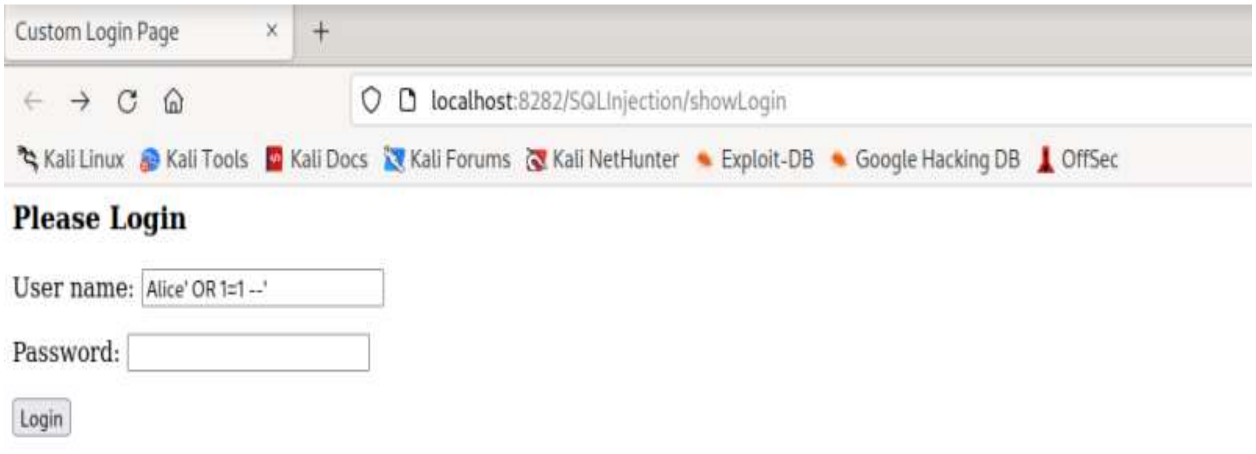
Task 5.1P: SQL Injection

For my work, I focused on SQL Injection web application as this one is used to demonstrate the SQL injection attacks. The objective was to need to discover the weak point in the code that caused the problem, then bring out the flaw to be able to log in even without using a valid password.

Making Use of the Vulnerability

- I entered the following input - Alice' OR 1=1 --"

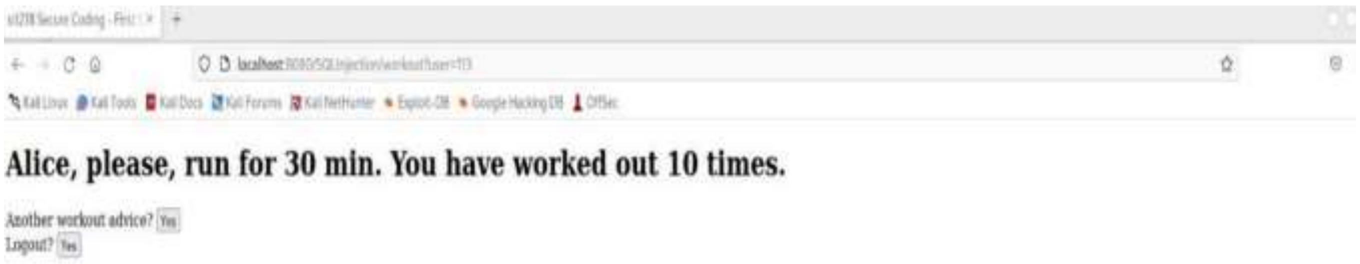


Please Login

User name: Alice' OR 1=1 --'

Password:

- According to me, through this particular SQL injection string, I was in a position to log into the application without entering the correct password. The application is insecure because the input from the field username is used directly in the SQL query without performing the validation step that is supposed to take place.



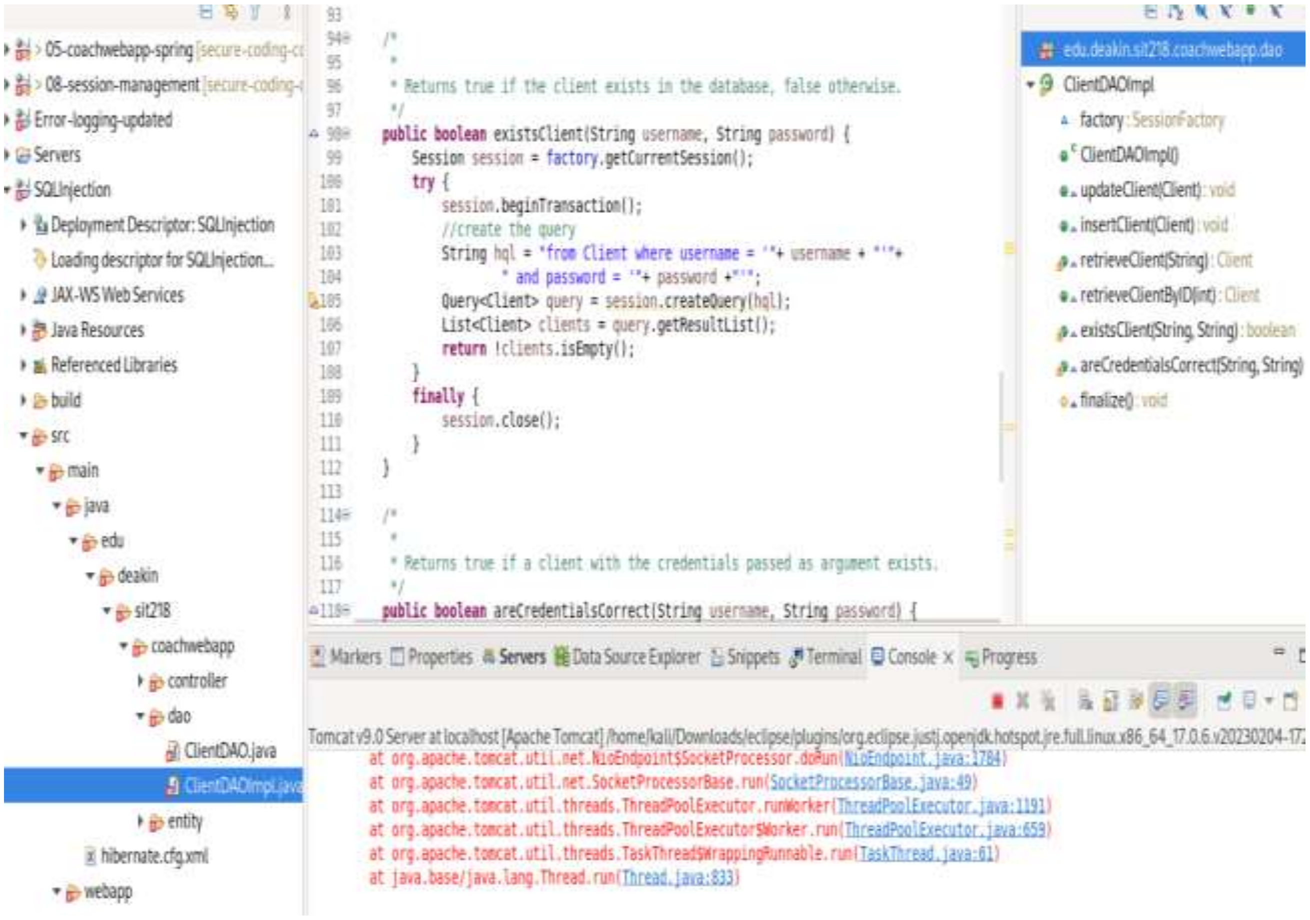
Alice, please, run for 30 min. You have worked out 10 times.

Another workout advice?

Logout?

How to Determine the Vulnerable Code

- There are routines in place. The vulnerable code is present in the client and are Credentials Correct. The problem is that the entered login and password are inserted into the SQL query at once. SQL injections can be an issue due to the lack of sanitization.


05-coachwebapp-spring [secure-coding-c
08-session-management [secure-coding-
Error-logging-updated
Servers
SQLInjection
Deployment Descriptor: SQLInjection
Loading descriptor for SQLInjection...
JAX-WS Web Services
Java Resources
Referenced Libraries
build
src
main
java
edu
deakin
sit218
coachwebapp
controller
dao
ClientDAO.java
ClientDAOImpl.java
entity
hibernate.cfg.xml
webapp
edu.deakin.sit218.coachwebapp.dao
ClientDAOImpl
factory : SessionFactory
ClientDAOImpl()
updateClient(Client) : void
insertClient(Client) : void
retrieveClient(String) : Client
retrieveClientByID(int) : Client
existsClient(String, String) : boolean
areCredentialsCorrect(String, String)
finalize() : void
Markers
Properties
Servers
Data Source Explorer
Snippets
Terminal
Console
Progress

```java
    /*
     *
     * Returns true if the client exists in the database, false otherwise.
     */
    public boolean existsClient(String username, String password) {
        Session session = factory.getCurrentSession();
        try {
            session.beginTransaction();
            //create the query
            String hql = "from Client where username = '"+ username + "'"+
                    " and password = '"+ password +"'";
            Query<Client> query = session.createQuery(hql);
            List<Client> clients = query.getResultList();
            return !clients.isEmpty();
        }
        finally {
            session.close();
        }
    }

    /*
     *
     * Returns true if a client with the credentials passed as argument exists.
     */
    public boolean areCredentialsCorrect(String username, String password) {
```

```
Tomcat v9.0 Server at localhost [Apache Tomcat] /home/kali/Downloads/eclipse/plugins/org.eclipse.justj.openjdk.hotspot.jre.full.linux.x86_64_17.0.6.v20230204-172
	at org.apache.tomcat.util.net.NioEndpoint$SocketProcessor.doRun(NioEndpoint.java:1784)
	at org.apache.tomcat.util.net.SocketProcessorBase.run(SocketProcessorBase.java:49)
	at org.apache.tomcat.util.threads.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1191)
	at org.apache.tomcat.util.threads.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:659)
	at org.apache.tomcat.util.threads.TaskThread$WrappingRunnable.run(TaskThread.java:61)
	at java.base/java.lang.Thread.run(Thread.java:833)
```

- This code is weak because there is no input validation: The application allows the user's input directly into the query without determination of the input as risky. Direct SQL Concatenation: Attacks are easy with this due to the fact that the SQL query is made by joining strings and hence the attackers may change the logic of a query.

Reasons for Code Vulnerability

- The problem is with the makeup of the query. Since user input is directly placed within the SQL query, an attacker can “bypass authentication” by introducing an SQL injection code (OR 1=1). So normally when the SQL statement compares to TRUE, the attacker gets in without a password that is why this happens.