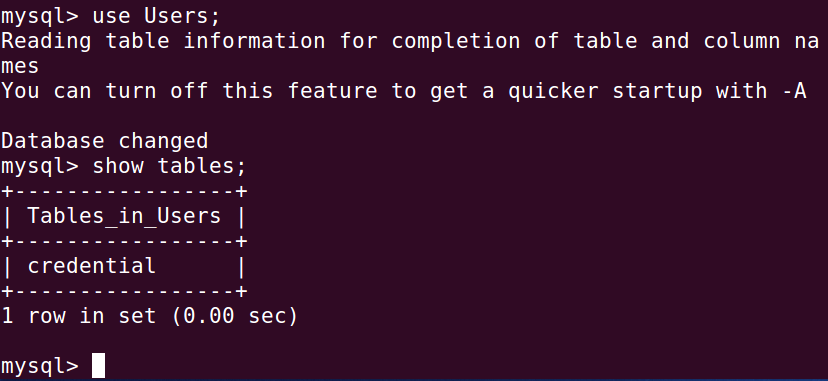
**SQL Injection Attack Lab**

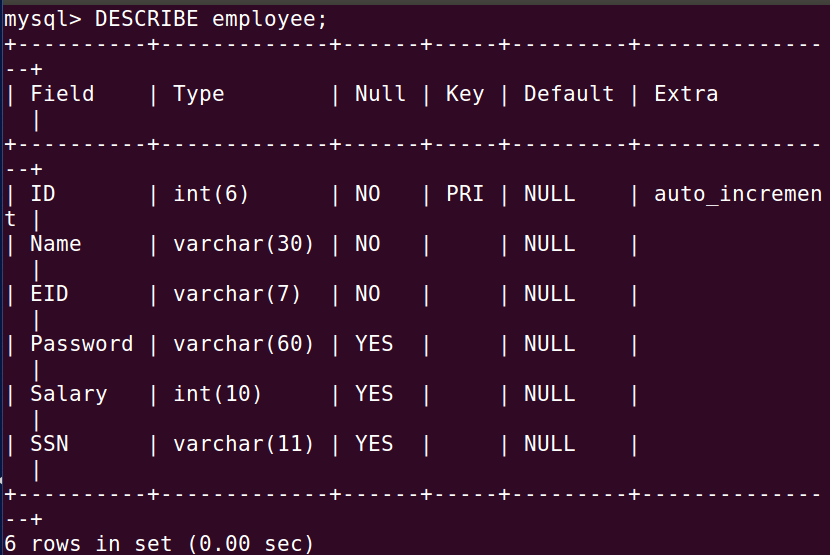
**——57118125施祺**

**Task 1: Get Familiar with SQL Statements**

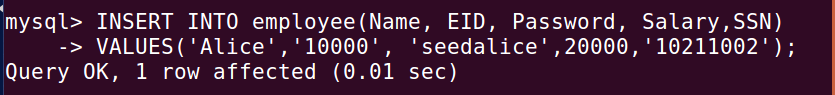
首先进入SQL系统



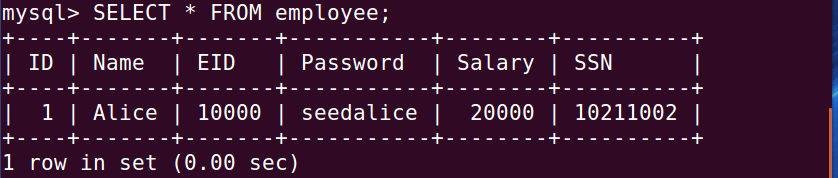
创建一个数据表



输入Alice的信息



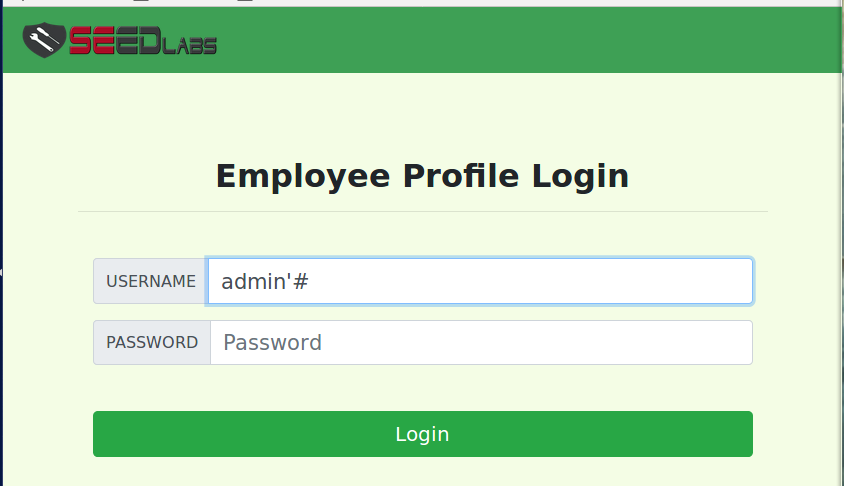
查看，发现Alice数据成功存入

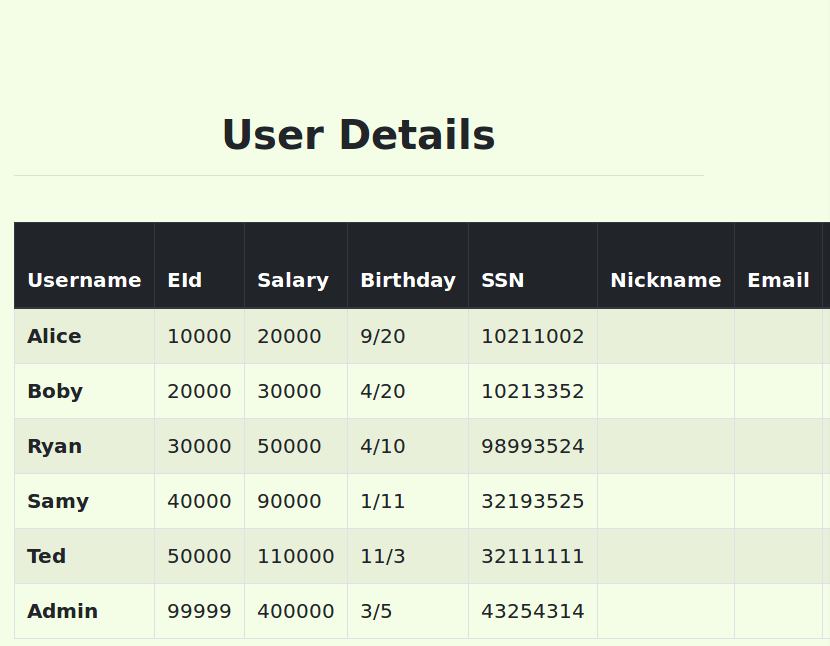


**Task 2: SQL Injection Attack on SELECT Statement**

**Task 2.1**

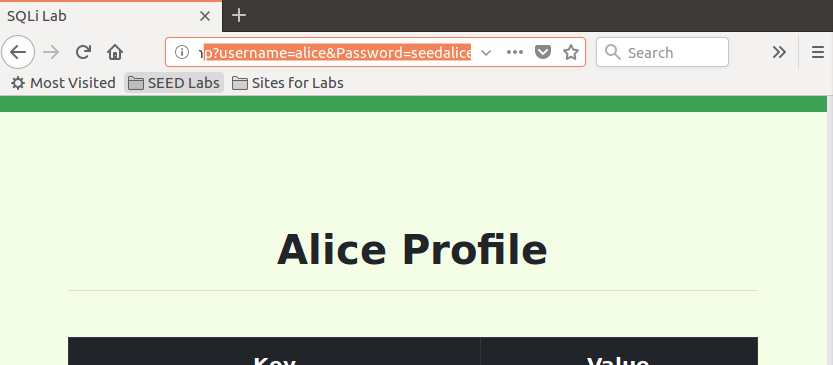
通过‘#将密码注释，便可直接以管理员身份进入



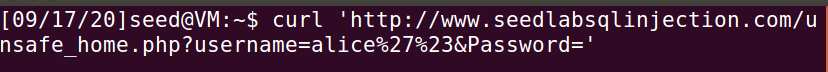


**Task 2.2:**

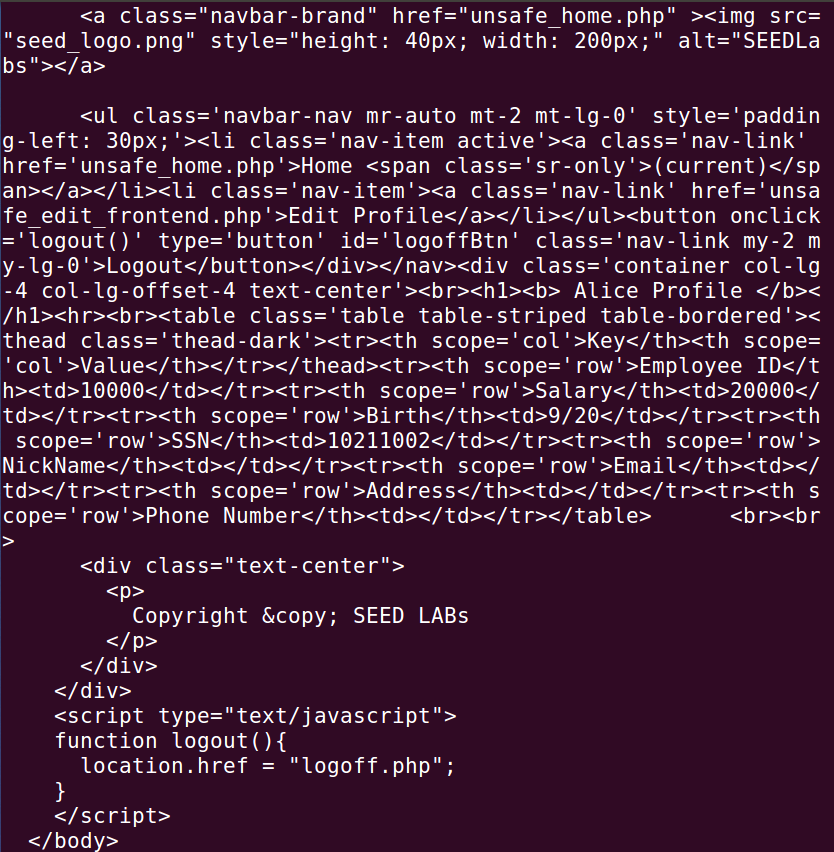
从页面中获取网址，在终端中输入命令行进行SQL注入



URL中会对撇号和空格，#号进行编码，编码之后变为%27，%20，%23

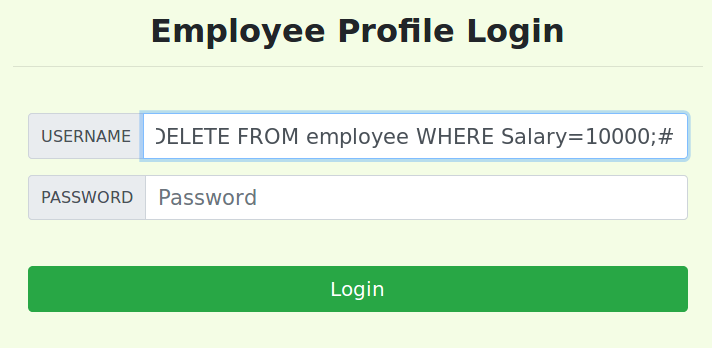


实现结果如下，显示了网页的内容，说明SQL注入成功

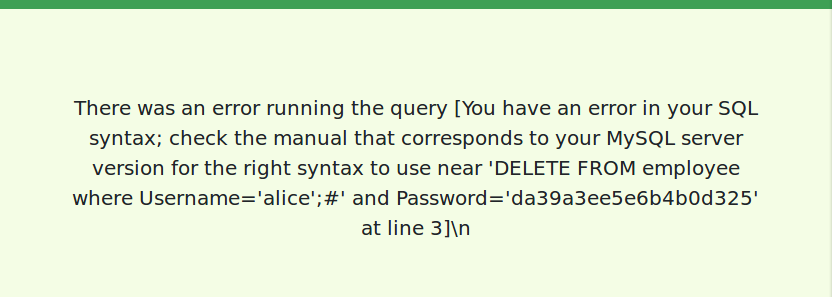


**Task 2.3:**

通过SQL语句以;作结的特性，在一句SQL语句后追加一句新的SQL语句，从而实现执行多条SQL语句的目标。



发现注入失败，原因是因为SQL对注入攻击有一种防护措施，无法执行两条SQL语句



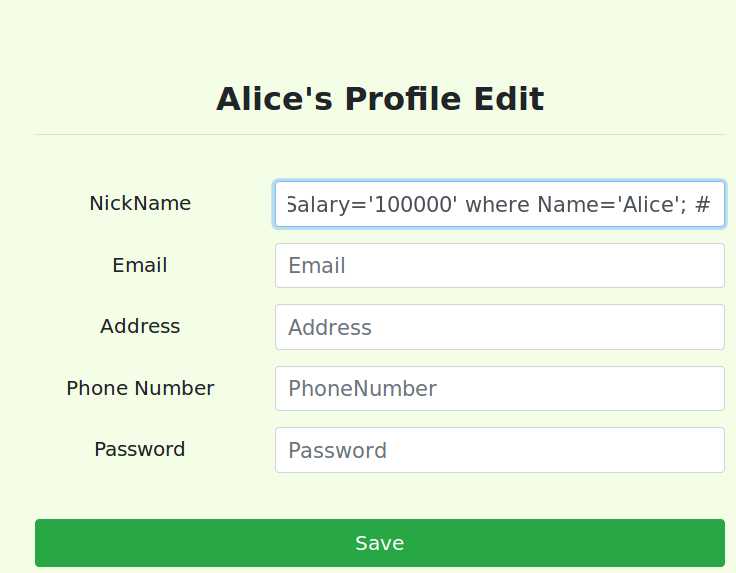
**Task 3: SQL Injection Attack on UPDATE Statement**

**Task 3.1:**

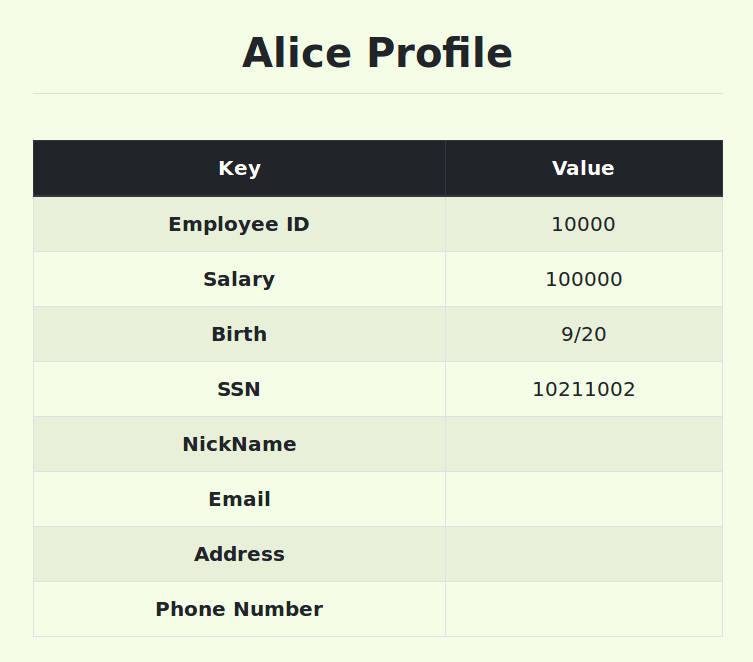
要修改个人资料，首先我们使用Alice的账户登录，进入资料修改界面

类似Task 2的方法，我们观察该功能的源码，发现仍然可以通过输入特殊内容达到修改非权限内资料的目的。

在NickName栏中输入 ',Salary='100000' where Name='Alice'; # ，保存



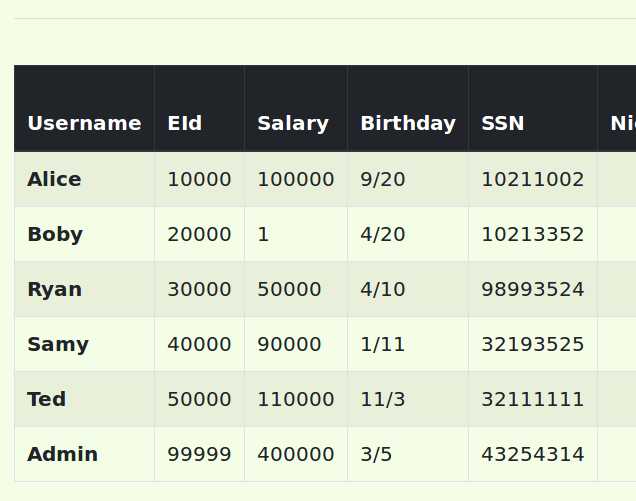
再次登出，我们会发现Alice的薪水已经被修改，攻击成功。



**Task 3.2**

与上一个任务类似，我们只需要更改 WHERE的参数即可

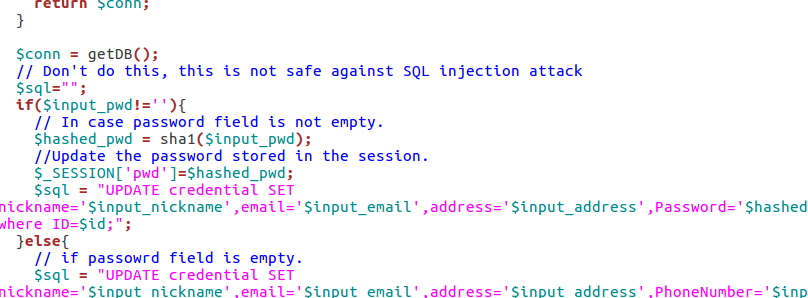
在Nickname一栏中输入 ',Salary='1' where Name='Boby'; # ,保存



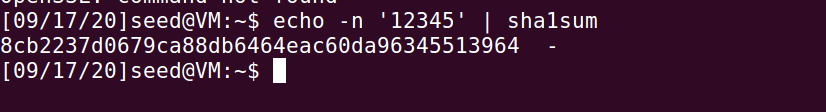
可以看到boby的薪水被成功修改，攻击成功

**Task 3.3**

观察unsafe edit backend.php文件中有关password的描述，发现密码的存储方式并非明文，而是以sha1的格式存储。因此我们如果要将Boby的密码修改为12345，需要知道12345在sha1加密后的值。

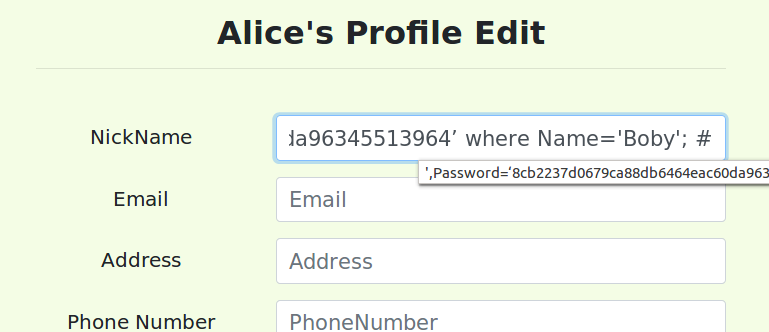


在终端输入命令对明文加密

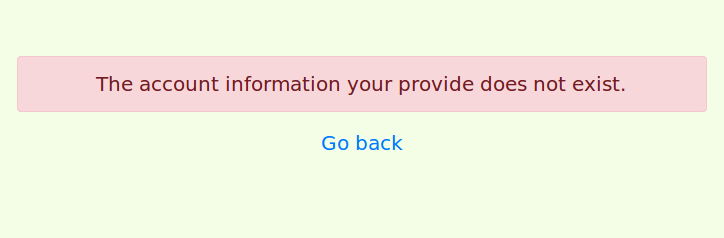


得知明文经过sha1加密后的密文之后，再次修改，在Nickname一栏中输入

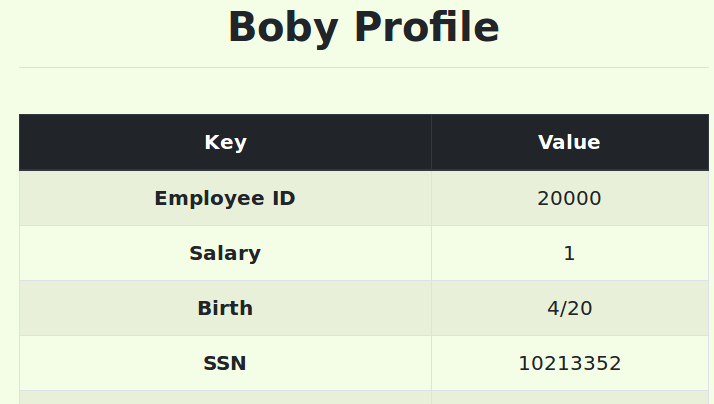
',Password=‘8cb2237d0679ca88db6464eac60da96345513964’ where Name='Boby'; #



使用原密码登录，发现错误



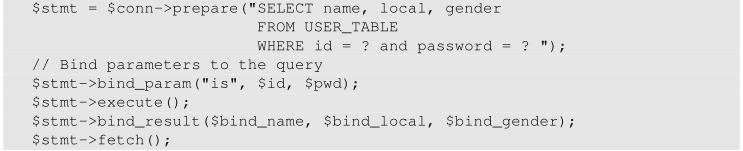
使用12345登录，成功，说明攻击成功。



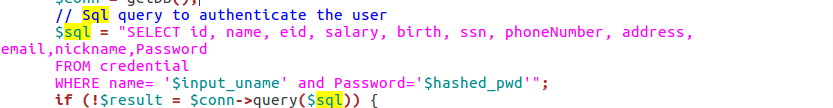
**Task 4: Countermeasure — Prepared Statement**

导致SQL注入攻击的根本原因是SQL将数据与代码混合。因此，防范SQL注入攻击的最佳策略是把数据和代码分开。

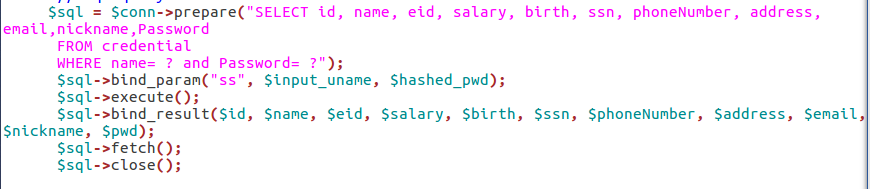
因此在修改的代码中，我们先输入一个模板给系统，在填充数据段以？保留位置。而数据的输入在bind\_parm()函数中转化为二进制形式发送到数据库，不再视为代码，这样就能保证数据被封装。



在源码文件unsafe\_home.php中找到有关sql的定义



将该语句修改



保存后，再次登录界面，尝试SQL攻击，发现攻击失败，说明应对策略成功。

