SQL INJECTION AND PREVENTION TECHNIQUES

Prof. B. C. Julme, Mr.Videh Paliwal, Mr.Sukrut Badhe, Mr.Ninad Narayane, Mr. Vikrant Bhise

Department of Computer Science. University of Pune MH (India)

Abstract*:*

An SQL injection attack targets interactive Web applications that employ database services. Some papers in literature even refer to stored procedures as a remedy against SQL injection attacks.Injection attack is a method that can inject any kind of malicious string or anomaly string on the original string. During an SQL injection attack, an attacker might provide malicious SQL query segments as user input which could result in a different database request. This technique combines static application code analysis with runtime validation to eliminate the occurrence of such attacks.Most of the pattern basedtechniques are used static analysis and patterns are generated from the attacked statements. In the initial step evaluation, we consider somesample of standard attack patterns and it shows that the proposed algorithm is everything well against the SQL InjectionAttack.

Keywords:
SQL Injection Attack; Pattern different; Static Pattern; Dynamic Pattern Crafting

Introduction

SQL Injection is a method of exploiting the database of web application. It is done by injecting the SQL statements as an input string to gain an unauthorized access to a database.

These databases and user personal information is target to the attacks. Web applications are typically interact with backenddatabase SQL server MySQL, MSSQL, ORACLE, POSTGRESS, toward name a few. This low–level interaction (or) communication is dynamic (or) session based because it does not take into account the structure of the output language. Injection is a code injection technique that exploits security vulnerability in website software. This happens because the developers are not fully aware of attacks by SQL Injection and its causes. SQL-Injection Attacks (SQLIA's) are one of the topmost threats for web application security. Intrusion Detection System (IDS) and Network Intrusion Detection System (NIDS). The Web applications that are vulnerable to SQL-Injection attacks user inputs the attacker's embeds commands and gets executed. Hence we can say that SQL injection attack is an unauthorized access of database. NIDS are not support for the service oriented applications (web attack), because NIDS are working lower level layers as shown in figure [11]

An SQL injection is a kind of injection vulnerability in which the attacker tries to inject arbitrary pieces of malicious data into the input fields of an application, which, when processed by the application, causes that data to be executed as a piece of code by the back end SQL server, thereby giving undesired results which the developer of the application did not anticipate. The backend server can be any SQL server (MySQL, MSSQL, ORACLE, POSTGRESS, to name a few) The ability of the attacker to execute code (SQL statements) through vulnerable input parameters empowers him to directly interact with the back end SQL server, thereby leveraging almost a complete compromise of system in most cases.

What are different types of SQL injections?

SQL injections can be classified and categorized in different ways, based on the type of data extraction channel, the response received from server, how server responses aid in leveraging the successful exploitation, impact point, etc.

Based on the data extraction channel

- Inband or inline
- Out-of-band

SQL injections that use the same communication channel as input to dump the information back are called inband or inline SQL Injections. This is one of the most common methods, readily explained on the Internet in different posts. For example, a query parameter, if injectable, leads to the dumping of info on the web page.

Injections that use a secondary or different communication channel to dump the output of

ISSN: 2278 – 7798

queries performed via the input channel are referred to as out-of-band SQL injections. For example, the injection is made to a web application and a secondary channel such as DNS queries is used to dump the data back to the attacker domain.

Literature Survey:

An SQL injection attack has a set of properties, suchas assets under threat, vulnerabilities being exploited andattack techniques utilized by threat agents.Model-based guard constructor prevention is an efficient method in preventing an SQL injection attack. his method is established on breaking the of input, code, data, and database access situation that would employ an SQL injection attack. As shown in the , initially instrument the PHP string to collect the samples of query which authentically used at database application program interface call point.

Stored procedures have the same effect as the use of prepared statements when implemented safely*. They require the developer to define the SQL code first, and then pass in the parameters after. The difference between prepared statements and stored procedures is that the SQL code for a stored procedure is defined and stored in the database itself, and then called from the application. Both of these techniques have the same effectiveness in preventing SQL injection so your organization should choose which approach makes the most sense for you.

*Note: 'Implemented safely' means the stored procedure does not include any unsafe dynamic SQL generation. Developers do not usually generate dynamic SQL inside stored procedures. However, it can be done, but should be avoided. If it can't be avoided, the stored procedure must use input validation or proper escaping as described in this article to make sure that all user supplied input to the stored procedure can't be used to inject SQL code into the dynamically generated query. Auditors should always look for uses of sp_execute, execute or exec within SQL Server stored procedures. Similar audit guidelines are necessary for similar functions for other vendors.

Related Work

Types of SQL Injection Attacks

In this section, we present and discuss the different kinds of SQL Injection Attacks. The different types of attacks are generally not performed in isolation; many of them are used together or sequentially, depending on the specific goals of the attacker. Note also that there are countless variations of each attack type.

Tautologies

Attack Intent: Bypassing authentication; identifying injectable parameters; extracting data.

Description: The general goal of a tautology-based attack is to inject code in one or more conditional statements so that they always evaluate to true. The most common usages are to bypass authentication pages and extract data. In this type of injection, an attacker exploits an injectable field that is used in a query's WHERE conditional.

Transforming the conditional into a tautology causes all of the rows in the database table targeted by the query to be returned. In general, for a tautology-based attack to work, an attacker must consider not only the injectable/vulnerable parameters, but also the coding constructs that evaluate the query results. (Halfond, Viegas, & Alessandro , 2006)

Example 1: Bypassing login script.

Query: *SELECT name from authors where username = '$_POST[username]' AND password='$_POST[password]';*

This query take input from the system user; suppose the user enters:

Username: a' OR '1=1'

Password: a' OR '1=1'

Constructed query: *SELECT name from authors where username = 'a' OR '1=1' AND password='a' OR '1=1'*

The code injected in the conditional (OR 1=1) transforms the entire WHERE clause into a tautology. The database uses the conditional as the

ISSN: 2278 – 7798

basis for evaluating each row and deciding which to return. Because the condition, the query evaluates to true for each row and returns all of them. This would cause this user to be authenticated as the user whose data is in the first row in the returned result set.

Solution:

$username = $_POST[username];

$username = mysqli_real_escape_string ($username);

mysql_query (SELECT first_name, last_name from authors where username = '$username');

Illegal/Logically Incorrect Queries

Attack Intent: Identifying injectable parameters; Performing database finger printing; Extracting data.

Description: This attack lets the attacker gather important information about the type and structure of the back-end database of an application. The attack is considered a preliminary, information gathering step for other attacks. The vulnerability leveraged by this attack is that the default error page returned by application servers is often overly descriptive; originally intended to help programmers debug their applications, further helps attackers gain information about the schema of the back-end database. When performing this attack, an attacker tries to inject statements that cause a syntax, type conversion, or logical error into the database. Syntax errors can be used to identify injectable parameters. Type errors can be used to deduce the data types of certain columns or to extract data. Logical errors often reveal the names of the tables and columns that caused the error.

Example 2: Cause a type conversion error that can reveal relevant data.

Password: AND 'pin: "convert (int, (select top 1 name from sysobjects where xtype='u'))

Query: *SELECT name from authors where username = '' AND password='' AND 'pin = convert (int,(select top 1 name from sysobjects where xtype='u'))*

The query attempts to extract the first user table (xtype='u') from the database's metadata table (assume the application is using Microsoft SQL Server, for which the metadata table is called sysobjects). The query then tries to convert this table name into an integer. Because this is not a legal type conversion, the database throws an error. For Microsoft SQL Server, the default error would be "Microsoft OLE DB Provider for SQL Server (0x80040E07) Error converting nvarchar value 'CreditCards' to a column of data type int."

Two useful pieces of information in this message aids an attacker. First, the attacker can see that the database is an SQL Server database. Second, the error message reveals the value of the string that caused the type conversion to occur. In this case, this value is also the name of the first user-defined table in the database: "CreditCards." A similar strategy can be used to systematically extract the name and type of each column in the database. Using this information about the schema of the database, an attacker can then create further attacks that target specific pieces of information.

Union Query

Attack Intent: Bypassing Authentication; extracting data.

Description: In union-query attacks, an attacker exploits a vulnerable parameter to change the data set returned for a given query. With this technique, an attacker can trick the application into returning data from a table different than the one that was intended by the developer. Attackers do this by injecting a statement of the form: UNION SELECT <rest of injected query>. Because the attackers completely control the second/injected query, they can use that query to retrieve information from a specified table. 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. One example usage of this multiple attacks is where the attacker uses the logically incorrect query attack to data about a table's structure then use the union query to get data from this table.

Example 3: Referring to example 2, an attacker could inject the text

Username: ' UNION SELECT cardNo from CreditCards where acctNo=10032 - -"

ISSN: 2278 – 7798

Query: *SELECT name from authors where username = '' UNION SELECT cardNo from CreditCards where acctNo=10032 -- AND password=''*

Note: It is common technique to force the SQL parser to ignore the rest of the query written by the developer with -- which is the comment sign in SQL.

Assuming that there is no login equal to "", the original first query returns the null set, whereas the second query returns data from the "CreditCards" table. The database takes the results of these two queries, unions them, and returns them to the application.

Piggy Backed Queries

Attack Intent: Extracting data; Adding or modifying data; Performing DOS; executing remote commands.

Description: In this attack, an attacker tries to inject additional queries into the original query. We distinguish this type from others because, in this case, attackers are not trying to modify the original intended query; instead, they are trying to include new and distinct queries that "piggy-back" on the original query. As a result, the database receives multiple SQL queries which are all executed. This type of attack can be extremely harmful. If successful, attackers can insert virtually any type of SQL command, including stored procedures into the additional queries and have them executed along with the original query. Vulnerability to this type of attack is often dependent on having a database configuration that allows multiple statements to be contained in a single string.

Example 4: The attacker inputs:

Password: ""; drop table users - -"

Query: *SELECT name from authors where username = '' AND password=''* drop table users -- AND pin=123

After completing the first query, the database would recognize the query delimiter (";") and execute the injected second query. Dropping the users table would likely destroy valuable information. Other types of queries could insert new users into the database or execute stored procedures. Note that many databases do not require a special character to separate distinct queries, so simply scanning for a query separator is not an effective way to prevent this type of attack.

Solution: Configure the database to block executing multiple statements within a single string.

Stored Procedures

Attack Intent: Performing privilege escalation; performing DOS; Executing remote commands.

Description: SQL Injection Attacks of this type try to execute stored procedures present in the database. Most vendors ship databases with a standard set of stored procedures that extend the functionality of the database and allow for interaction with the operating system. Therefore, once an attacker determines which backend database is in use, SQL Injection Attacks can be crafted to execute stored procedures provided by that specific database. Additionally, because stored procedures are often written in special scripting languages, they can contain other types of vulnerabilities, such as buffer overflows; these vulnerabilities allow attackers to run arbitrary code on the server or escalate their privileges. Here is a stored procedure that checks credentials:

```
CREATE PROCEDURE DBO. is Authenticated

@userName varchar2, @pass varchar2, @pin int

AS EXEC ("SELECT accounts FROM users

WHERE login='" +@userName+ "' and pass='" +@password+ "' and pin=" +@pin);

GO
```

Example 5: Demonstrates how a parameterized stored procedure can be exploited via an SQL Injection Attack. In the example, we assume that the query string constructed at lines 5, 6 and 7 of our example has been replaced by a call to the stored procedure defined in Figure 2. The stored procedure returns a true/false value to indicate whether the user's credentials authenticated correctly. To launch an SQL Injection Attack, the attacker simply enters:

Password: ' ; SHUTDOWN; --

ISSN: 2278 – 7798

Query: *SELECT name from authors where username = 'Jay' AND password=' '*; SHUTDOWN; --

At this point, this attack works like a piggy-back attack. The first query is executed normally, and then the second, malicious query is executed, which results in a database shut down. This example shows that stored procedures can be vulnerable to the same range of attacks as traditional application code

> Instrumentation is nobody but to add an output instruction before database application interface calls.

But sanitization and validation are far from the whole story. Here are ten ways you can help prevent or mitigate SQL injection attacks:

Trust no-one: Assume all user-submitted data is evil and validate and sanitize everything.

Don't use dynamic SQL when it can be avoided: used prepared statements, parameterized queries or stored procedures instead whenever possible.

Update and patch: vulnerabilities in applications and databases that hackers can exploit using SQL injection are regularly discovered, so it's vital to apply patches and updates as soon as practical.

Firewall: Consider a web application firewall (WAF) – either software or appliance based – to help filter out malicious data. Good ones will have a comprehensive set of default rules, and make it easy to add new ones whenever necessary. A WAF can be particularly useful to provide some security protection against a particular new vulnerability before a patch is available.

Reduce your attack surface: Get rid of any database functionality that you don't need to prevent a hacker taking advantage of it. For example, the xp_cmdshell extended stored procedure in MS SQL spawns a Windows command shell and passes in a string for execution, which could be very useful indeed for a hacker. The Windows process spawned by **xp_cmdshell** has the same security privileges as the SQL Server service account.

Use appropriate privileges: don't connect to your database using an account with admin-level privileges unless there is some compelling reason to do so. Using a limited access account is far safer, and can limit what a hacker is able to do.

Keep your secrets secret: Assume that your application is not secure and act accordingly by encrypting or hashing passwords and other confidential data including connection strings.

Don't divulge more information than you need to: hackers can learn a great deal about database architecture from error messages, so ensure that they display minimal information. Use the "Remote Only" custom Errors mode (or equivalent) to display verbose error messages on the local machine while ensuring that an external hacker gets nothing more than the fact that his actions resulted in an unhandled error.

Don't forget the basics: Change the passwords of application accounts into the database regularly. This is common sense, but in practice these passwords often stay unchanged for months or even years.

Buy better software: Make code writers responsible for checking the code and for fixing security flaws in custom applications before the software is delivered. SANS suggests you incorporate terms from this sample contract into your agreement with any software vendor.

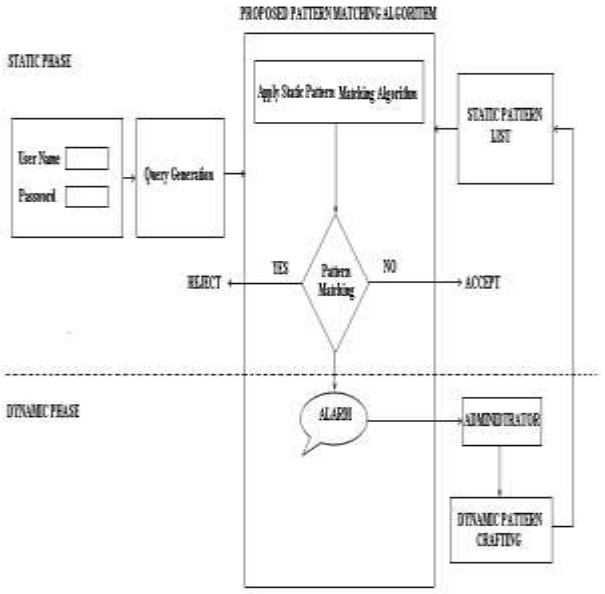
RELATED WORK

In web based security problems, SQLIA has the top most priority.Basically, we can classify the detectionand prevention techniques into two broad categories. Firstapproach is trying to detect SQLIA through checkingAnomalous SQL Query structure using string matching, pattern matching and query processing. In the secondapproach uses data dependencies among data items which areless likely to change for identifying malicious databaseactivities. In both the categories, many of the researchersproposed different schemes with integrating data mining andintrusion detection systems.Hal fond et al [21] developed a technique that uses a model–based approach to detect illegal queries before they areexecuted on the database. William et al [20] proposed a system WASP to prevent SQL Injection Attacks by a method called

ISSN: 2278 – 7798

positive tainting. Srivastava et al [22] offered a weighted sequence mining approach for detecting data base attacks.The contribution of this paper is to propose a technique for detecting and preventing SQLIA using both static phase and dynamic phase.The anomaly SQL Queries are detectionin static phase. In the dynamic phase, if any of the query is identified as anomaly query then new pattern will be created from the SQL Query and it will be added to the Static Pattern List (SPL).

Architecture:



Explain:

The proposed architecture is given in figure 2 below. The proposed scheme has the following two modules, 1) Static Phase and 2) Dynamic Phase

In the Static Pattern List, we maintain a list of known Anomaly Pattern. In Static Phase, the user generated SQL Queries are checked by applying Static Pattern Matching Algorithm. In Dynamic Phase, if any form of new anomaly is occur then Alarm will indicate and new Anomaly Pattern will be generated. The new anomaly pattern will be updated to the Static Pattern List. The following steps are performed during Static and Dynamic Phase,

Static Phase

Step 1: User generated SQL Query is send to the proposed Static Pattern Matching Algorithm

Step 2: The Static Pattern Matching Algorithm is given in Pseudo Code is given below.

Step 3: The Anomaly patterns are maintained in Static Pattern List, during the pattern matching process each pattern is compared with the stored Anomaly Pattern in the list

Step 4: If the pattern is exactly match with one of the stored pattern in the Anomaly Pattern List then the SQL Query is affected with SQL Injection Attack

Dynamic Phase

Step 1: Otherwise, Anomaly Score value is calculated for the usergenerated SQL Query, If the Anomaly Score value is morethan the Threshold value, then a Alarm is given andQuery will be pass to the Administrator.

Step 2: If the Administrator receives any Alarm then the Querywill be analyze by manually. If the query is affected byany type of injection attack then a pattern will be generated and the pattern will be added to the StaticPattern list.

Algorithms

Static Pattern Matching Algorithm

1: Procedure SPMA(Query, SPL[ ])
INPUT: Query□ User Generated Query
SPL[ ]□Static Pattern List with m Anomaly Pattern
2: For j = 1 to m do
3: compare all values query length and pattern values if both are same then
4: calculate anomaly value
5: If (anomly) Score Value Anomaly ≥ Threshold
6: then
7: Retrun Alarm □ Administrator
8: Else
9: Return Query □ Accepted
10: End If
11: Else
12: Return Query □ Rejected
13: End If
14: End For

End Procedure

Aho – Corasick Multiple Keyword Matching Algorithm

1: Procedure AC(y,n,q0)
INPUT: y**[]** array of m bytes representing the text input (SQL Query Statement)

ISSN: 2278 – 7798

1:N[] integer representing the text length (SQL Query Length) q0 [] initial state (first character in pattern)
2: State [ ] q0
3: For i = 1 to n do
4: While g (State, y[i] = = fail) do
5: *State* ← *f* (*State*)
6: End While
7: *State* ← *g*(*State*,.*y*[*i*])
8: If o(State) □ then
9: Output i
10: Else
11: Output
12: End If
13: End for
14: End Procedure

CONCLUSION:

In this paper, we have proposed a scheme for detection andprevention of SQL Injection Attack using Aho–Corasick pattern matching algorithm. Initial stage evaluation shows that the proposed scheme is produce notfalse positive and false negative. The pattern matching processtakes minimum of O (n) time.
Thiscan help in fixing or atleast reducing the possibility of occurrence of vulnerability that can damage the databasesecurity.Present day development is more focused on Web Applications so there is an urgent need for educating the developers & Students on SQL Injection thereby allowing programmers and system administrators to understand the attacks more thoroughly, more attacks will be detected and more countermeasures will be introduced into the systems.

REFERENCES

[1] Amit Kumar Pandey, "SECURING WEB APPLICATIONS FROMAPPLICATION-LEVEL ATTACK", master thesis, 2007
[2] C.J. Ezeife, J. Dong, A.K. Aggarwal, "SensorWebIDS: A Web MiningIntrusion Detection System", International Journal of Web InformationSystems, volume 4, pp. 97-120, 2007
[3] S.Axelsson, "Intrusion detection systems: A survey and taxonomy",Technical Report, Chalmers Univ., 2000
[4] Marhusin, M.F.; Cornforth, D.; Larkin, H., "An overview of recentadvances in intrusion detection", in proceeding of IEEE 8th Internationalconference on computer and information technology CIT, 2008
[5] S. F. Yusufovna., "Integrating Intrusion Detection System and DataMining", International Symposium on Ubiquitous MultimediaComputing, 2008
[6] Low, W. L., Lee, S. Y., Teoh, P., "DIDAFIT: Detecting Intrusions inDatabases Through Fingerprinting Transactions", in Proceedings of the4th International Conference on Enterprise Information Systems (ICEIS),2002
[7] F. Valeur, D. Mutz, and G.Vigna, "A learning-based approach to thedetection of sql injection attacks", in proceedings of the conference ondetection of intrusions and Malware and vulnerability assessment(DIMVA), 2005
[8] Bertino, E., Kamra, A, Terzi, E., and Vakali, A, "Intrusion detection inRBAC-administered databases", in the Proceedings of the 21st AnnualComputer Security Applications Conference, 2005

ISSN: 2278 – 7798