactic structure of SQL statements that are produced by the application. SQL statements that do not conform to the specifications are considered as security violations and their execution is blocked. The detection technique is based on the assumption that injected SQL commands have differences in their structure with regard to the expected SQL commands that are built by the scripts of the web application. Therefore, if the intended structure of the expected SQL commands has been explicitly pre-determined, it is possible to detect malicious modifications that alter this structure.

An Approach to Detect and Prevent SQL Injection Attacks in Database Using Web Service identifies the SQLIA by using web services oriented XPATH authentication Technique [2]. To detect and prevent SQLIA with runtime monitoring, without stopping the operation login page is redirected to their checking page. Active Guard worked as susceptibility detector to detect and prevent the susceptibility Character and meta character. Service Detector filtration model use to validate user input from XPATH_validator where the sensitive data is stored. User input field compare with data existed in XPATH_validator if it is identical then authorized user is allowed for next processing. Web Service Oriented XPATH Authentication Technique does not allow directly to access database in database server.

SQLProb [6] approach is that the extracted user input data in the context of the syntactic structure of the query can be evaluated. This approach is fully modular and does not require access to the source code of the web applications or the database. Also the system is easily deployable to existing enterprise environments and can protect multiple front-end web applications without any modifications. Experimental results indicated that high detection rate with reasonable performance overhead can be achieved making the system ideal for environments where software or architecture changes is not an economically viable option.

3. Proposed Framework for Data Sanitization

Figure 3 illustrates the block diagram of Reverse Proxy Model. In a client server model, a reverse proxy server is placed, in between the client and the server. The presence of the proxy server is not known to the user. The sanitizing application is placed in the Reverse proxy server. A reverse proxy is used to sanitize the request from the user. When the request become high, more reverse proxy's can be used to handle the request. This enables the system to maintain a low response time, even at high load.

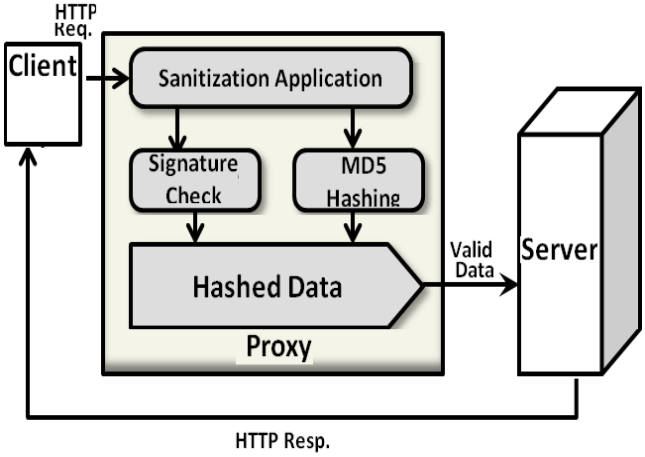

Figure 1: Block Diagram of Reverse Proxy Model

3.1 Steps for Data Sanitization

The client sends the request to the server. The request is red-irected to the reverse proxy. The sanitizing application in the proxy server extracts the URL from the HTTP and the user data from the SQL statement. First the URL is send to the signature check and after that the user data (Using prototype query model) is encrypted using the MD5 hash.. The sanitizing application sends the validated URL and hashed user data to the web application in the server. The filter in the server denies the request if the sanitizing application had marked the URL request malicious. If the URL is found to be benign, then the hashed value is send to the database of the web application. If the hashed user data matches the stored hash value in the database, then the data is retrieved and the user gains access to the account. Otherwise the user is denied access. When the first request is over database response is returned to client and fetch the next request.

The general work of Reverse proxy model is as follows:

- The client sends the request onto the server.
- The request is redirected onto the reverse proxy.
- Study the payload for next processes.
- The sanitizing application among the proxy server extracts the URL direct from HTTP and naturally the user data from the SQL statement.
- The URL is send onto the signature check. The reader data (Using prototype query model) is encrypted making use of the MD5 hash.
- Check is it a query, if yes send for signature check otherwise send data to the client.
- The sanitizing application sends the validated URL and hashed user data onto the web application in the whole server.
- The filter within the server denies the request in case the sanitizing application had marked the URL request malicious.
- The sanitizing application among the proxy server extracts the URL direct from HTTP and naturally the user data from the SQL statement.
- The URL is send onto the signature check.
- The reader data (Using prototype query model) is encrypted making use of the MD5 hash.
- Check is it a query, if yes send for signature check otherwise send data to the client.

Licensed Under Creative Commons Attribution CC BY

- The sanitizing application sends the validated URL and hashed user data onto the web application in the whole server.
- The filter within the server denies the request in case the sanitizing application had marked the URL request malicious.

TABLE 1. PSEUDO CODE FOR SANITIZATION PROCESS

```
PROPOSED SANITIZATION PROCESS
INPUT: USER DATA AND EXTRACT THE URL FROM THE HTTP;
OUTPUT: GAIN ACCESS TO VALID ACCOUNT
PARSE THE USER DATA INTO TOKEN-TOK;
 WHILE (TOK! = 0)
{
 IF (TOK = RESERVED SQL KEYWORD)
 THEN
 MOVE (TOK TO USER DATA ARRAY-UDA);
 FOR (EVERY DATA IN UDA)
 {
 CONVERT TO CORRESPONDING MD5 AND STORE IN MD5-UDA;
 }
PARSE THE URL INTO TOKENS-TOKS;
 WHILE (TOKS! = 0)
 {
 IF (URL = BENIGN USING THE SIGNATURE CHECK)
 SET THE FLAG TO CONTINUE;
 ELSE
 SET THE FLAG TO DENY;
 }
 SEND THE MD5-UDA AND FLAG TO WEB APPLICATION SERVER;
 }
```

3.2 Extraction of User Data

The SQL statement is extracted from the HTTP request and the query is tokenized. The tokenized query is in comparison to the prototype document. A prototype document consists of all of the SQL queries from the Web application. The query tokens are transformed into XML format. The XSL's pattern matching algorithm is designed to locate the prototype model equivalent to the received Query. XSL's Pattern Matching: The query is first analyzed and tokenized as elements. The prototype document maintains the query pertained to the next particular application. By way of example the input query is, SELECT * FROM members WHERE login='admin' AND password= 'XYZ' OR '1=1'.When this query is received this is converted into XML format using a XML schema.

4. Conclusion and Future Work

In this paper we first identified various types of SQLIA's. Then we investigated on SQL injection detection and prevention techniques. A good understanding of SQL injection techniques can help developers to make their applications and a network more secure against this vulnerability. We have presented an efficient approach ie. A reverse proxy framework which is based on sanitization technique and it is mainly using Data cleaning and Message Digest algorithm. In the future work the focus will be on optimization of the system and detecting alternate techniques that will increase effectiveness ,efficiency and performance of the system.

References

[1] W. G. J. Halfond, et al., *"A Classification of SQL-Injection Attacks and Countermeasures,"* in Proceedings of the IEEE International Symposium on Secure Software Engineering, Arlington, VA, USA, 2006.

[2] Indrani Balasundaram 1 Dr. E. Ramaraj, "An Approach to Detect and Prevent SQL Injection Attacks in Database Using Web Service," in IJCSNS International Journal of Computer Science and Network Security, VOL.11 No.1, January 2011.

[3] Stephen W. Boyd and Angelos D. Keromytis Department of Computer Science Columbia University, "SQLrand: Preventing SQL Injection Attacks".

[4] Shaimaa Ezzat Salama, Mohamed I. Marie, Laila M. El-Fangary & Yehia K. Helmy, "Web Anomaly Misuse Intrusion Detection Framework for SQL Injection Detection", (IJACSA) International Journal of Advanced Computer Science and Applications, Vol. 3, No. 3, 2012

[5] Veera Venkateswaramma P, "An Effective Approach for Protecting Web from SQL Injection Attacks", International Journal of Scientific & Engineering Research Volume 3, Issue 3, March -2012

[6] Anyi Liu, Yi Yuan,Duminda Wijesekera," SQLProb: A Proxy-based Architecture towards Preventing SQL Injection Attacks", SAC'09 March 8-12, 2009, Honolulu, Hawaii, U.S.A.

[7] William G.J. Halfond and Alessandro Orso "Preventing SQL Injection Attacks Using AMNESIA" , ICSE'06, May 20–28, 2006, Shanghai, China.

[8] Zhendong Su, Gary Wassermann," The Essence of Command Injection Attacks in Web Applications", January 11.13, 2006, Charleston, South Carolina, USA.

[9] Gregory T. Buehrer, Bruce W. Weide, and Paolo A. G. Sivilotti, "Using Parse Tree Validation to Prevent SQL Injection Attacks", September 2005 Lisbon, Portugal.

[10] Anil Kumar Mandapati , Adi lakshmi.Yannam, "Web Application Vulnerability Analysis Using Robust SQL Injection Attacks", International Journal of Engineering Trends and Technology- Volume3Issue5- 2012

[11] Konstantinos Kemalis and Theodoros Tzouramanis," SQL-IDS: A Specification-based Approach for SQL-Injection Detection", SAC'08, March 16-20, 2008, Fortaleza, Ceará, Brazil.

Licensed Under Creative Commons Attribution CC BY