Paul Garewal
V00803658

Title:

How we can Inject Cyber Attacks with SQL: A Databased Approach

Abstract:

Lab 7 consisted of exploiting SQL code injection techniques and vulnerabilities in the interface between web applications and database servers. We specifically exploited the vulnerability present when user's inputs are not correctly checked within the web applications before being sent to the back-end database servers.

Aim:

The aim of this lab was to gain first-hand experience on SQL injection techniques. This included exploiting web applications that take inputs from users and how they interact with databases, hence the rationale for the inclusion of implementing SQL in this technique.

Introduction and Background:

Many web applications take inputs from users, and then use these inputs to construct SQL queries to gather information from the database. Web applications also use SQL queries to store information on the database. Furthermore, SQL queries, when not carefully constructed, can cause SQL injection vulnerabilities, which is one of the most common attacks on web applications.

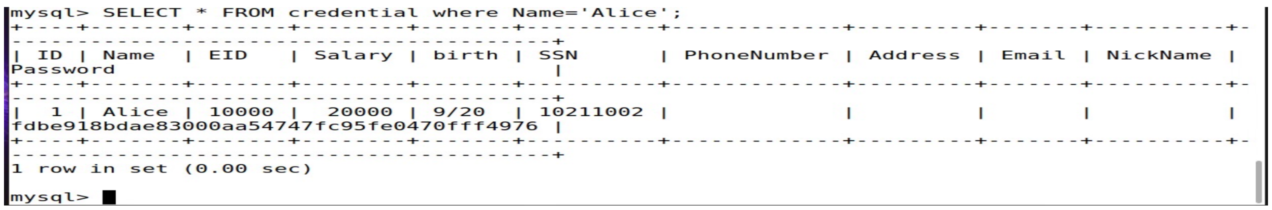
Method:

The method and tools used in the lab revolved around exploiting SQL injections in docker containers. One container will host the web application, while the other will host the database. Additionally, we utilized MySQL database to set up these SQL injections. Further, we used SQL queries on the web application that holds personal information about employees. This information was gathered by injecting SQL queries to the login page of the web application, which stores all of the employee information. Our job, as the attacker, was to log into the web application without knowing any employee's credentials, from the login page and without the login page in the command line.

Results and Discussion:

To get warmed up with SQL statements we were able to access the user's database and gather Alice's information with the SELECT statement.

(SCREENSHOT 1: EMPLOYEE INFORMATION)

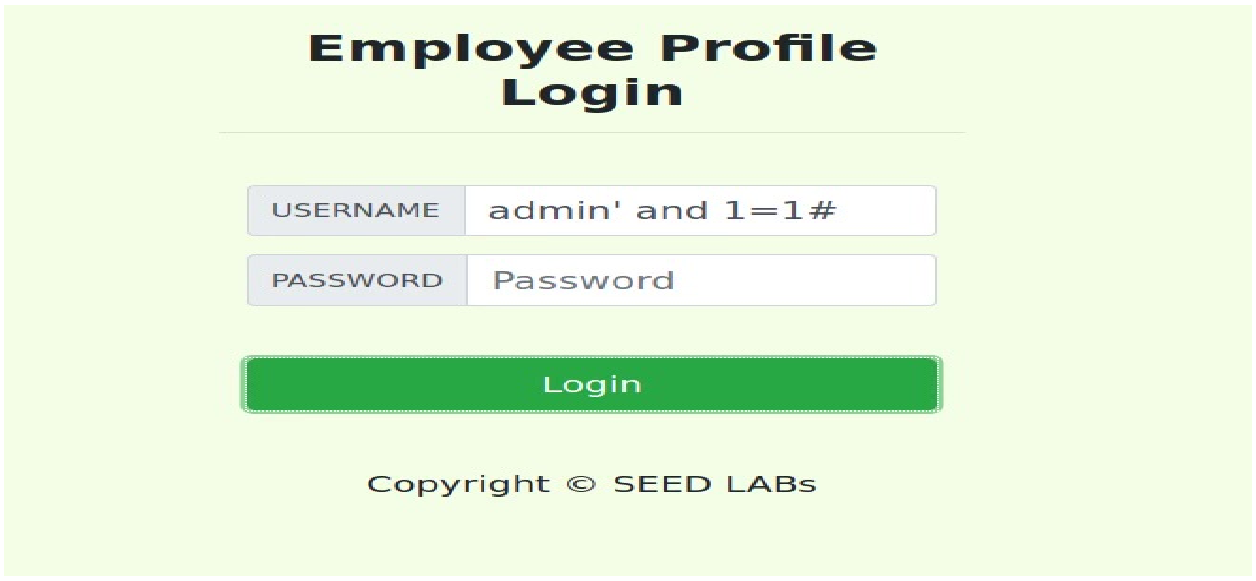


```
mysql> SELECT * FROM credential where Name='Alice';
+----+-------+-------+--------+-------+----------+-------------+---------+-------+----------+-
-----------------------------------------+
| ID | Name  | EID   | Salary | birth | SSN      | PhoneNumber | Address | Email | NickName |
Password                                 |
+----+-------+-------+--------+-------+----------+-------------+---------+-------+----------+-
-----------------------------------------+
|  1 | Alice | 10000 |  20000 | 9/20  | 10211002 |             |         |       |          |
fdbe918bdae83000aa54747fc95fe0470fff4976 |
+----+-------+-------+--------+-------+----------+-------------+---------+-------+----------+-
-----------------------------------------+
1 row in set (0.00 sec)

mysql> 
```

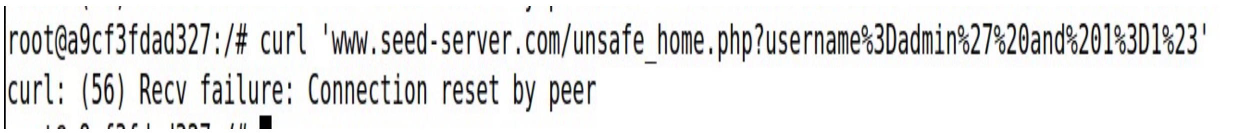
The initial attack for this lab involved using the SELECT statement from SQL and the login page. We were able to login only knowing the username of the employee, which we set to admin`. Then providing an always true SELECT statement that 1=1, we were able to fool the system into thinking we knew the password. This trick worked extremely well and provides further exploration into how we gain that password and use other SELECT statements to gain access.

(SCREENSHOT 2: LOGIN PAGE SQL)



Our last attack involved using the previous attack, however from the command line. Unfortunately we were unable to perform these attacks, similar to other members of our lab class.

(SCREENSHOT 3: FAILED COMMAND LINE SQL ATTACK)



```
root@a9cf3fdad327:/# curl 'www.seed-server.com/unsafe_home.php?username%3Dadmin%27%20and%201%3D1%23'
curl: (56) Recv failure: Connection reset by peer
```

Question answer:

We cannot use two SQL statements because of the countermeasures put in place. This is because “The API functions mysqli::query() and mysqli::real_query() do not set a connection flag necessary for activating multi queries in the server. An extra API call is used for multiple statements to reduce the damage of accidental SQL injection attacks.” [1]. Therefore, it requires special handling when using multiple SQL statements.

References:

[1] “Multiple statements - manual,” *php*. [Online]. Available: https://www.php.net/manual/en/mysqli.quickstart.multiple-statement.php. [Accessed: 04-Nov-2022].