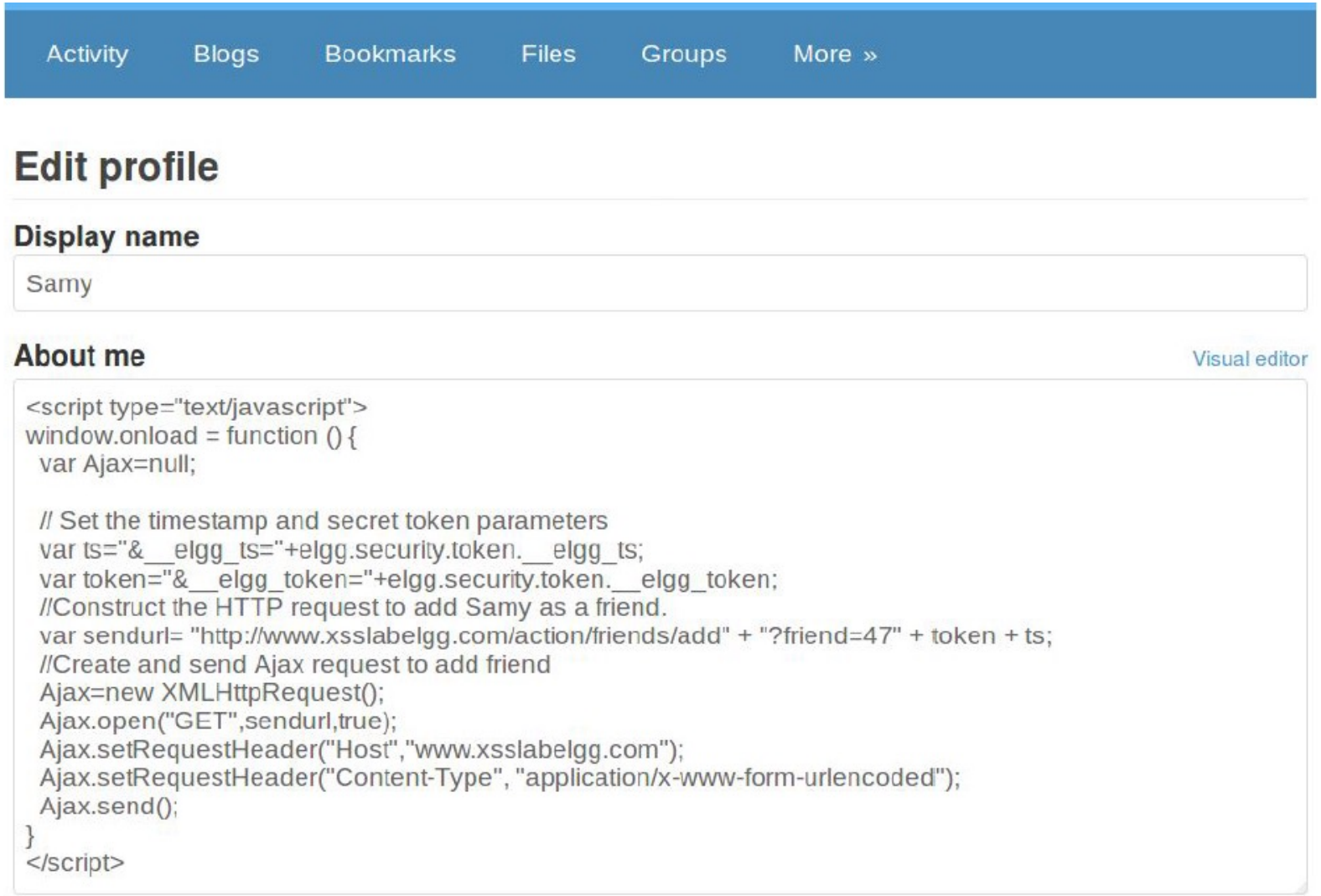
CS336 Lab 4

SQL Injection Attack

Code injection attack

Code Injection is the general term for attack types which consist of injecting code that is then interpreted/executed by the application.

Cause:
Poor handling of untrusted input.
Lack of input validation.



SQL injection

- SQL injection is a code injection technique that exploits the vulnerabilities in the interface between web applications and database servers.
- SQL (Structured Query Language)
  - a standard language for interacting with a relational database management system (RDBMS).
- The vulnerability is present when user's inputs are not correctly checked within the web applications before sending to the back-end database servers.
- Many web applications take inputs from users, and then use these inputs to construct SQL queries, so the web applications can:
  - pull the information out of the database
  - store information in the database.

Outline

- Task 1: Get familiar with SQL statements
- Task 2: SQL Injection Attack on SELECT Statement
- Task 3: SQL Injection Attack on UPDATE Statement

Brief Tutorial of SQL

- **Log in to MySQL:** We will use MySQL database, which is an open-source relational database management system. We can log in using the following command:

```
$ mysql -uroot -pseedubuntu
Welcome to the MySQL monitor.
...
mysql>
```

- **Create a Database**: Inside MySQL, we can create multiple databases. “SHOW DATABSES” command can be used to list existing databases. We will create a new database called dbtest:

```
mysql> SHOW DATABASES;
.......
mysql> CREATE DATABASE dbtest;
```

SQL Tutorial: Create a Table

- A relational database organizes its data using tables. Let us create a table called employee with seven attributes (i.e. columns) for the database “dbtest”

- We need to let the system know which database to use as there may be multiple databases

```
USE dbtest
```

- After a table is created, we can use describe to display the structure of the table

```
DESCRIBE employee;
```

```
mysql> USE dbtest
mysql> CREATE TABLE employee (
  ID        INT (6) NOT NULL AUTO_INCREMENT,
  Name      VARCHAR (30) NOT NULL,
  EID       VARCHAR (7) NOT NULL,
  Password VARCHAR (60),
  Salary    INT (10),
  SSN       VARCHAR (11),
  PRIMARY KEY (ID)
);
mysql> DESCRIBE employee;
+----------+-------------+------+-----+---------+----------------+
| Field    | Type        | Null | Key | Default | Extra          |
+----------+-------------+------+-----+---------+----------------+
| ID       | int(6)      | NO   | PRI | NULL    | auto_increment |
| Name     | varchar(30) | NO   |     | NULL    |                |
| EID      | varchar(30) | NO   |     | NULL    |                |
| Password | varchar(60) | YES  |     | NULL    |                |
| Salary   | int(10)     | YES  |     | NULL    |                |
| SSN      | varchar(11) | YES  |     | NULL    |                |
+----------+-------------+------+-----+---------+----------------+
```

SQL Tutorial: Insert a Row

- We can use the INSERT INTO statement to insert a new record into a table :

```
mysql> INSERT INTO employee (Name, EID, Password, Salary, SSN)
       VALUES ('Ryan Smith', 'EID5000', 'paswd123', 80000,
               '555-55-5555');
```

- Here, we insert a record into the “employee” table.

- We do not specify a value of the ID column, as it will be automatically set by the database.

SQL Tutorial: SELECT Statement

- The **SELECT** statement is the most common operation on databases
- It retrieves information from a database

```
mysql> SELECT * FROM employee;
+----+---------+---------+----------+--------+-------------+
| ID | Name    | EID     | Password | Salary | SSN         |
+----+---------+---------+----------+--------+-------------+
|  1 | Alice   | EID5000 | paswd123 |  80000 | 555-55-5555 |
|  2 | Bob     | EID5001 | paswd123 |  80000 | 555-66-5555 |
|  3 | Charlie | EID5002 | paswd123 |  80000 | 555-77-5555 |
|  4 | David   | EID5003 | paswd123 |  80000 | 555-88-5555 |
+----+---------+---------+----------+--------+-------------+
mysql> SELECT Name, EID, Salary FROM employee;
+---------+---------+--------+
| Name    | EID     | Salary |
+---------+---------+--------+
| Alice   | EID5000 |  80000 |
| Bob     | EID5001 |  80000 |
| Charlie | EID5002 |  80000 |
| David   | EID5003 |  80000 |
+---------+---------+--------+
```

Asks the database for all its records, including all the columns

Asks the database only for Name, EID and Salary columns

SQL Tutorial: WHERE Clause

- It is uncommon for a SQL query to retrieve all records in a database.
- WHERE clause is used to set conditions for several types of SQL statements including SELECT, UPDATE, DELETE etc.

```
mysql> SQL Statement
       WHERE predicate;
```

- The above SQL statement only reflects the rows for which the predicate in the WHERE clause is TRUE.
- The predicate is a logical expression; multiple predicates can be combined using keywords **AND** and **OR**.
- Lets look at an example in the next slide.

SQL Tutorial: WHERE Clause

```
mysql> SELECT * FROM employee WHERE EID='EID5001';
+----+------+---------+----------+--------+-------------+
| ID | Name | EID     | Password | Salary | SSN         |
+----+------+---------+----------+--------+-------------+
|  2 | Bob  | EID5001 | paswd123 |  80000 | 555-66-5555 |
+----+------+---------+----------+--------+-------------+

mysql> SELECT * FROM employee WHERE EID='EID5001' OR Name='David';
+----+-------+---------+----------+--------+-------------+
| ID | Name  | EID     | Password | Salary | SSN         |
+----+-------+---------+----------+--------+-------------+
|  2 | Bob   | EID5001 | paswd123 |  80000 | 555-66-5555 |
|  4 | David | EID5003 | paswd123 |  80000 | 555-88-5555 |
+----+-------+---------+----------+--------+-------------+
```

- The first query returns a record that has EID5001 in EID field
- The second query returns the records that satisfy either EID='EID5001' or Name='David'

SQL Tutorial: WHERE Clause

- If the condition is always True, then all the rows are affected by the SQL statement

```
mysql> SELECT * FROM employee WHERE 1=1;
+----+---------+---------+----------+--------+-------------+
| ID | Name    | EID     | Password | Salary | SSN         |
+----+---------+---------+----------+--------+-------------+
|  1 | Alice   | EID5000 | paswd123 |  80000 | 555-55-5555 |
|  2 | Bob     | EID5001 | paswd123 |  80000 | 555-66-5555 |
|  3 | Charlie | EID5002 | paswd123 |  80000 | 555-77-5555 |
|  4 | David   | EID5003 | paswd123 |  80000 | 555-88-5555 |
+----+---------+---------+----------+--------+-------------+
```

- This 1=1 predicate looks quite useless in real queries, but it will become useful in SQL Injection attacks

SQL Tutorial: UPDATE Statement

- We can use the UPDATE Statement to modify an existing record.

```
mysql> UPDATE employee SET Salary=82000 WHERE Name='Bob';
mysql> SELECT * FROM employee WHERE Name='Bob';
+----+------+---------+----------+--------+--------------+
| ID | Name | EID     | Password | Salary | SSN          |
+----+------+---------+----------+--------+--------------+
|  2 | Bob  | EID5001 | paswd123 |  82000 | 555-66-5555 |
+----+------+---------+----------+--------+--------------+
```

SQL Tutorial: Comments

MySQL supports three **comment** styles

- Text from the # character to the end of line is treated as a comment
- Text from the “--” to the end of line is treated as a comment.
- Similar to C language, text between /* and */ is treated as a comment

```
mysql> SELECT * FROM employee;    # Comment to the end of line
mysql> SELECT * FROM employee;    -- Comment to the end of line
mysql> SELECT * FROM /* In-line comment */ employee;
```

In this lab

- database -- **Users**
- table – **credential**
- There are mainly two roles in this web application:
- Administrator is a privilege role and can manage each individual employees' profile information;
- Employee is a normal role and can view or update his/her own profile information.

Name	Employee ID	Password	Salary	Birthday	SSN	Nickname	Email	Address	Phone#
Admin	99999	seedadmin	400000	3/5	43254314				
Alice	10000	seedalice	20000	9/20	10211002				
Boby	20000	seedboby	50000	4/20	10213352				
Ryan	30000	seedryan	90000	4/10	32193525				
Samy	40000	seedsamy	40000	1/11	32111111				
Ted	50000	seedted	110000	11/3	24343244				

Task 1: get familiar with SQL statements

- In MySQL:

➢ 1.1 Use a SQL command to print all the profile information of the employee `Alice`.

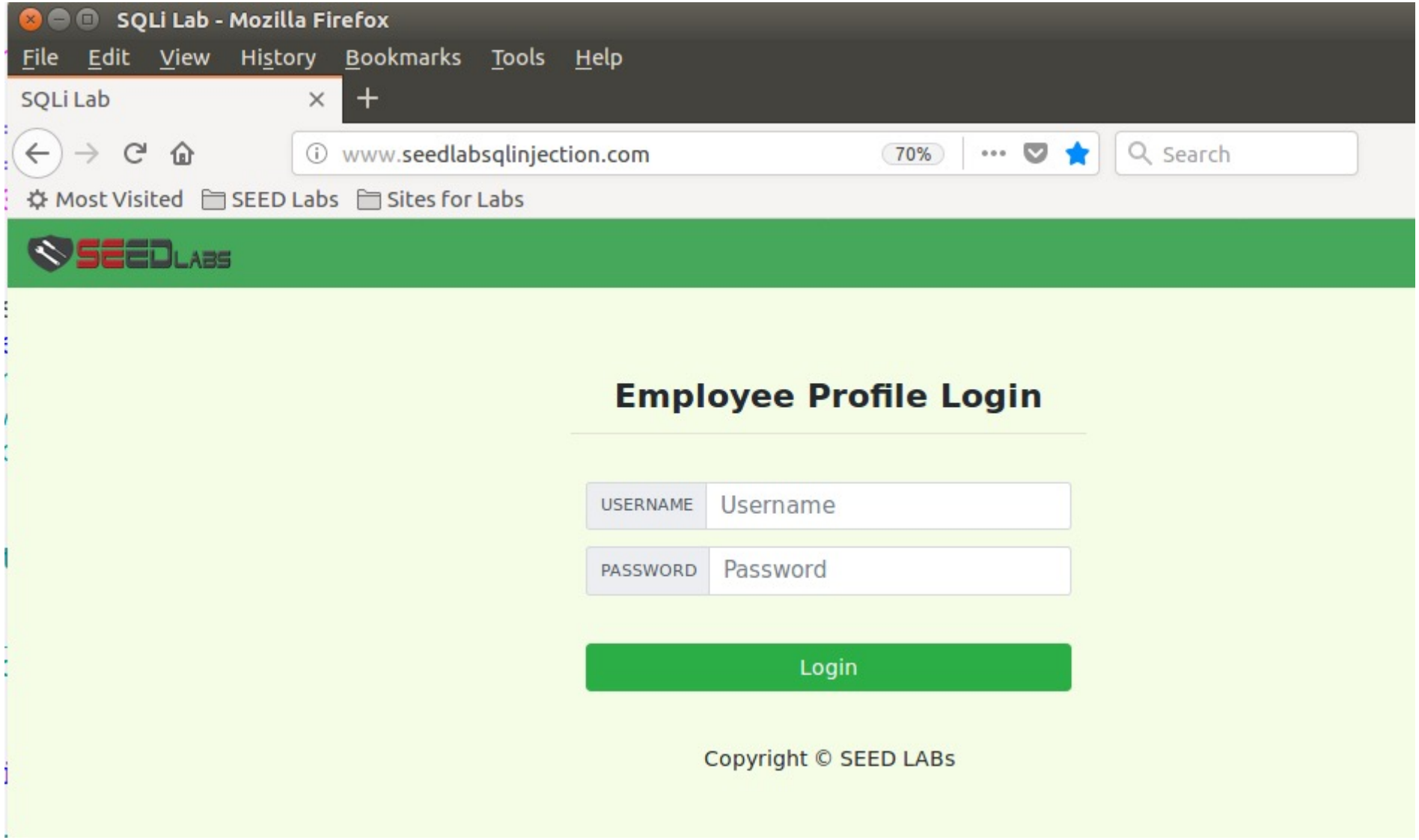
➢ 1.2 Use a SQL command to print any three columns of the employee `Boby`. (eg. name, ssn, salary columns)

➢ 1.3 Use a SQL command to print all the profile information of the employees `Samy` and `Ted`.

- Please provide the screenshot of your results.

Outline

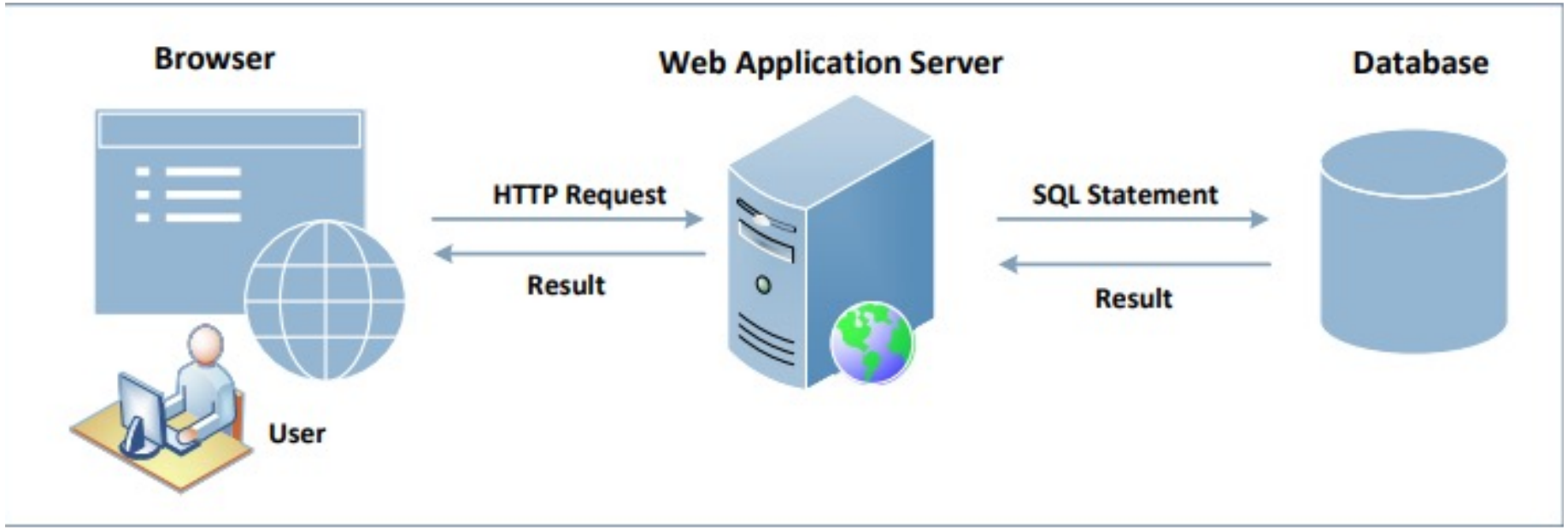
- Task 1: Get familiar with SQL statements
- Task 2: SQL Injection Attack on SELECT Statement
- Task 3: SQL Injection Attack on UPDATE Statement



- login page: www.seedlabsqlinjection.com

Interacting with Database in Web Application

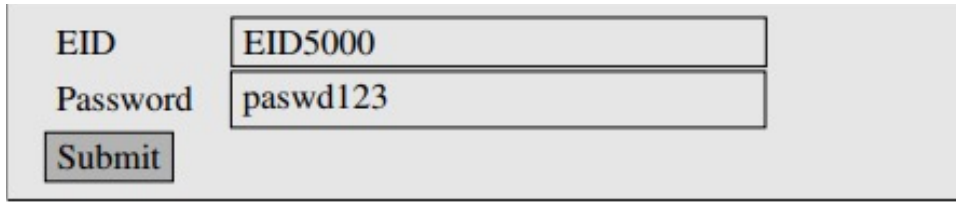
- A typical web application consists of three major components:



- SQL Injection attacks can cause damage to the database. As we notice in the figure, the users do not directly interact with the database but through a web server. If this channel is not implemented properly, malicious users can attack the database.

Getting Data from User

- This example shows a form where users can type their data. Once the submit button is clicked, an HTTP request will be sent out with the data attached



- The HTML source of the above form is given below:

```
<form action="getdata.php" method="get">
  EID:      <input type="text" name="EID"><br>
  Password: <input type="text" name="Password"><br>
            <input type="submit" value="Submit">
</form>
```

- Request generated is:

```
http://www.example.com/getdata.php?EID=EID5000&Password=paswd123
```

Getting Data from User

- The request shown is an HTTP GET request, because the method field in the HTML code specified the get type
- In GET requests, parameters are attached after the question mark in the URL
- Each parameter has a name=value pair and are separated by “&”

```
http://www.example.com/getdata.php?EID=EID5000&Password=paswd123
```

- In the case of HTTPS, the format would be similar but the data will be encrypted
- Once this request reached the target PHP script the parameters inside the HTTP request will be saved to an array $_GET or $_POST. The following example shows a PHP script getting data from a GET request

```
<?php
   $eid = $_GET['EID'];
   $pwd = $_GET['Password'];
   echo "EID: $eid --- Password: $pwd\n";
?>
```

How Web Applications Interact with Database

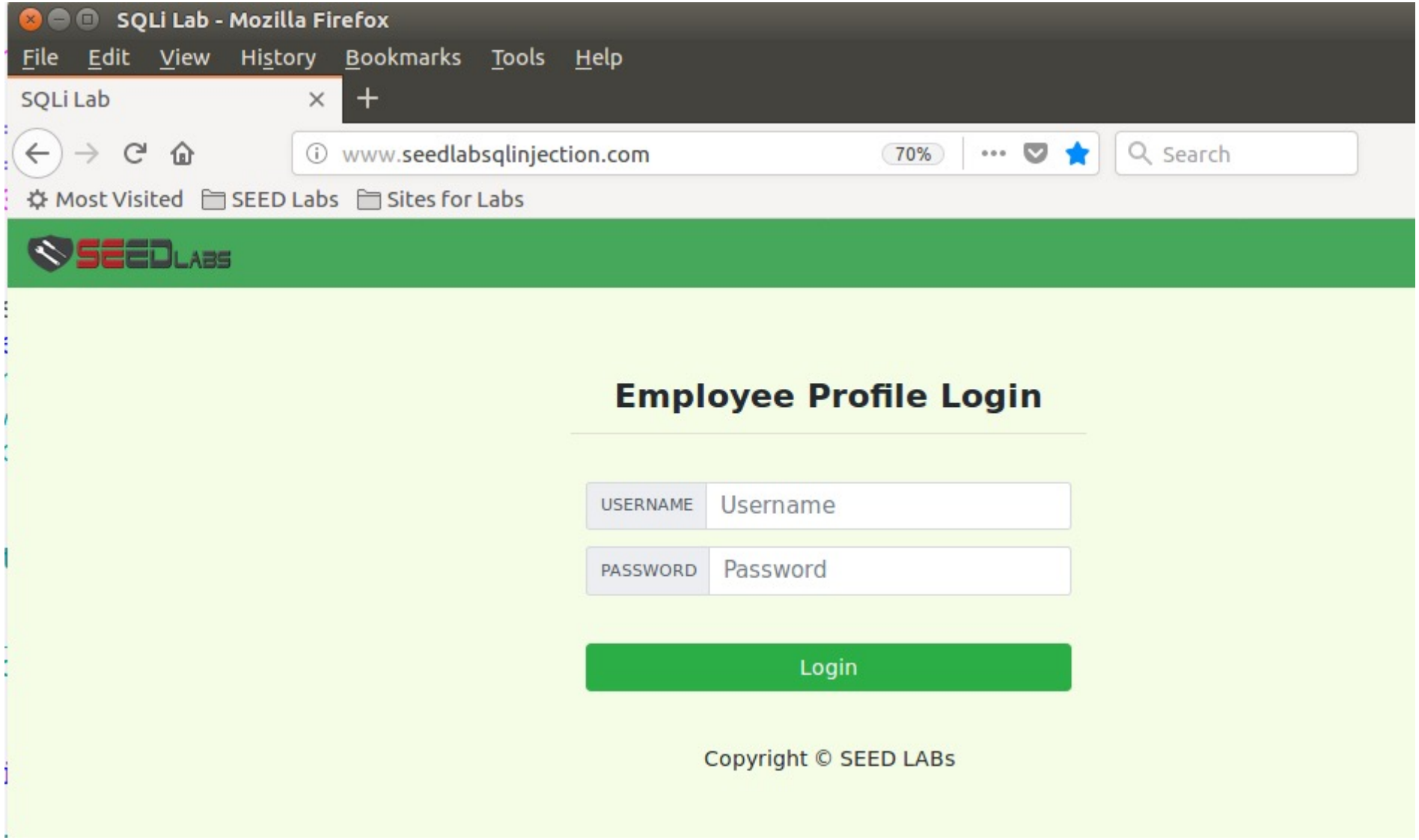
- Construct the query string and then send it to the database for execution.
- The channel between user and database creates a new attack surface for the database.

```
/* getdata.php */
<?php
   $eid = $_GET['EID'];
   $pwd = $_GET['Password'];

   $conn = new mysqli("localhost", "root", "seedubuntu", "dbtest");
   $sql = "SELECT Name, Salary, SSN
           FROM employee
           WHERE eid= '$eid' and password='$pwd'";

   $result = $conn->query($sql);
   if ($result) {
      // Print out the result
      while ($row = $result->fetch_assoc()) {
        printf ("Name: %s -- Salary: %s -- SSN: %s\n",
                $row["Name"], $row["Salary"], $row['SSN']);
      }
      $result->free();
   }
   $conn->close();
?>
```

Constructing SQL statement



- login page: www.seedlabsqlinjection.com

unsafe_home.php

- User authentication

```php
session_start();
// if the session is new extract the username password from the GET request
$input_uname = $_GET['username'];
$input_pwd = $_GET['Password'];
$hashed_pwd = sha1($input_pwd);
```

```php
// Sql query to authenticate the user
$sql = "SELECT id, name, eid, salary, birth, ssn, phoneNumber, address,
email,nickname,Password
FROM credential
WHERE name= '$input_uname' and Password='$hashed_pwd'";
```

unsafe_home.php located in the var/www/SQLInjection/

- **Task 2.1: SQL Injection Attack from webpage**. Your task is to log into the web application as the administrator from the login page, so you can see the information of all the employees. We assume that you do know the administrator's account name which is `admin`, but you do not the password. You need to decide what to type in the `Username` and `Password` fields to succeed in the attack.

```
// Sql query to authenticate the user
$sql = "SELECT id, name, eid, salary, birth, ssn, phoneNumber, address,
email,nickname,Password
FROM credential
WHERE name= '$input_uname' and Password='$hashed_pwd'";
```

```
// Sql query to authenticate the user
$sql = "SELECT id, name, eid, salary, birth, ssn, phoneNumber, address,
email,nickname,Password
FROM credential
WHERE name= '                ' and Password='$hashed_pwd'";
```

Text from the **#** character to the end of line is treated as a comment

- **Task 2.2: SQL Injection Attack from command line**. Your task is to repeat Task 2.1, but you need to do it without using the webpage. You can use command line tools, such as `curl`, which can send HTTP requests. One thing that is worth mentioning is that if you want to include multiple parameters in HTTP requests, you need to put the URL and the parameters between a pair of single quotes; otherwise, the special characters used to separate parameters (such as `&`) will be interpreted by the shell program, changing the meaning of the command. The following example shows how to send an HTTP GET request to our web application, with two parameters (`username` and `Password`) attached:

Getting Data from User

- The request shown is an HTTP GET request, because the method field in the HTML code specified the get type
- In GET requests, parameters are attached after the **question mark** in the URL
- Each parameter has a name=value pair and are separated by “&”

```
http://www.example.com/getdata.php?EID=EID5000&Password=paswd123
```

- In the case of HTTPS, the format would be similar but the data will be encrypted
- Once this request reached the target PHP script the parameters inside the HTTP request will be saved to an array $_GET or $_POST. The following example shows a PHP script getting data from a GET request

```
<?php
   $eid = $_GET['EID'];
   $pwd = $_GET['Password'];
   echo "EID: $eid --- Password: $pwd\n";
?>
```

Launching SQL Injection Attacks using cURL

- More convenient to use a command-line tool to launch attacks.
- Easier to automate attacks without a graphic user interface.
- Using cURL, we can send out a form from a command-line, instead of from a web page.

```
% curl 'www.example.com/getdata.php?EID=a' OR 1=1 #&Password='
```

- The above command will not work. In an HTTP request, special characters are in the attached data needs to be encoded or they maybe mis-interpreted.
- In the above URL we need to encode the apostrophe, whitespace and the # sign and the resulting cURL command is as shown below:

```
% curl 'www.example.com/getdata.php?EID=a%27%20
                                    OR%201=1%20%23&Password='
Name: Alice -- Salary: 80000 -- SSN: 555-55-5555<br>
Name: Bob -- Salary: 82000 -- SSN: 555-66-5555<br>
Name: Charlie -- Salary: 80000 -- SSN: 555-77-5555<br>
Name: David -- Salary: 80000 -- SSN: 555-88-5555<br>
```

- **Task 2.2: SQL Injection Attack from command line.** Your task is to repeat Task 2.1, but you need to do it without using the webpage. You can use command line tools, such as `curl`, which can send HTTP requests. One thing that is worth mentioning is that if you want to include multiple parameters in HTTP requests, you need to put the URL and the parameters between a pair of single quotes; otherwise, the special characters used to separate parameters (such as `&`) will be interpreted by the shell program, changing the meaning of the command. The following example shows how to send an HTTP GET request to our web application, with two parameters (`username` and `Password`) attached:

- For example, to use curl
- curl 'www.SeedLabSQLInjection.com/unsafe_home.php?username=alice&Password=seedalice'

single quote ' -- %27
white space -- %20
pound sign # -- %23
semicolon ; -- %3b

- **Task 2.3: Append a new SQL statement**. In the above two attacks, we can only steal information from the database; it will be better if we can modify the database using the same vulnerability in the login page. An idea is to use the SQL injection attack to turn one SQL statement into two, with the second one being the update or delete statement. In SQL, semicolon (;) is used to separate two SQL statements. Please describe how you can use the login page to get the server run two SQL statements. Try the attack to delete a record from the database, and describe your observation.

Hint: You may not be able to do it. Just describe your observation.

Outline

- Task 1: Get familiar with SQL statements
- Task 2: SQL Injection Attack on SELECT Statement
- Task 3: SQL Injection Attack on UPDATE Statement

Task 3: SQL Injection Attack on UPDATE statement

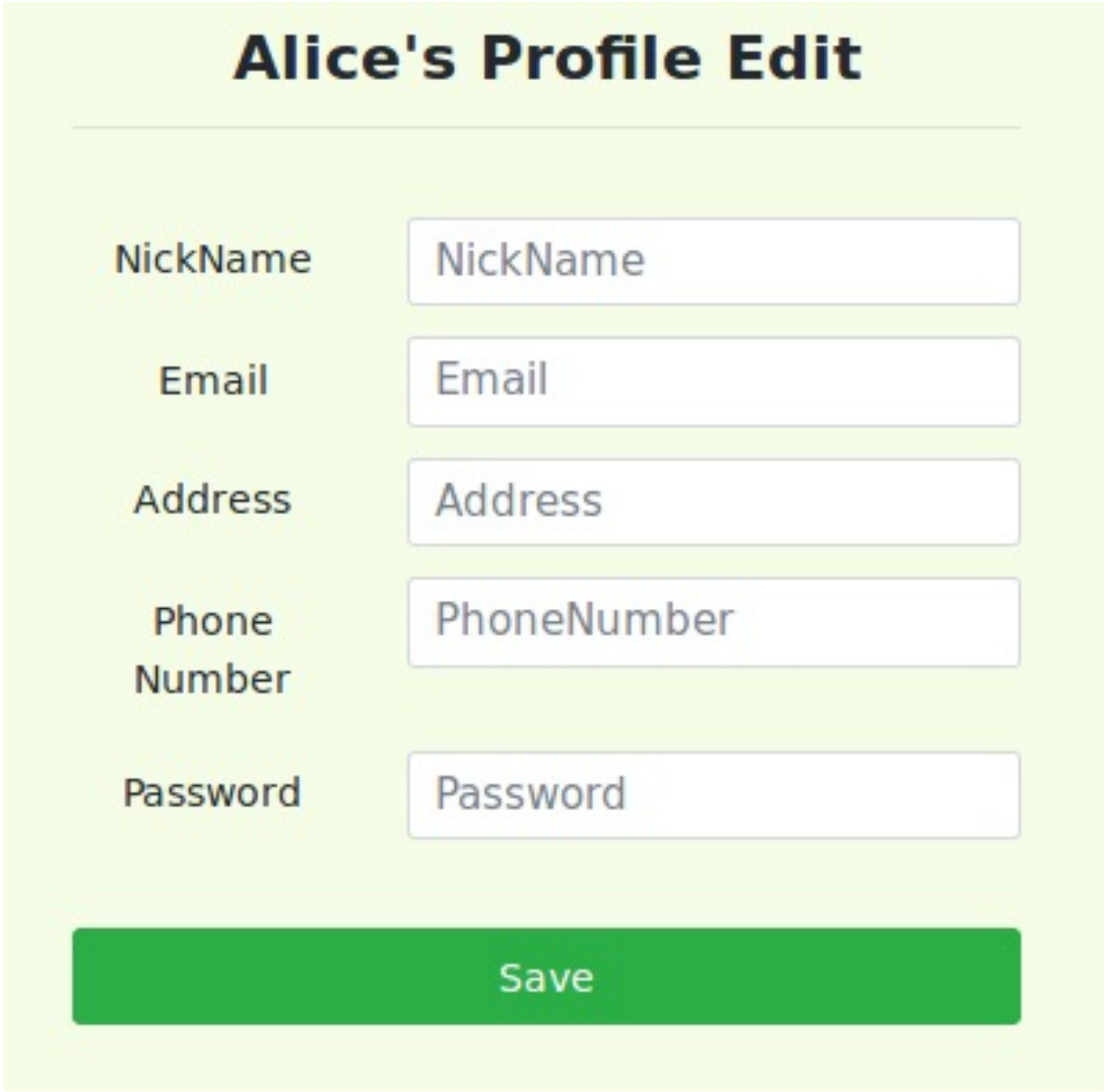
- If a SQL injection vulnerability happens to an UPDATE statement, the damage will be more severe, because attackers can use the vulnerability to modify databases.
- In our Employee Management application, there is an Edit Profile page that allows employees to update their profile information, including nickname, email, address, phone number, and password.
- To go to this page, employees need to log in first.
  - Let's login as Alice. (username: alice, password: seedalice)

- Task 3.1: Modify your own salary. As shown in the Edit Profile page, employees can only update their nicknames, emails, addresses, phone numbers, and passwords; they are not authorized to change their salaries. Assume that you (Alice) are a disgruntled employee, and your boss Boby did not increase your salary this year. You want to increase your own salary by exploiting the SQL injection vulnerability in the Edit-Profile page. Please demonstrate how you can achieve that. We assume that you do know that salaries are stored in a column called `salary`.
- Task 3.2: Modify other people' salary. After increasing your own salary, you decide to punish your boss Boby. You want to reduce his salary to 1 dollar. Please demonstrate how you can achieve that.

Name	Employee ID	Password	Salary	Birthday	SSN	Nickname	Email	Address	Phone#
Admin	99999	seedadmin	400000	3/5	43254314				
Alice	10000	seedalice	20000	9/20	10211002				
Boby	20000	seedboby	50000	4/20	10213352				
Ryan	30000	seedryan	90000	4/10	32193525				
Samy	40000	seedsamy	40000	1/11	32111111				
Ted	50000	seedted	110000	11/3	24343244				

Edit profile page

After you login, you can edit your profile.



- When employees update their information through the Edit Profile page, the following **SQL UPDATE** query will be executed.
- The PHP code implemented in unsafe_edit_backend.php file is used to update employee's profile information.
- The PHP file is located in the /var/www/SQLInjection directory.

```php
<?php
session_start();
$input_email = $_GET['Email'];
$input_nickname = $_GET['NickName'];
$input_address= $_GET['Address'];
$input_pwd = $_GET['Password'];
$input_phonenumber = $_GET['PhoneNumber'];
$uname = $_SESSION['name'];
$eid = $_SESSION['eid'];
$id = $_SESSION['id'];
```

construct query to update info

```
// Don't do this, this is not safe against SQL injection attack
$sql="";
if($input_pwd!=''){
  // In case password field is not empty.
  $hashed_pwd = sha1($input_pwd);
  //Update the password stored in the session.
  $_SESSION['pwd']=$hashed_pwd;
  $sql = "UPDATE credential SET
nickname='$input_nickname',email='$input_email',address='$input_address',Password='$hashed_pwd',
where ID=$id;";
}else{
  // if passowrd field is empty.
  $sql = "UPDATE credential SET
nickname='$input_nickname',email='$input_email',address='$input_address',PhoneNumber='$input_pho
where ID=$id;";
}
$conn->query($sql);
```

unsafe_edit_backend.php

```
    $sql = "UPDATE credential SET
nickname='$input_nickname',email='$input_email',address='$input_address',Password='$hashed_pwd',
where ID=$id;";
```

- $sql = "UPDATE credential SET nickname='$input_nickname',email='$input_email',address='$input_address',Password='$hashed_pwd',PhoneNumber='$input_phonenumber' where ID=$id;";

- UPDATE credential SET salary='99998' where name='alice';
- UPDATE credential SET nickname=' ' ,email=... where ID=$id;

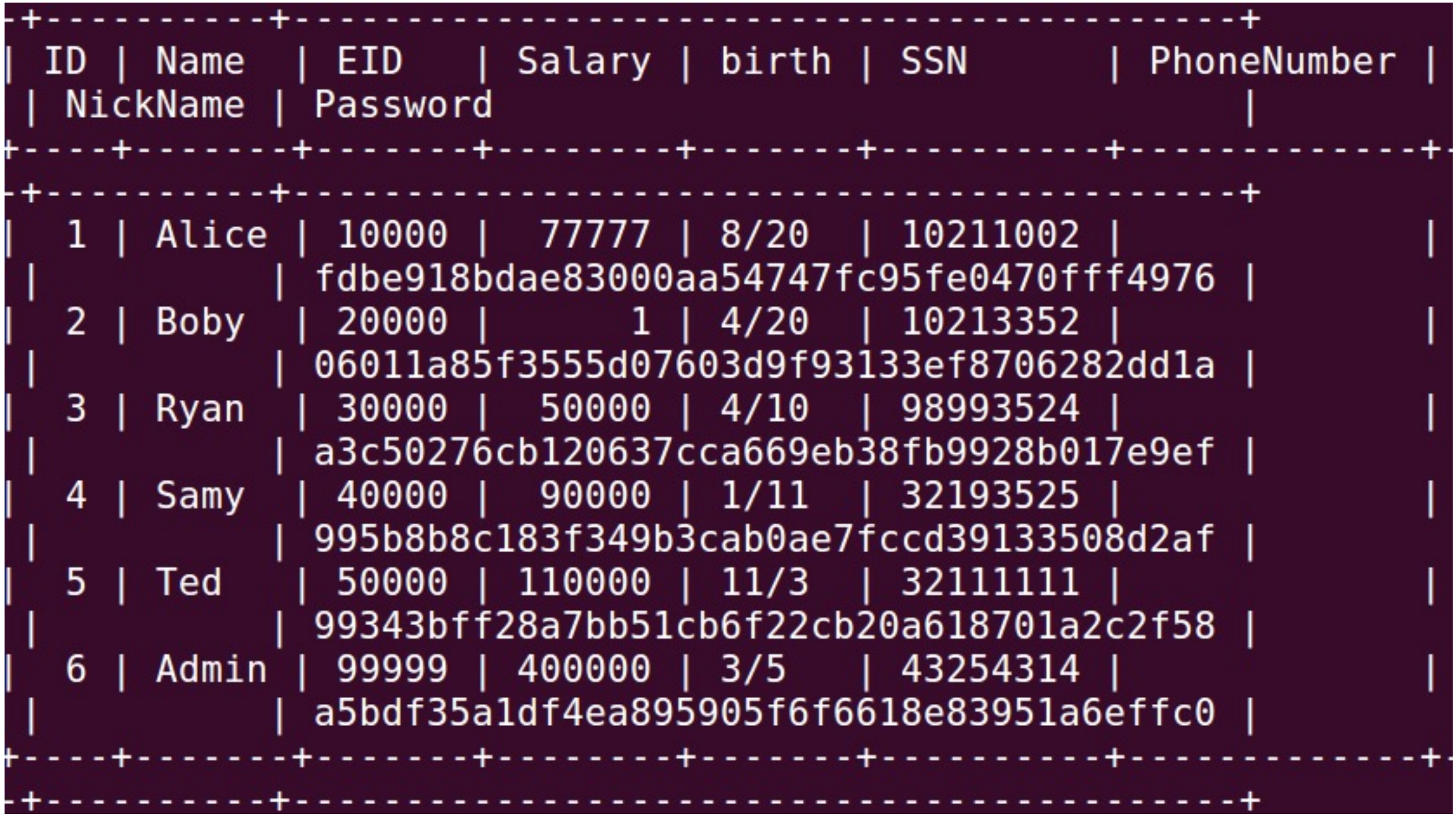
- Task 3.1: Modify your own salary. As shown in the Edit Profile page, employees can only update their nicknames, emails, addresses, phone numbers, and passwords; they are not authorized to change their salaries. Assume that you (Alice) are a disgruntled employee, and your boss Boby did not increase your salary this year. You want to increase your own salary by exploiting the SQL injection vulnerability in the Edit-Profile page. Please demonstrate how you can achieve that. We assume that you do know that salaries are stored in a column called `salary`.
- Task 3.2: Modify other people' salary. After increasing your own salary, you decide to punish your boss Boby. You want to reduce his salary to 1 dollar. Please demonstrate how you can achieve that.

- Task 3.3: **Modify other people' password.** After changing Boby's salary, you are still disgruntled, so you want to change Boby's password to something that you know, and then you can log into his account and do further damage. Please demonstrate how you can achieve that. You need to demonstrate that you can successfully log into Boby's account using the new password.
- One thing worth mentioning here is that the database stores the hash value of passwords instead of the plaintext password string. You can again look at the unsafe_edit_backend.php code to see how password is being stored. It uses SHA1 hash function to generate the hash value of password.

construct query to update info

```
// Don't do this, this is not safe against SQL injection attack
$sql="";
if($input_pwd!=''){
  // In case password field is not empty.
  $hashed_pwd = sha1($input_pwd);
  //Update the password stored in the session.
  $_SESSION['pwd']=$hashed_pwd;
  $sql = "UPDATE credential SET
nickname='$input_nickname',email='$input_email',address='$input_address',Password='$hashed_pwd',
where ID=$id;";
}else{
  // if passowrd field is empty.
  $sql = "UPDATE credential SET
nickname='$input_nickname',email='$input_email',address='$input_address',PhoneNumber='$input_pho
where ID=$id;";
}
$conn->query($sql);
```

unsafe_edit_backend.php



ID	Name	EID	Salary	birth	SSN	PhoneNumber	NickName	Password
1	Alice	10000	77777	8/20	10211002			fdbe918bdae83000aa54747fc95fe0470fff4976
2	Boby	20000	1	4/20	10213352			06011a85f3555d07603d9f93133ef8706282dd1a
3	Ryan	30000	50000	4/10	98993524			a3c50276cb120637cca669eb38fb9928b017e9ef
4	Samy	40000	90000	1/11	32193525			995b8b8c183f349b3cab0ae7fccd39133508d2af
5	Ted	50000	110000	11/3	32111111			99343bff28a7bb51cb6f22cb20a618701a2c2f58
6	Admin	99999	400000	3/5	43254314			a5bdf35a1df4ea895905f6f6618e83951a6effc0

How to get the hash value?

- Write a small php code (using nano or other editors) named genPswd.php

```php
<?php
echo sha1("attacker");
echo "\n";
?>
```

- Execute it with command in terminal:

```
php genPswd.php
```

```
[10/25/19]seed@VM:~$ php genPswd.php
52e51cf3f58377b8a687d49b960a58dfc677f0ad
```

- Use the hash value to construct your input on the edit profile webpage.

All tasks for lab 4

- Task 1: Get familiar with SQL statements
- Task 2: SQL Injection Attack on SELECT Statement
- Task 3: SQL Injection Attack on UPDATE Statement