OVERVIEW OF SQL INJECTION

SUMMARY

SQL (Structured Query Language) Injection is one of most vulnerable and serious attack that can be occurred on dynamic web applications which can have a serious impact on the data stored in the databases. In this write up I have explained the concept of SQL Injection, its types, how it occurred, and prevention and detection methods

INTRODUCTION

As we know, the internet has become a wide-spread solution for the information system. Data is the most crucial component of an information system .This large amount of data is stored in databases for different kinds according to the type of the data.

Further, invention of smart phones gave rise to a lot of creative technology and gave a new aspect to the use of mobile phones for the users. Most Android applications need to save data like user settings or large amounts of information.
With the continuous increase in the number of users accessing the web applications, mobile phone security of this data is becoming more essential for an organization.
Though the user stored personal data in the database it can be accessed from anywhere around the globe over the network which makes it vulnerable for the attackers to gain access or modify the data using the code injection technique. Code injection means injecting the malicious code into the application program with the intention of exploiting or compromising the database.

Overview of SQL INJECTION

SQL is the acronym for Structured Query Language. It is used to retrieve and manipulate data in the database. SQL query is the way to insert, modify, extract, and delete the data in the database. SQL injection the attack in which attacker interfere the queries that are done to the

Attacker can perform this attack with many different intentions like:

1.To pull out data: Attackers can take out the sensitive information stored on the database . One example is if the attacker gains access to the admin database .

2.To extract data – Sensitive data will be grabbed by the attacker. Suppose if the admin database is hacked the entire database becomes vulnerable.

3.To access data – They try to break the privileges and get access to the entire database and try to manipulate the data.

4.Finger print the database- In this attack, the database version and its type will be derived out by the attacker. This attack helps them to try different types of queries in different applications.

5.Injectable parameters are found – using some of the automatic tools the vulnerable parameters will be found for attack.

6. Authentication Bypass –application authentication mechanisms will be bypassed to enter inside the database.

7 Database schema identification - From the database table name, data type of each field, column name, etc. will be retrieved to gather information successfully

8.To perform denial of service – Dropping table and system shutdown falls under this category. Attacker tries to intrude inside the system to perform some specific instruction within the database

TYPES OF SQL INJECTION

As mentioned in many of the research papers SQL Injection can be of the following types:

1. **TAUTOLOGIES BASED SQL INJECTION ATTACK**: In logic mathematics, tautology is the formulae which is possible in every possible case. In a tautology-based attack, the code is injected using the conditional OR operator such that the query always evaluates to TRUE.

   The main purpose for this form of SQL injection is to Identify injectable parameters, Bypass authentication and Extract data

   It is done by inserting a tautology such as (1=1) or (- -).in the WHERE clause of the SQL query. The query transforms the original condition into a tautology, causes all the rows in the database table are open to an unauthorized user.

2. **SQL INJECTION UNION ATTACK** :When an application is vulnerable to SQL injection and the results of the query are returned within the application's responses, the UNION keyword can be used to retrieve data from other tables within the database. This results in an SQL injection UNION attack.

   This type of attack uses Union Operator (U) while inserting the SQL Query. The two sql query are joined with the Union Operator. The first statement is a normal query after which the malicious query is appended with union operator. Hence, it is used to bypass the prevention and detection mechanism of the system. The example shows how it can be proceeded. The example shows that the second query is malicious and text following (–) is disregarded as it becomes comment for the SQL Parser. Taking this an advantage the attacker attacks the web application or website with this query. select * from accounts where id='212' UNION select * from credit_card where user='admin'--' and pass='pass

   3.**Piggybacked Queries**: Intent of Attack- Retrieval of Information, Denial of Service In this type of attack the attacker "Piggy Back" the query with the original query in the input fields present on the web application. The piggy backed can be defined as "on or as if on the back of another". The database receives the multiple queries [3]. During the execution, the premier query works as in a normal case the second query adjoined with the first query is used for SQL Injection Attack. It is considered as a menacing attack since it fully exploits the database. Using proper prevention and detection techniques this type of attack can be prevented. As an

Example: select customer_info from accounts where login_id = ‘‘admin’’ AND pass = ‘123’ ; DELETE FROM accounts WHERE CustomerName = ’Albert’; After executing the first query the interpreter sees the ‘;’ SemiColon and executes the second query with the first query. The second query is malicious so it will delete the all the data of the customer ‘Albert’. Hence, these type of malicious act can be protected by firstly determining the correct SQL Query through proper validation or to use different detection techniques. This type of attack can be prevented using Static Analysis, Run time monitoring is not needed.

4) **Stored Procedure**: Intent of Attack: Escaping Authentication, Denial of Service, More freedom on Database Stored Procedures are widely used as a subroutine in a relational database management system. They are compiled into single execution plan and extensively used for performing commonly occurring tasks. Its used in businesses as it provides single point of control while performing the business rules. IT Professionals think that SQL Stored Procedures are remedy for the SQL Injection as Stored Procedures are placed on the front of the databases the security features cannot be applied to them. The stored procedures do not use the standard Structured Query Language, it uses its own scripting languages which does not have same vulnerability as SQL but different vulnerability related to the Scripting language still exist. As an Example: CREATE PROCEDURE User_info @username varchar2 @pass varchar2 @customerid int AS BEGIN EXEC(‘Select customer_info from customer_table where username=‘ ‘‘+@username ’’ ’ and pass = ‘ ‘‘+@pass ’’ ’ GO

This type of procedures are vulnerable for the SQL Injection Attack. Any malicious user can enter the malicious data in the fields of username and password. The simple command entered by the user can destroy whole database or it can lead to service disruption. It is always advised, not to store the critical information on the stored procedures, as it lacks the most important security features.

5.**Blind SQL Injections**: Blind SQL injection is a type of SQL injection attack where the attacker indirectly discovers information by analyzing server reactions to injected SQL queries, even though injection results are not visible. Blind SQL injection attacks are used against web applications that are vulnerable to SQL injection but don’t directly reveal information. While more time-consuming than regular SQL injection, blind SQL injection attacks can be automated to map out the database structure and extract sensitive information from the database server.

Attackers came up with methods to go around the lack of error messages and still know if the input is being interpreted as an SQL statement. This is how the *Blind SQL Injection* technique was born (sometimes called *Inferential SQL Injection*). There are two variants of this technique that are commonly used: *Content-based Blind SQL Injection* and *Time-based Blind SQL Injection*.

Content-based Blind SQL Injection

In the case of a Content-based Blind SQL Injection attack, the attacker makes different SQL queries that ask the database TRUE or FALSE questions. Then they analyze differences in responses between TRUE and FALSE statements.

This is an example of a web page of an online shop, which displays items that are for sale. The following link will display details about item 34, which are retrieved from a database.

http://www.shop.local/item.php?id=34

The SQL statement used for this request is:

SELECT column_name, column_name_2 FROM table_name WHERE id = 34

The attacker may manipulate the request to:

http://www.shop.local/item.php?id=34 and 1=2

The SQL statement changes to:

SELECT column_name_2 FROM table_name WHERE ID = 34 and 1=2SELECT name, description, price FROM Store_table WHERE ID = 34 and 1=2

This will cause the query to return FALSE and no items are displayed in the list. The attacker then proceeds to change the request to:

http://www.shop.local/item.php?id=34 and 1=1

And the SQL statement changes to:

SELECT column_name, column_name_2 FROM table_name WHERE ID = 34 and 1=1SELECT name, description, price FROM Store_table WHERE ID = 34 and 1=1

This returns TRUE, and the details of item with ID 34 are shown. This is a clear indication that the page is vulnerable.

Time-based Blind SQL Injection

In the case of time-based attacks, the attacker makes the database perform a time-intensive operation. If the web site does not return a response immediately, the web application is vulnerable to Blind SQL Injection. A popular time-intensive operation is the *sleep* operation.

Based on the previous example, the attacker would first benchmark the web server response time for a regular query. They would then issue the following request:

http://www.shop.local/item.php?id=34 and if(1=1, sleep(10), false)

The web application is vulnerable if the response is delayed by 10 seconds.

Consequences of Blind SQL Injections

Blind SQL Injections are often used to build the database schema and get all the data in the database. This is done using brute force techniques and requires many requests but may be automated by attackers using SQL Injection tools.

Acunetix can detect Blind SQL Injection vulnerabilities. Acunetix also includes a Blind SQL Injector tool, which allows the penetration tester to verify that the Blind SQL vulnerability exists and demonstrate the consequences of the vulnerability. Take a demo and find out more about running Blind SQLi scans against your website or web application.

PREVENTION FROM SQL INJECTION

An organization can adopt the following policy to protect itself against SQL Injection attacks.

User input should never be trusted - It must always be sanitized before it is used in dynamic SQL statements.

Stored procedures – these can encapsulate the SQL statements and treat all input as parameters.

Prepared statements –prepared statements to work by creating the SQL statement first then treating all submitted user data as parameters. This has no effect on the syntax of the SQL statement.

Regular expressions –these can be used to detect potential harmful code and remove it before executing the SQL statements.

Database connection user access rights –only necessary access rights should be given to accounts used to connect to the database. This can help reduce what the SQL statements can perform on the server.

Error messages –these should not reveal sensitive information and where exactly an error occurred. Simple custom error messages such as "Sorry, we are experiencing technical errors. The technical team has been contacted. Please try again later" can be used instead of display the SQL statements that caused the error

Perform Penetration Tests-Internal QA and security testing are important, but catching everything is next to impossible. Many third-party companies will perform tests against an application to test for many vulnerabilities, including SQL injection.

As companies whose sole purpose is to locate security flaws, they will tend to have a high level of success in uncovering vulnerabilities that were overlooked internally. Once documented, an organization can fix each vulnerability and a future test can validate the improvement over time.
Common threats found by penetration-testing companies are:

- Insecure Passwords
- Default users or passwords are used
- Injection (including SQL injection)

- Unpatched software vulnerabilities
- Out-of-support/end-of-life software
- Configuration mistakes
- Cross-Site Scripting

An organization benefits greatly from hiring a third-party company and allowing them to perform a site-wide security test. Even more important is to repeat these tests regularly to ensure that new vulnerabilities are found and dealt with quickly, so as to minimize the exposure period.
Even if you have a security officer or team, the ability to find everything on one's own is practically impossible. An alternative to hiring a security firm would be to purchase penetration testing software and run the tests yourself. This can also be effective and removes the potential challenges of allowing third-party company access to your systems.

Code Review-This is a critical step of the software development process. The code should always be reviewed prior to testing and release. In an ideal world, code would be reviewed by multiple people, each with a different area of expertise, such as security, performance, or application domain knowledge**.**

Code review allows problems to be located and fixed early in the development lifecycle and long before they reach users. Code review cannot replace other security measures, but is an exceptionally important step that can quickly allow us to find bugs that could very well lead to security vulnerabilities (among other bugs). The time spent up-front on code review will save significantly more time later on bug identification and remediation.

Minimizing the Impact of SQL Injection

In addition to preventing SQL injection, we would be negligent if we did not identify our ability to make mistakes and acknowledge the need to have other security measures as well. Building solid security, in general, helps in reducing the impact of SQL injection and ensures that we are not one coding mistake away from a data breach!

While writing better code is important, it alone will not prevent us from being the victim of SQL injection. Why?

- We do not have the time to review and re-review all code (and do so perfectly) to ensure that we have caught everything
- We cannot control code outside of our domains, such as vendor code, OS code, microservices, or other apps that are maintained by other parties
- Anticipating all bugs, past, present, and future is a futile task
- Applications evolve and the code that is secure today may not be so secure tomorrow. Now infamous Intel vulnerabilities Meltdown and Spectre underscore this fact!
- People are human and make mistakes

The moral of this story is that we should have tight security on top of preventing SQL injection. This minimizing the impact of any mistakes or vulnerabilities we may encounter and ensures that even if someone gains unauthorized access, their success will be greatly limited and (hopefully) detected.

Principle of Least Privilege & Login Security

Always limit server and database security the bare minimum of what is needed. The sysadmin role should truly be limited to those few people whose job it is to administer a server. The db_owner security role should be reserved for the small number of use-cases in which an operator needs full database control, which is generally rare.

Applications should not need either of these roles and should be able to operate within the confines of some amount of read/write/execute access to a specific database or set of databases. Vendor apps will occasionally request sysadmin or db_owner privileges for the purpose of installations or updates. These scenarios should be investigated thoroughly and if those permissions are indeed required, determine if they can be rescinded when the installation is complete. Allowing an app to have unfettered server access is dangerous. Allowing that access to an app that is out of your control is even riskier.

Different apps should use different logins. Login sharing is dangerous and makes it harder to identify the source of a connection. Ideally, Windows authentication is used whenever possible.

In addition to being more secure, it is easier to control access as there are additional permissions that can be adjusted in Active Directory that can enable, disable, or alter users via user-specific or group policies.

Similarly, different SQL Server services should have different logins. SQL Server, SQL Server Agent, SSRS, and SSIS should operate on different credentials than the applications that use them. Each of those services should also take advantage of unique logins, thus minimizing exposure if any one of those logins were compromised. These logins should also be distinct from those used for other servers, services, and file shares.