Project Training

Session 7

SQL Injections

SQL Injection (SQLi) refers to an injection attack wherein an attacker can execute malicious SQL statements (also commonly referred to as a malicious payload) that control a web application's database server (also commonly referred to as a Relational Database Management System – RDBMS). Since an SQL Injection vulnerability could possibly affect any website or web application that makes use of an SQL-based database, the vulnerability is one of the oldest, most prevalent and most dangerous of web application vulnerabilities.

SQL Injections

By leveraging an SQL Injection vulnerability, given the right circumstances, an attacker can use it to bypass a web application's authentication and authorization mechanisms and retrieve the contents of an entire database. SQL Injection can also be used to add, modify and delete records in a database, affecting data integrity.

To such an extent, SQL Injection can provide an attacker with unauthorized access to sensitive data including, customer data, personally identifiable information (PII), trade secrets, intellectual property and other sensitive information.

SQL Injections

In order to run malicious SQL queries against a database server, an attacker must first find an input within the web application that is included inside of an SQL query.

In order for an SQL Injection attack to take place, the vulnerable website needs to directly include user input within an SQL statement. An attacker can then insert a payload that will be included as part of the SQL query and run against the database server.

SQL Injections

The following server-side pseudo-code is used to authenticate users to the web application.

```
# Define POST variables
uname = request.POST['username']
passwd = request.POST['password']
# SQL query vulnerable to SQLi
sql = "SELECT id FROM users WHERE username='" + uname + "' AND password='" + passwd + "'"
# Execute the SQL statement
database.execute(sql)
```

The above script is a simple example of authenticating a user with a username and a password against a database with a table named users, and a username and password column.
The above script is vulnerable to SQL Injection because an attacker could submit malicious input in such a way that would alter the SQL statement being executed by the database server.A simple example of an SQL Injection payload could be something as simple as setting the password field to password' OR 1=1.

SQL Injections

What's the worst an attacker can do with SQL?

SQL is a programming language designed for managing data stored in an RDBMS, therefore SQL can be used to access, modify and delete data. Furthermore, in specific cases, an RDBMS could also run commands on the operating system from an SQL statement.

Keeping the above in mind, when considering the following, it's easier to understand how lucrative a successful SQL Injection attack can be for an attacker.

- An attacker can use SQL Injection to bypass authentication or even impersonate specific users.
- One of SQL's primary functions is to select data based on a query and output the result of that query. An SQL Injection vulnerability could allow the complete disclosure of data residing on a database server.

SQL Injections

- Since web applications use SQL to alter data within a database, an attacker could use SQL Injection to alter data stored in a database. Altering data affects data integrity and could cause repudiation issues, for instance, issues such as voiding transactions, altering balances and other records.
- SQL is used to delete records from a database. An attacker could use an SQL Injection vulnerability to delete data from a database. Even if an appropriate backup strategy is employed, deletion of data could affect an application's availability until the database is restored.
- Some database servers are configured (intentional or otherwise) to allow arbitrary execution of operating system commands on the database server. Given the right conditions, an attacker could use SQL Injection as the initial vector in an attack of an internal network that sits behind a firewall.

SQL Injections

The anatomy of an SQL Injection attack

An SQL Injection needs just two conditions to exist – a relational database that uses SQL, and a user controllable input which is directly used in an SQL query.

In the example below, it shall be assumed that the attacker's goal is to exfiltrate data from a database by exploiting an SQL Injection vulnerability present in a web application.

Supplying an SQL statement with improper input, for example providing a string when the SQL query is expecting an integer, or purposely inserting a syntax error in an SQL statement cause the database server to throw an error.

SQL Injections

The anatomy of an SQL Injection attack

Errors are very useful to developers during development, but if enabled on a live site, they can reveal a lot of information to an attacker. SQL errors tend to be descriptive to the point where it is possible for an attacker to obtain information about the structure of the database, and in some cases, even to enumerate an entire database just through extracting information from error messages – this technique is referred to as error-based SQL Injection. To such an extent, database errors should be disabled on a live site, or logged to a file with restricted access instead.

SQL Injections

The anatomy of an SQL Injection attack

Another common technique for exfiltrating data is to leverage the UNION SQL operator, allowing an attacker to combine the results of two or more SELECT statements into a single result. This forces the application to return data within the HTTP response – this technique is referred to as union-based SQL Injection.

Types of SQL Injection (SQLi)

SQL Injection can be used in a range of ways to cause serious problems. By levering SQL Injection, an attacker could bypass authentication, access, modify and delete data within a database. In some cases, SQL Injection can even be used to execute commands on the operating system, potentially allowing an attacker to escalate to more damaging attacks inside of a network that sits behind a firewall.

SQL Injection can be classified into three major categories – In-band SQLi, Inferential SQLi and Out-of-band SQLi.

Types of SQL Injection (SQLi)

In-band SQLi (Classic SQLi)

In-band SQL Injection is the most common and easy-to-exploit of SQL Injection attacks. In-band SQL Injection occurs when an attacker is able to use the same communication channel to both launch the attack and gather results.

The two most common types of in-band SQL Injection are Error-based SQLi and Union-based SQLi.

Types of SQL Injection (SQLi)

Error-based SQLi

Error-based SQLi is an in-band SQL Injection technique that relies on error messages thrown by the database server to obtain information about the structure of the database. In some cases, error-based SQL injection alone is enough for an attacker to enumerate an entire database. While errors are very useful during the development phase of a web application, they should be disabled on a live site, or logged to a file with restricted access instead.

Types of SQL Injection (SQLi)

Union-based SQLi

Union-based SQLi is an in-band SQL injection technique that leverages the UNION SQL operator to combine the results of two or more SELECT statements into a single result which is then returned as part of the HTTP response.

Types of SQL Injection (SQLi)

Inferential SQLi (Blind SQLi)

Inferential SQL Injection, unlike in-band SQLi, may take longer for an attacker to exploit, however, it is just as dangerous as any other form of SQL Injection. In an inferential SQLi attack, no data is actually transferred via the web application and the attacker would not be able to see the result of an attack in-band (which is why such attacks are commonly referred to as “blind SQL Injection attacks”). Instead, an attacker is able to reconstruct the database structure by sending payloads, observing the web application’s response and the resulting behavior of the database server.

The two types of inferential SQL Injection are Blind-boolean-based SQLi and Blind-time-based SQLi.

Types of SQL Injection (SQLi)

Boolean-based (content-based) Blind SQLi

Boolean-based SQL Injection is an inferential SQL Injection technique that relies on sending an SQL query to the database which forces the application to return a different result depending on whether the query returns a TRUE or FALSE result.

Depending on the result, the content within the HTTP response will change, or remain the same. This allows an attacker to infer if the payload used returned true or false, even though no data from the database is returned. This attack is typically slow (especially on large databases) since an attacker would need to enumerate a database, character by character.

ORM (Object Relational Mapper)

Object-relational mapping (ORM, O/RM, and O/R mapping tool) in computer science is a programming technique for converting data between incompatible type systems using object-oriented programming languages. This creates, in effect, a "virtual object database" that can be used from within the programming language. There are both free and commercial packages available that perform object-relational mapping, although some programmers opt to construct their own ORM tools.

However, many popular database products such as SQL database management systems (DBMS) can only store and manipulate scalar values such as integers and strings organized within tables. The programmer must either convert the object values into groups of simpler values for storage in the database (and convert them back upon retrieval), or only use simple scalar values within the program. Object-relational mapping implements the first approach.

ORM (Object Relational Mapper)

Pros and Cons

Using ORM saves a lot of time because:

- DRY: You write your data model in only one place, and it's easier to update, maintain, and reuse the code.
- A lot of stuff is done automatically, from database handling to I18N.
- It forces you to write MVC code, which, in the end, makes your code a little cleaner.
- You don't have to write poorly-formed SQL (most Web programmers really suck at it, because SQL is treated like a "sub" language, when in reality it's a very powerful and complex one).
- Sanitizing; using prepared statements or transactions are as easy as calling a method.

ORM (Object Relational Mapper)

Pros and Cons

Using an ORM library is more flexible because:

- It fits in your natural way of coding (it's your language!).
- It abstracts the DB system, so you can change it whenever you want.
- The model is weakly bound to the rest of the application, so you can change it or use it anywhere else.
- It lets you use OOP goodness like data inheritance without a headache.

ORM (Object Relational Mapper)

Pros and Cons

But ORM can be a pain:

- You have to learn it, and ORM libraries are not lightweight tools;
- You have to set it up. Same problem.
- Performance is OK for usual queries, but a SQL master will always do better with his own SQL for big projects.
- It abstracts the DB. While it's OK if you know what's happening behind the scene, it's a trap for new programmers that can write very greedy statements, like a heavy hit in a for loop.

Assignment 3

For next week:

- Implement the stream page of the current logged user
  - Create, Update, Delete post
  - Create, Update, Delete a comment
  - Like/Unlike a post or comment