**实验六实验报告**

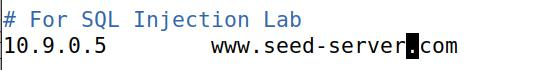
**基本信息：**

完成人姓名：顾琰 学号：57119117 完成日期：2021 年 7 月 30 日

**实验内容：**

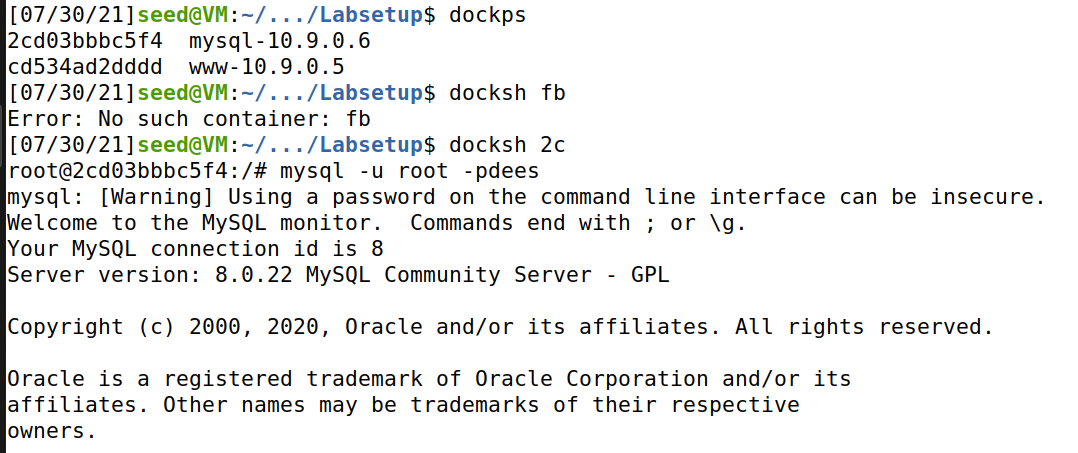
**环境配置：**

手动指定 DNS：

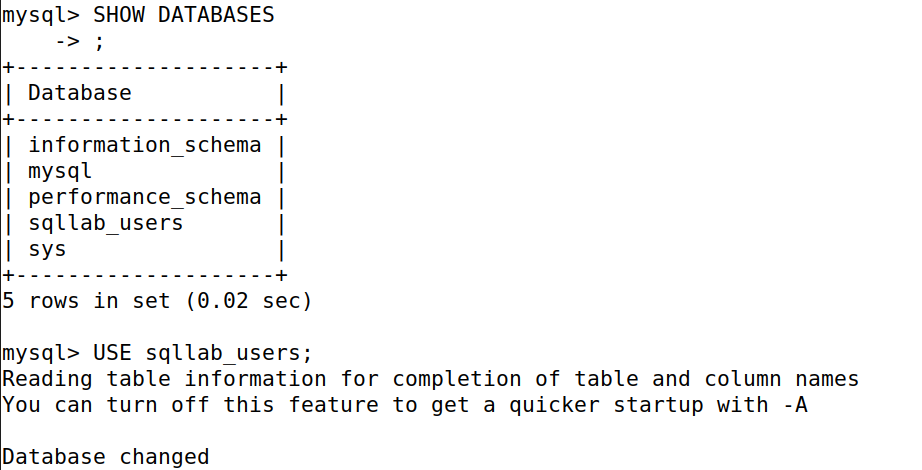


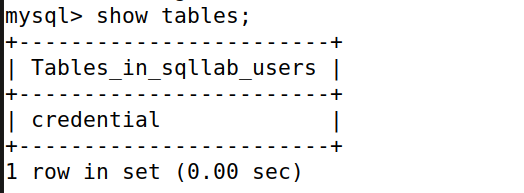
**Task 1: 熟悉 SQL 语句：**

进入数据库服务器并登录 MySQL：

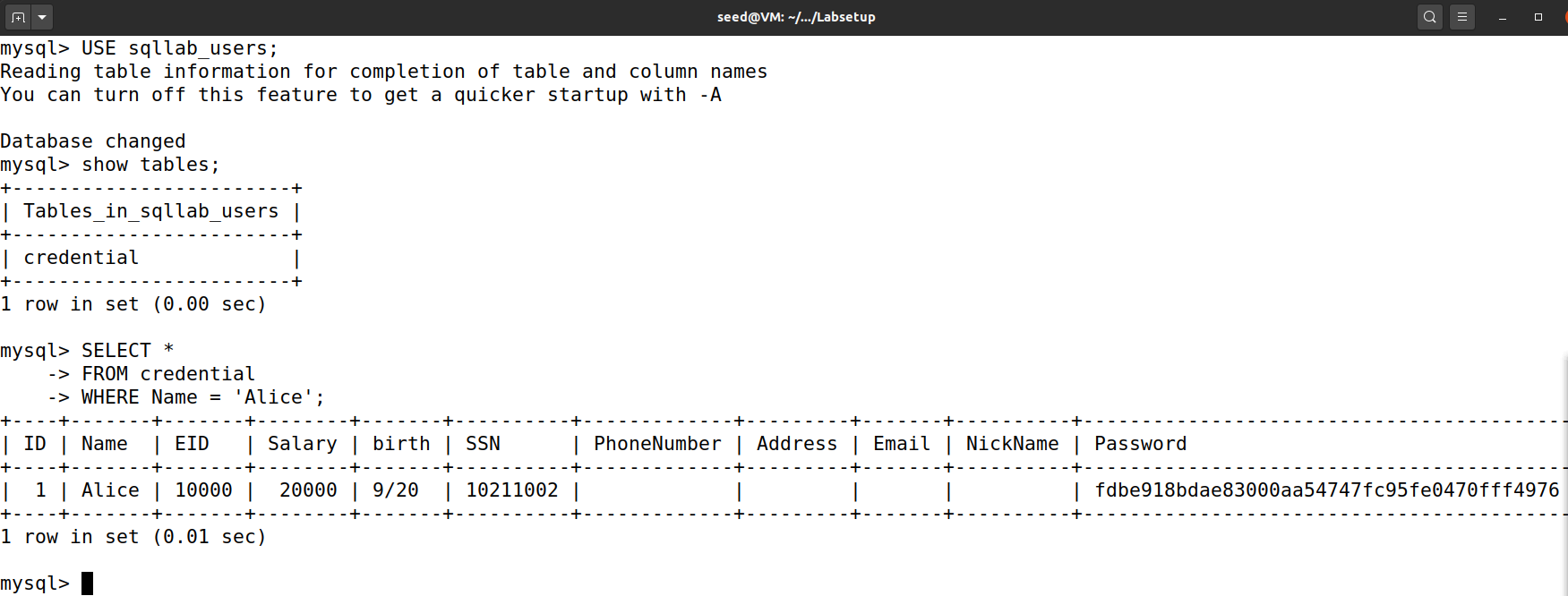


熟悉相关命令：



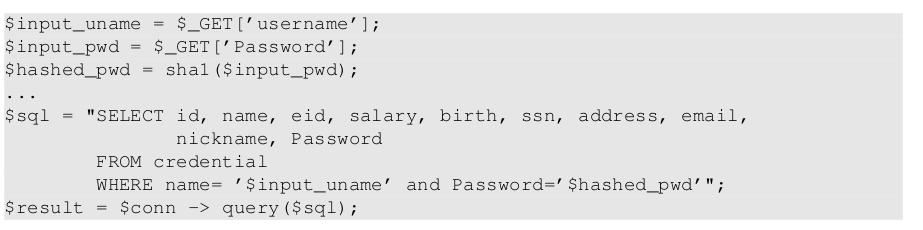


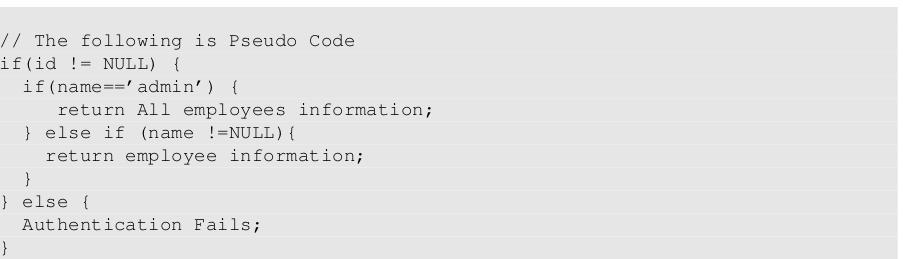
查询 Alice 所有信息：



**Task 2: SELECT 语句上的 SQL 注入：**

分析网页 php 源码寻找漏洞：





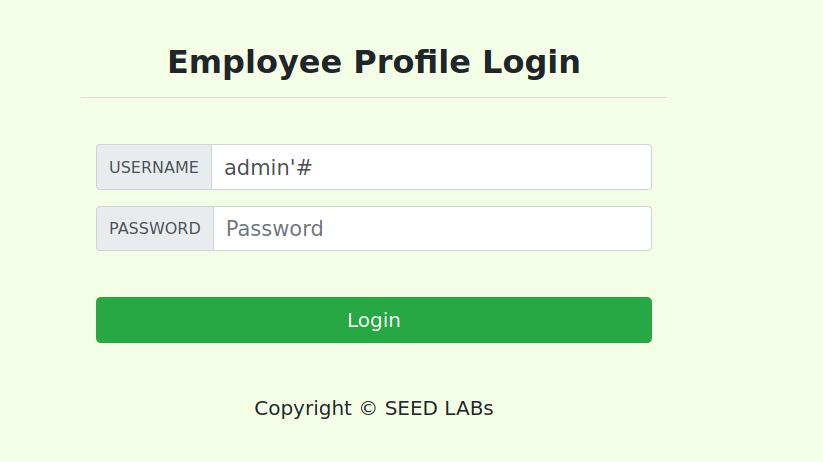
我们分析源码可以发现，网站开发者居然直接信任用户的输入，将用户的输入作为 SQL语句的一部分传入后端数据库，所以相当于我们可以在此处输入任何我们希望数据库执行的

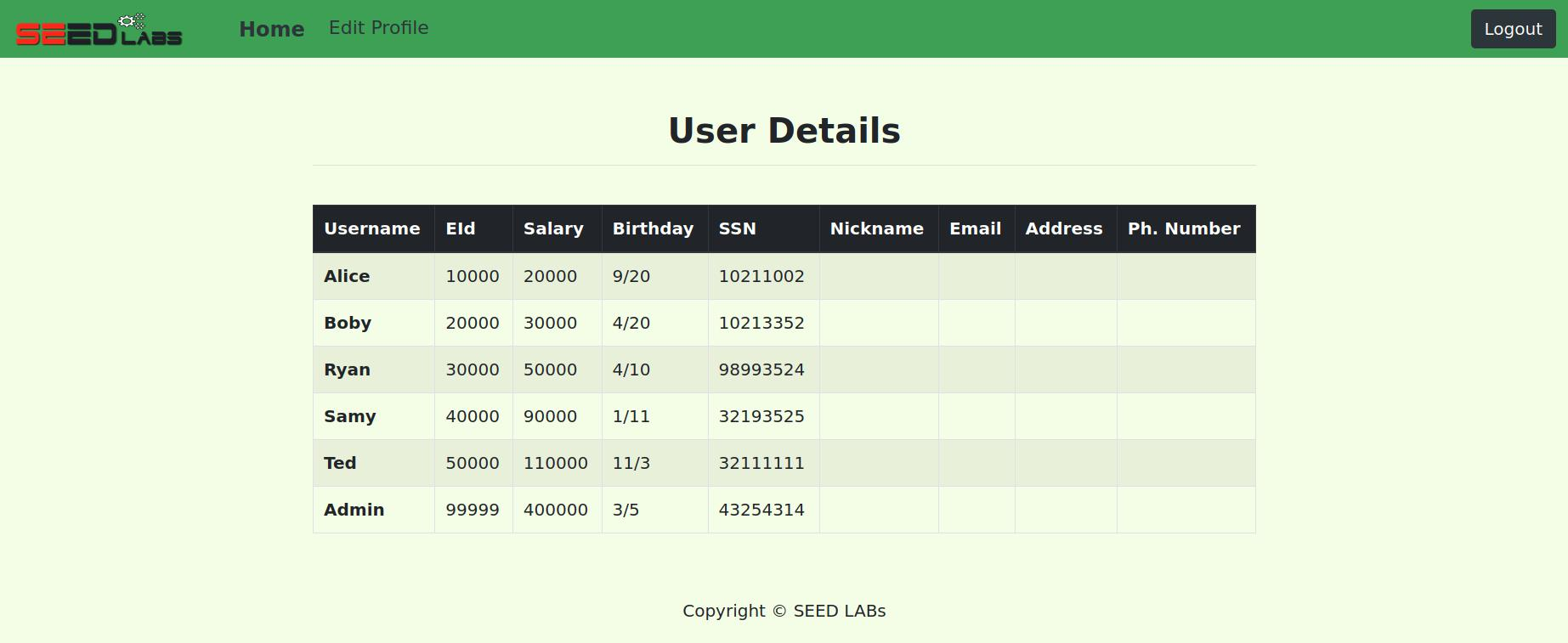
语句。

**①通过网页进行 SQL 注入攻击：**

现在我们假设我们知道管理员的账户是 admin，然后进行 SQL 注入攻击(注释掉 SQL 语

句中对密码的匹配)：

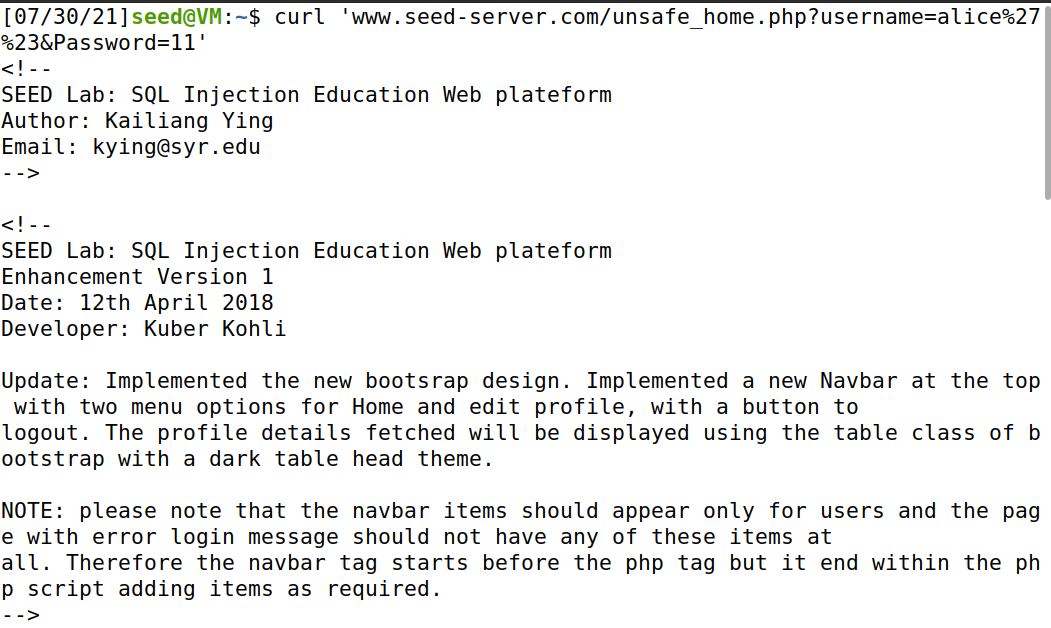




成功登录admin的账户。

**②通过命令行进行 SQL 注入攻击：**

注入内容和①相同，只是采用命令行方式，注意特殊符号使用编码输入。





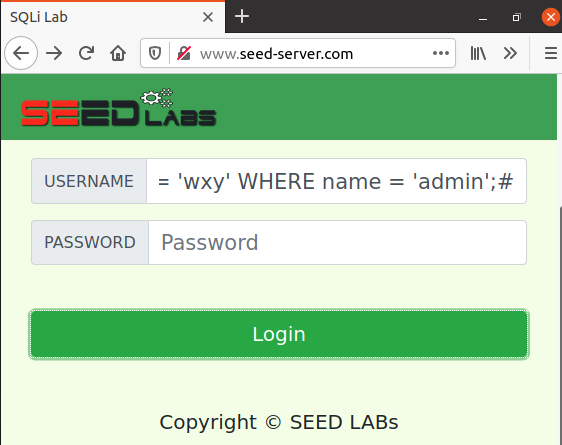
将页面源码与①中的对比，我们已经成功登入了 admin 的账户，返回了

admin 的账户页面。

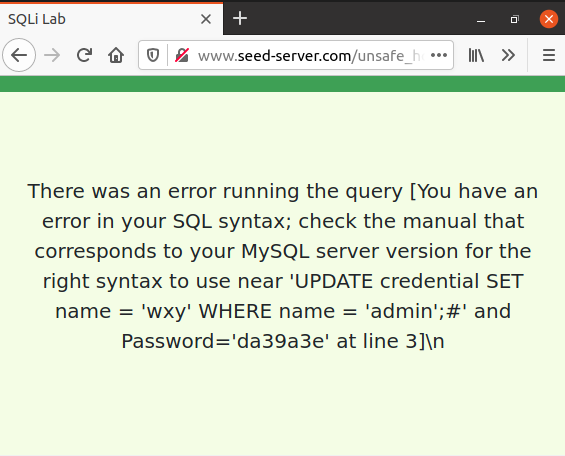
③追加新的 SQL 语句攻击：

尝试通过’；’注入第二条 SQL 语句。(输入 admin’;UPDATE credential SET name = ‘wxy’

WHERE name = ‘admin’;#)



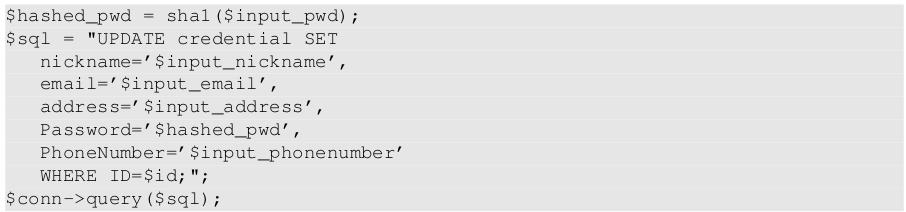
上述第二条命令尝试将数据库中的 admin 用户的 name 属性更改为wxy。



无法执行第二条命令。MySQL 中的 query 只允许执行一个命令，我们通过分号’;’隔开的第二个命令其不会被允许执行，这是一种预防 SQL 注入攻击的机制。

**Task 3: UPDATE 语句上的 SQL 注入：**

分析网页 php 源码寻找漏洞：



我们分析源码可以发现，网站开发者居然直接信任用户的输入，将用户的输入作为 SQL

语句的一部分传入后端数据库，所以相当于我们可以在此处输入任何我们希望数据库执行的

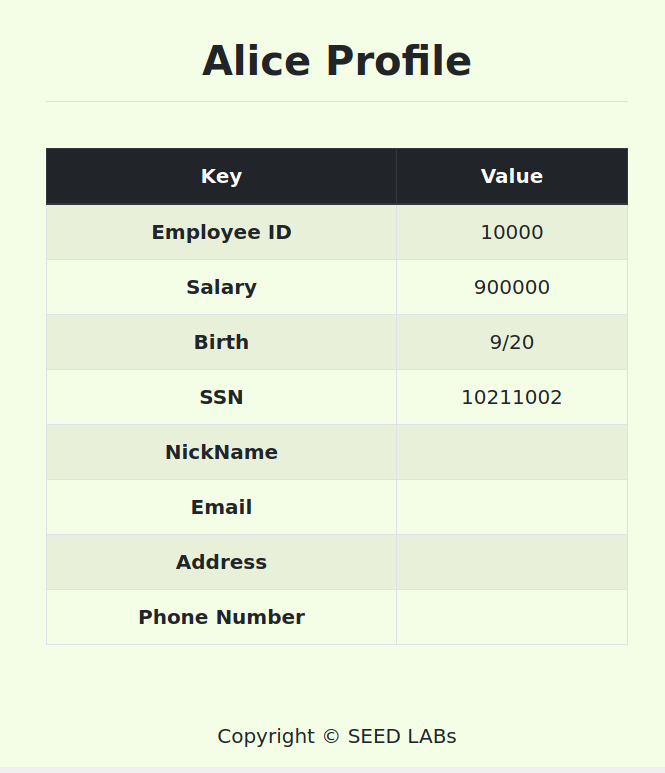
语句。

**①更改自己的薪水：**

假设我们是 Alice 并且我们已经知道薪水存储在数据库中的 salary 字段。



进行注入攻击：输入命令：’,salary=900000 WHERE name = ‘alice’;#

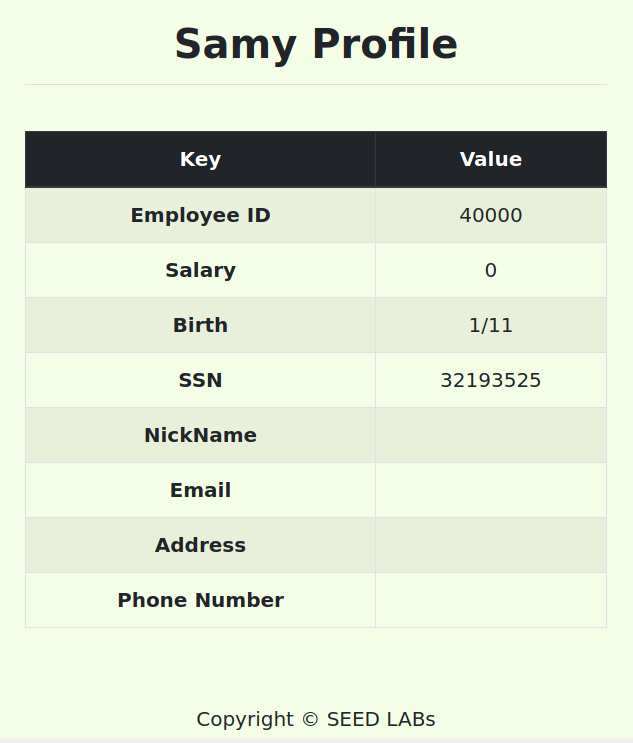


薪水成功修改为了 900000。

**②更改别人的薪水：**

输入命令：’,salary=0 WHERE name = ‘samy’;#

然后我们登录 samy 的账户查看工资：



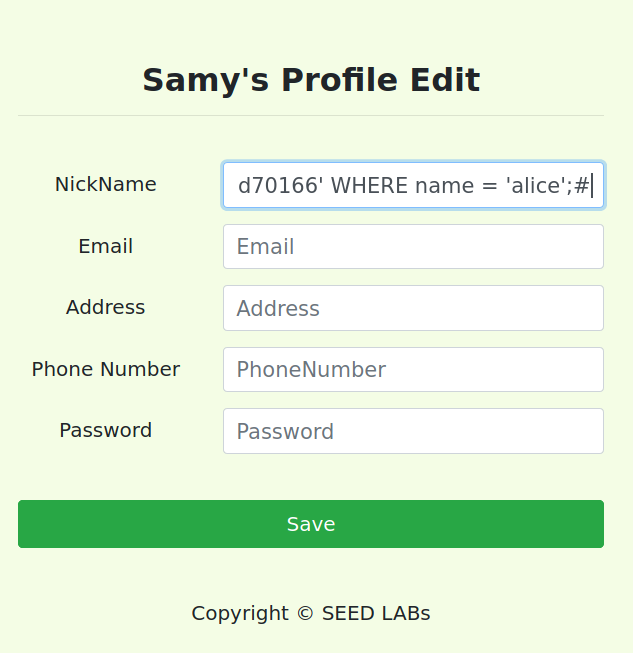
**③更改别人的密码：**

由于密码在数据库中的存储形式是 sha1 哈希值，所以我们先获取我们将要设置的

密码的哈希值。

IMG_256

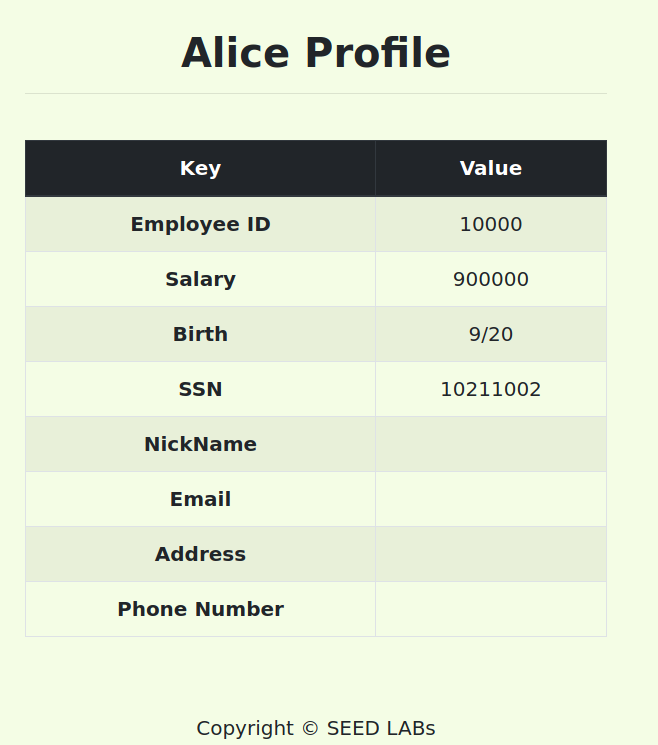
然后我们将上面的哈希值存储进数据库：



输入命令：’,Password=’1254df2027e06dc9eaa9f995213c1c9fedd70166’ WHERE

name = ‘alice’;#

输入alice的新密码：



登陆成功

**Task 4: 对策——准备好语句：**

修改 unsafe.php 文件。采用 Prepared Statement 方法，使得数据库先编译我们主

体的查询语句，然后将用户的输入作为数据传入，直接查询。由于用户输入的数据未经过编

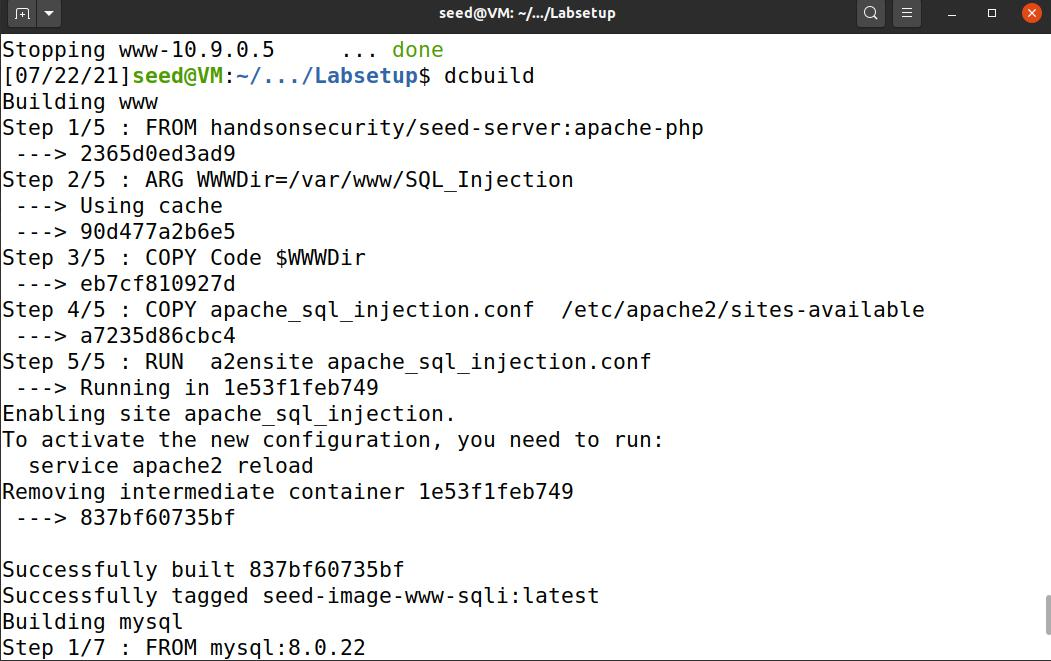
译阶段，所以用户即使注入了 SQL 语句也无法通过编译转化为数据库可以执行的语句。这

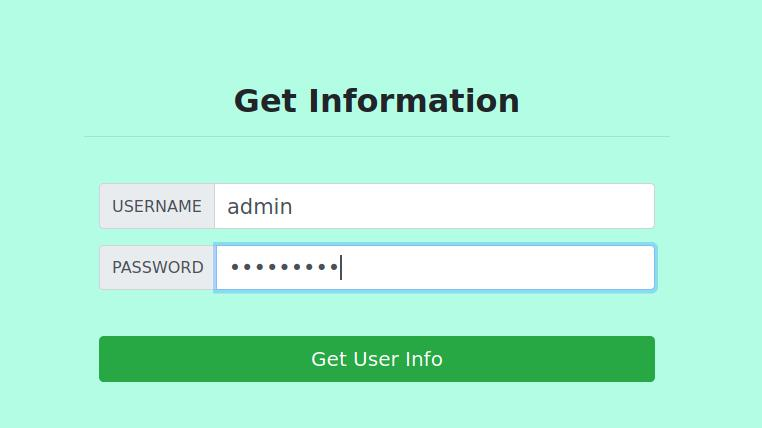
样便将我们开发人员写的执行代码和用户输入的数据代码清晰地划开了界限，以达到预防

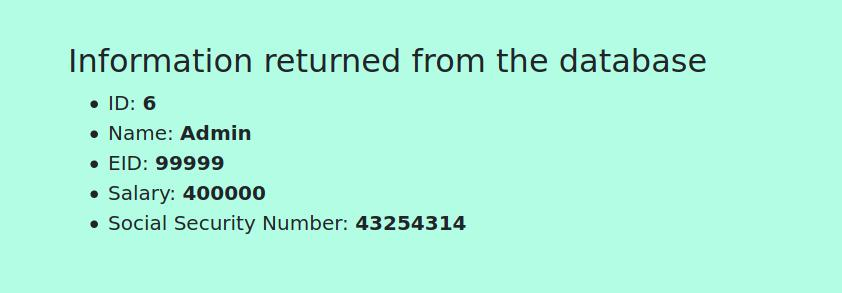
SQL 注入的效果。



再次编译运行：

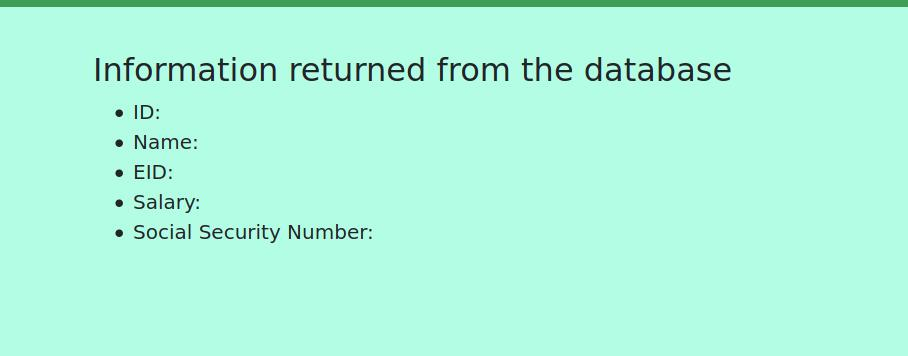






可见功能正常，接下来进行 SQL 注入攻击:





读取数据失败，说明攻击失效，防御策略成功。

**实验体会：**

本次实验是我们的第六次实验，经过本次实验，我总结了如下的知识点:

①网络应用程序一般将数据存储在数据库中。当它们需要从数据库访问数据时，需要构造SQL语句并将语句发送给数据库执行。通常，这些SQL语句包含不可信用户提供的数据。网络应用程序需要确保来自用户的数据不被当成代码，否则数据库可能会执行用户注入的指令。然而，很多网络应用程序对此风险并不重视，它们没有采取额外的措施来防范不可信代码被注入构造的SQL语句中，导致这些网络应用程序存在SQL注入漏洞。

②预防SQL注入攻击主要有两个方面：

1. 进行数据清洗，确保用户的输入中包含任何SQL代码。

（2）清楚地区分SQL代码与数据，当构造SQL语句时，分别发送数据和代码到数据库。通过这种方法，即便用户提供的数据中包含代码，这段代码也会被当成数据，不会对数据库造成破坏。预处理语句可以实现此目标。