HGAME2020 Final Writeup

原创

wh201906 于 2020-03-08 02:23:13 发布 426 收藏 1

分类专栏： CTF试水

本文链接：https://blog.csdn.net/wh201906/article/details/104726578

居然能苟进决赛

绝了

Misc - Good Video

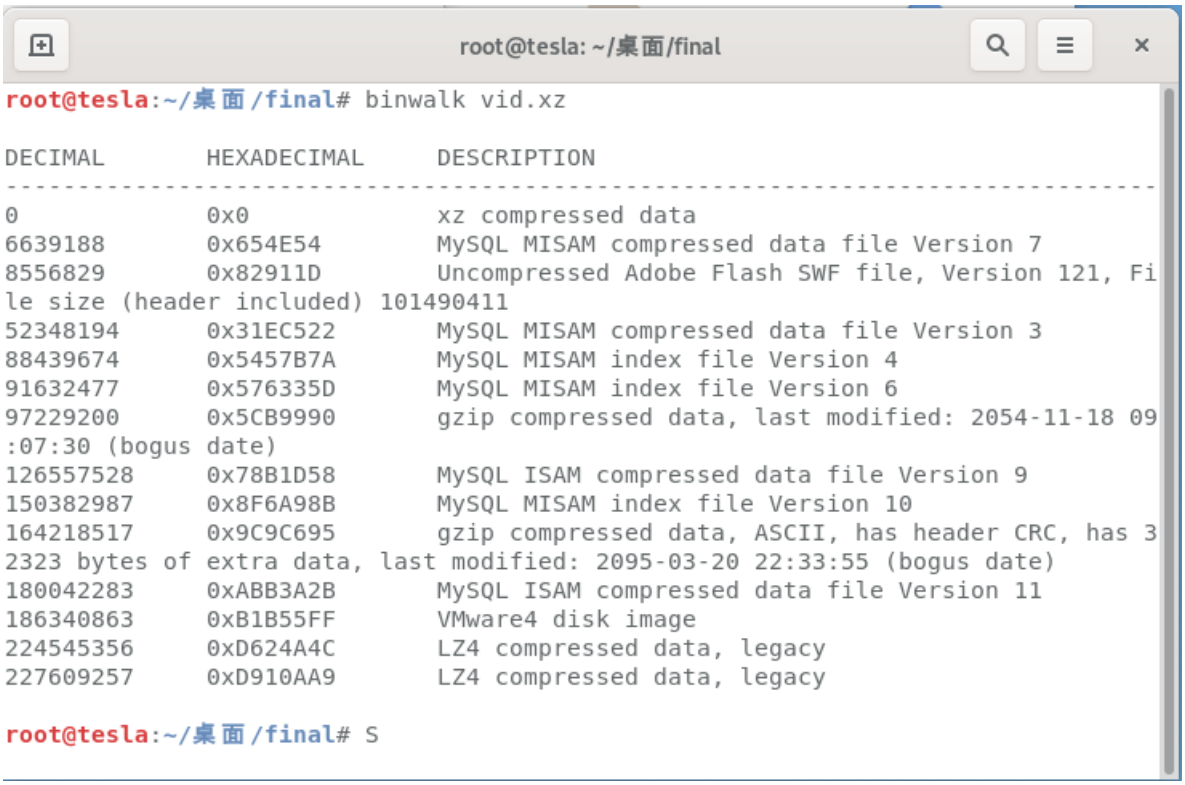
题目：

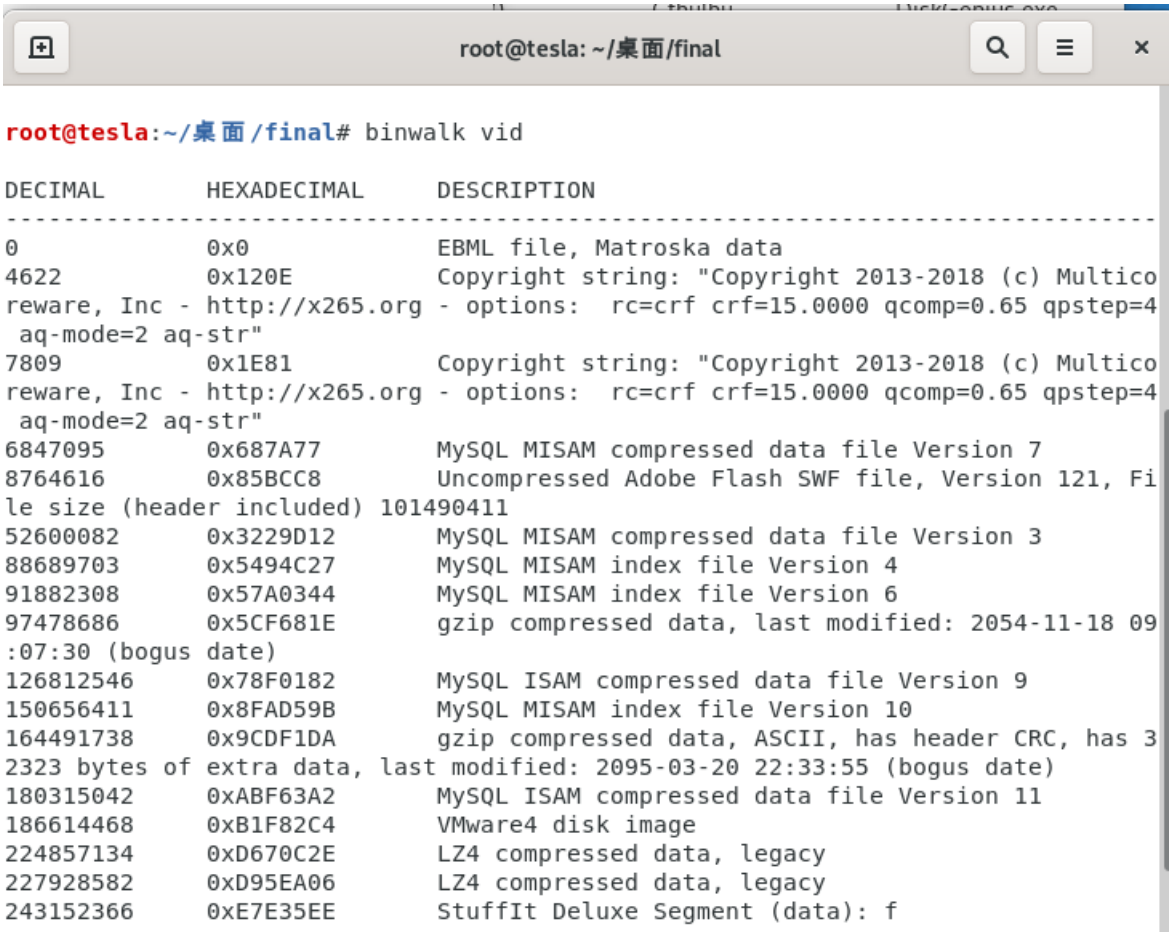
```
Need a video to fresh your mind?
```

终于又见到misc了

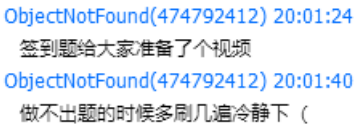
先binwalk扫了一遍压缩包和视频文件，得到了一大堆东西。。。。。

（居然还有vmdk？）

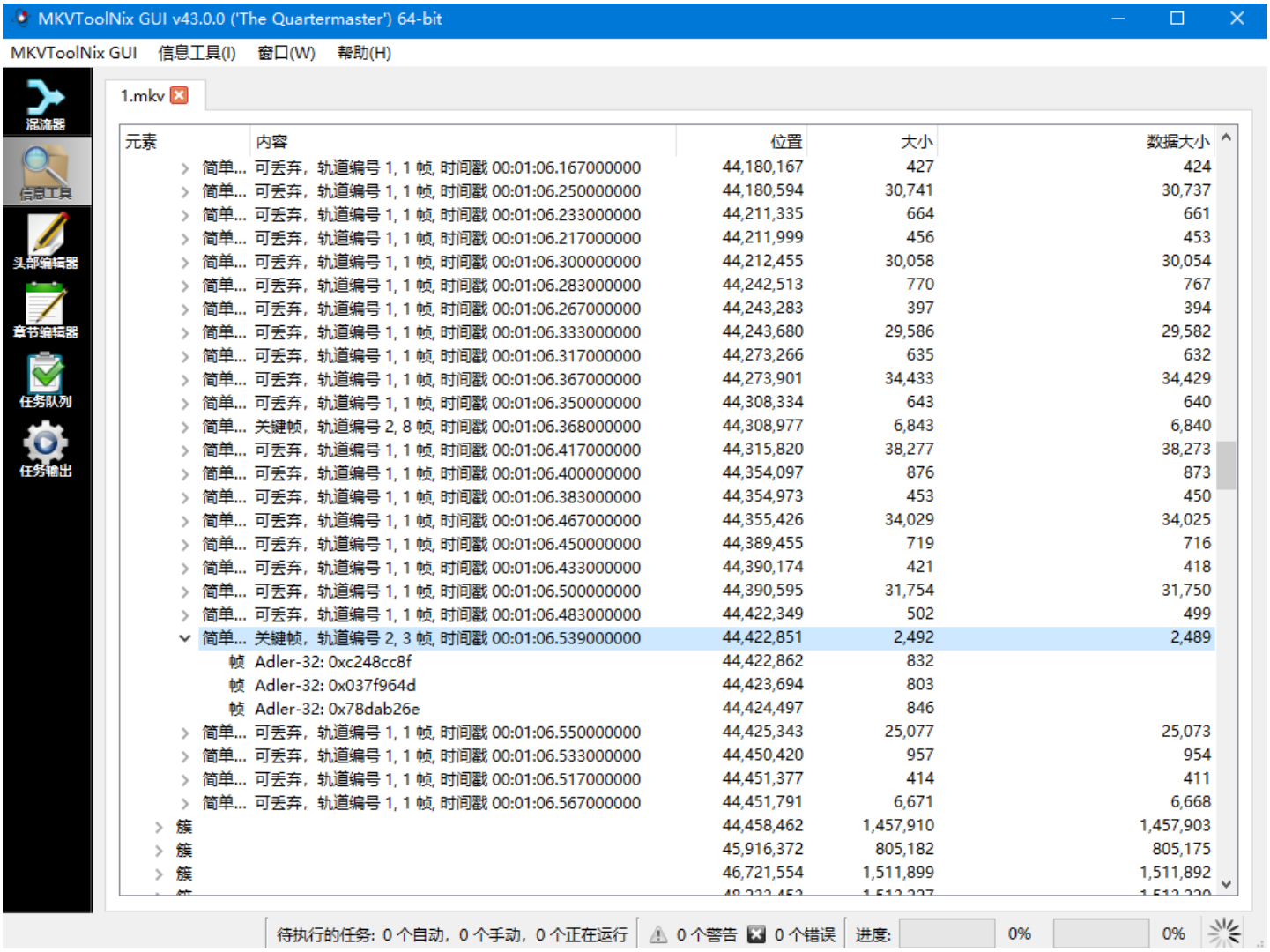




查了一下头部的EBML文件格式，发现这是一种面向未来的音视频框架，而且之后的Matroska也是一种封装格式
这些文件格式为了支持未来出现的新压缩格式因此灵活度很高，所以出题人可以在里面塞很多文件（然而并没有）
各种分离操作失败以后尝试直接对视频下手。Matroska格式最典型的例子就是.mkv文件。于是在网上下载MKVToolNix并尝试读取文件的音轨、字幕、备注等信息，无果
然后突然想起出题人叫我们多看看视频



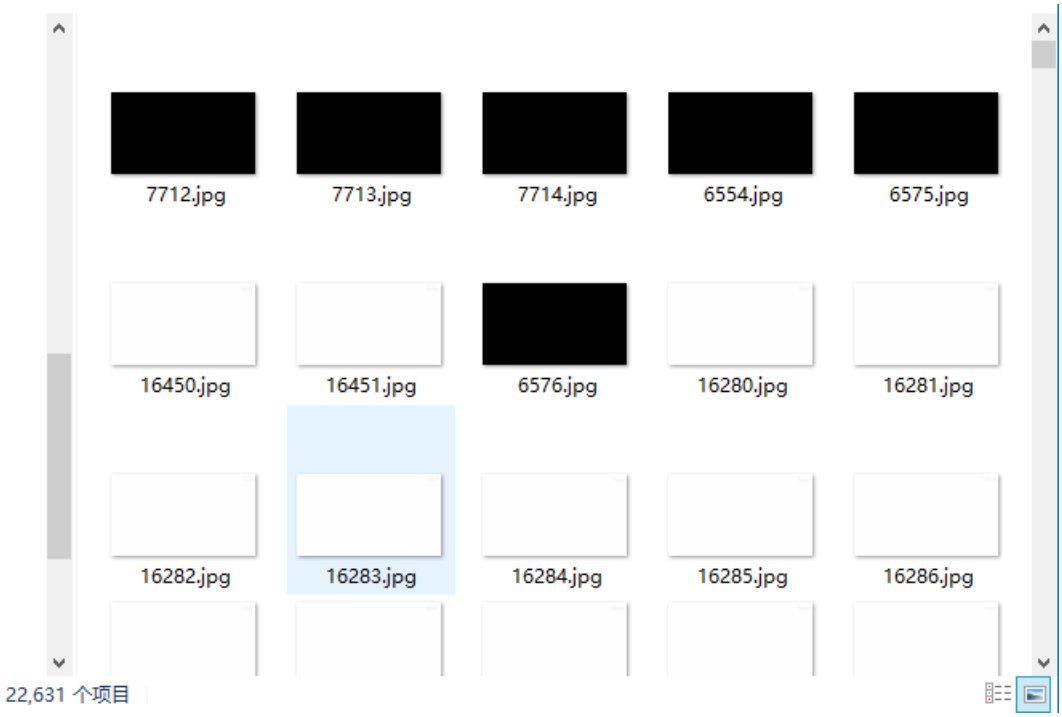
于是仔细欣赏了一遍小姐姐，发现视频里有若干张黑白色的图片闪过
用MKVToolNix看了一下帧的信息，因为肉眼能很明显看到黑白图片，所以优先查看关键帧，结果如下



考虑到手动找帧太麻烦，而且MKVToolNix不便于导出，于是使用ffmpeg来导出视频

```
.\ffmpeg\bin\ffmpeg -i 1.mkv .\out\%d.jpg
```

（需事先新建.\out文件夹）
然后得到了22000+张图片。。。。。



按照大小排序，先把过大的图片删掉，再筛选一下，然后找到了九张图片

拼合一下得到完整二维码，扫码即可获得flag

最终flag：hgame{gO0D_vld3O_EESvLoEPyC$zHlOJEHc0h&14}

Crypto - Response

题目：

```
Alice use mutual authentication protocol for her server, can you find her secret?
今日份的签到题
```

以及服务端脚本：

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
import os
import signal
import binascii
import socketserver

from Crypto.Cipher import AES
from Crypto.Util.Padding import pad, unpad

from Alice import KEY, MESSAGE

class Task(socketserver.BaseRequestHandler):

    def __init__(self, *args, **kargs):
        self.alice_prefix = b'Alice'
        self.server_prefix = b'Server'
        self.KEY = KEY
```

```
        super().__init__(*args, **kargs)

    def _recvall(self):
        BUFF_SIZE = 1024
        data = b''
        while True:
            part = self.request.recv(BUFF_SIZE)
            data += part
            if len(part) < BUFF_SIZE:
                break
        return data.strip()

    def send(self, msg, newline=True):
        try:
            if newline:
                msg += b'\n'
            self.request.sendall(msg)
        except:
            pass

    def recv(self, prompt=b'> '):
        self.send(prompt, newline=False)
        return self._recvall()

    def encrypt(self, data):
        iv, pt = data[:AES.block_size], data[AES.block_size:]
        pt = pad(pt, AES.block_size)
        aes = AES.new(self.KEY, AES.MODE_CBC, iv=iv)
        ct = aes.encrypt(pt)
        return iv + ct

    def decrypt(self, data, unpad_pt=False):
        iv, ct = data[:AES.block_size], data[AES.block_size:]
        aes = AES.new(self.KEY, AES.MODE_CBC, iv=iv)
        pt = aes.decrypt(ct)
        if unpad_pt:
            pt = unpad(pt, AES.block_size)
        return pt

    def timeout_handler(self, signum, frame):
        self.send(b"\n\nSorry, time out.\n")
        raise TimeoutError

    def handle(self):
        signal.signal(signal.SIGALRM, self.timeout_handler)
        signal.alarm(60)

        try:
            iv = os.urandom(AES.block_size)

            for _ in range(3):
                name = self.recv(prompt=b'\nWho are you? ')
                if name == b'Alice':
                    # authenticate the server
                    hex_challenge = self.recv(
                        prompt=b'Give me your challenge (in hex): '
                    )
                    challenge = binascii.unhexlify(hex_challenge)
                    response = self.encrypt(
                        iv
```

```
                        iv
                        + self.server_prefix
                        + challenge
                    )
                    hex_response = binascii.hexlify(response)
                    self.send(b'The Response (in hex): ' + hex_response)

                    # authenticate Alice
                    challenge = os.urandom(AES.block_size)
                    hex_challenge = binascii.hexlify(challenge)
                    self.send(b'The Challenge (in hex): ' + hex_challenge)
                    hex_response = self.recv(
                        prompt=b'Give me your response (in hex): '
                    )
                    response = binascii.unhexlify(hex_response)
                    data = self.decrypt(response)
                    if (data.startswith(self.alice_prefix)
                            and challenge in data):
                        self.send(b'\nWelcome, Alice.')
                        self.send(b'Here is a message for you: ')
                        self.send(b'\t' + MESSAGE)
                    else:
                        self.send(b'Go away hacker!')
                else:
                    self.send(b"You shouldn't be here.")
                    break

        except:
            pass

        self.request.close()

class ForkedServer(socketserver.ForkingMixIn, socketserver.TCPServer):
    pass

if __name__ == "__main__":
    HOST, PORT = '0.0.0.0', 1234
    server = ForkedServer((HOST, PORT), Task)
    server.allow_reuse_address = True
    server.serve_forever()
```

如同题目所说，确实是签到题，整个题就裸的CBC字节翻转攻击。但由于自己对这一方式不熟，再加上足够沙雕，浪费了很多时间

一开始根据题目提示去搜索Challenge/Response认证，没有找到明显的攻击方法。再加上之前玩PM3的时候了解到这种认证的应用范围很广，应该比较安全，所以准备从CBC下手
做Week4的CBCBC时了解到针对CBC的攻击方式一般是字节翻转攻击（byte flipping attack）和填充提示攻击（padding oracle attack），上次考了后者，那么这次应该就是字节翻转攻击了。

字节翻转攻击的教程网上很多，其原理是通过修改当前密文块来将下一个块的明文改为我们想要的内容。分析服务端的源码可知，当我们提交的response可以被解密成"Alice"+含challenge的文本时即可得到MESSAGE（因为这里response的内容不是唯一指定的，所以不会涉及到多次加密构造密文，只需要一轮双方的challenge/response即可得到结果）（感谢出题人）

其具体原理是，根据CBC模式加解密的流程图，解密过程中总有P[i] = C[i-1] ^ Mid[i]
（Mid[i]在本题中是通过C[i]和KEY通过AES算法解密得到的）
因此可以得到P[i] ^ C[i-1] = Mid[i]
而如果令C[i-1]=P[i] ^ C[i-1] ^ T[i]（T[i]为我们希望服务器解密后得到的明文块）
则有P'[i] = P[i] ^ C[i-1] ^ T[i] ^ Mid[i] = Mid[i] ^ T[i] ^ Mid[i] = T[i]，也即达到了修改服务器解密结果的目的
（当i=1时，第一个块P[1]对应的C[1-1]为IV，IV是加/解密时使用的初始化向量）
观察本题中的decrypt函数，其使用的IV来自response（由Alice提供）而不是服务器在加密时生成的IV，这也为构造第一个块的"Alice"提供了可能（而且解密时不考虑填充，进一步降低难度）
因此对于一轮双方的challenge/response，修改第i个块就需要提交构造过的i-1块密文和正确的i块密文，为了符合题目要求，需要至少提交4个块的内容作为response（构造的IV+原始第1块密文+构造密文+原始第3块密文），使得服务器解密获得"Alice…"块+废弃块+challenge块以通过验证
而由于一开始先验证服务器的身份，因此能够提前获得本次连接中正确的IV以及用作素材的明文/密文对

具体攻击思路如下：
定义一个对bytes的异或函数

```
XOR = lambda s1, s2: bytes([x ^ y for x, y in zip(s1, s2)])
```

先提交b'\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00'（10个\x00）作为验证服务器的challenge
服务器对P[1]:b'Server\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00'（刚好是一个块的大小）进行加密，返回IV+C[1]+C[padding]（最后一个块用处不大）
之后服务器提供response4alice
此时令attack_IV=IV ^ P[1] ^ T[1] (T[1] = b'Alice\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00')
令attack_ciphtxt=C[1] ^ P[1] ^ response4alice
提交attack_IV + C[1] + attack_ciphtxt + C[1]
即可通过验证

攻击脚本如下（当中变量命名和wp中稍有不同）：

```
from socket import socket
from telnetlib import Telnet
from time import sleep
from binascii import hexlify, unhexlify
import re

XOR = lambda s1, s2: bytes([x ^ y for x, y in zip(s1, s2)])

mychalnge = b'Server\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00'
sendtext = b'\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00'

sock = socket()
sock.connect(("xxx.xxx.xxx.xxx", xxxxx))
tel = Telnet()
tel.sock = sock
sleep(0.1)

text = tel.read_until(b'? ').decode()
print("Server:" + text)

tel.write(b"Alice")
sleep(0.1)

text = tel.read_until(b': ').decode()
print("Server:" + text)

tel.write(hexlify(sendtext))
sleep(0.1)

text = tel.read_until(b'The Challenge (in hex): ').decode()
print("Server:" + text)

c = unhexlify(re.search(r'(?<=(: )).+(?=\n)', text).group(0))
# print('###',c)
iv = c[:16]
cc = c[16:32]
target = b'Alice\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00'
attack_iv = XOR(iv, mychalnge)
print("!!!",attack_iv)
attack_iv = XOR(attack_iv, target)

text = tel.read_until(b'se (in hex): ').decode()
print("Server:" + text)

# print("###", re.search(r'.+(?=\n)', text).group(0))
srvchalnge = unhexlify(re.search(r'.+(?=\n)', text).group(0))
attack_cipher = XOR(iv, mychalnge)
attack_cipher = XOR(attack_cipher, srvchalnge)

tel.write(hexlify(attack_iv + cc + attack_cipher + cc))
tel.interact()
```

最终服务器返回信息如下：

```
Welcome, Alice.
Here is a message for you:
        The flag is hgame{RefL3cti0n_@tt4ck_wiTh_c8C~B1t-fLiPpiNg}, keep it secret.
```

最终flag：hgame{RefL3cti0n_@tt4ck_wiTh_c8C~B1t-fLiPpiNg}

沙雕操作：
没有注意到服务器计算response时会先加上一个“Server”，直接将16个b’\x00’作为P[1]
构造response时写成了attack_IV + P[1] + attack_ciphtxt + P[1]，为此试了2个小时
…

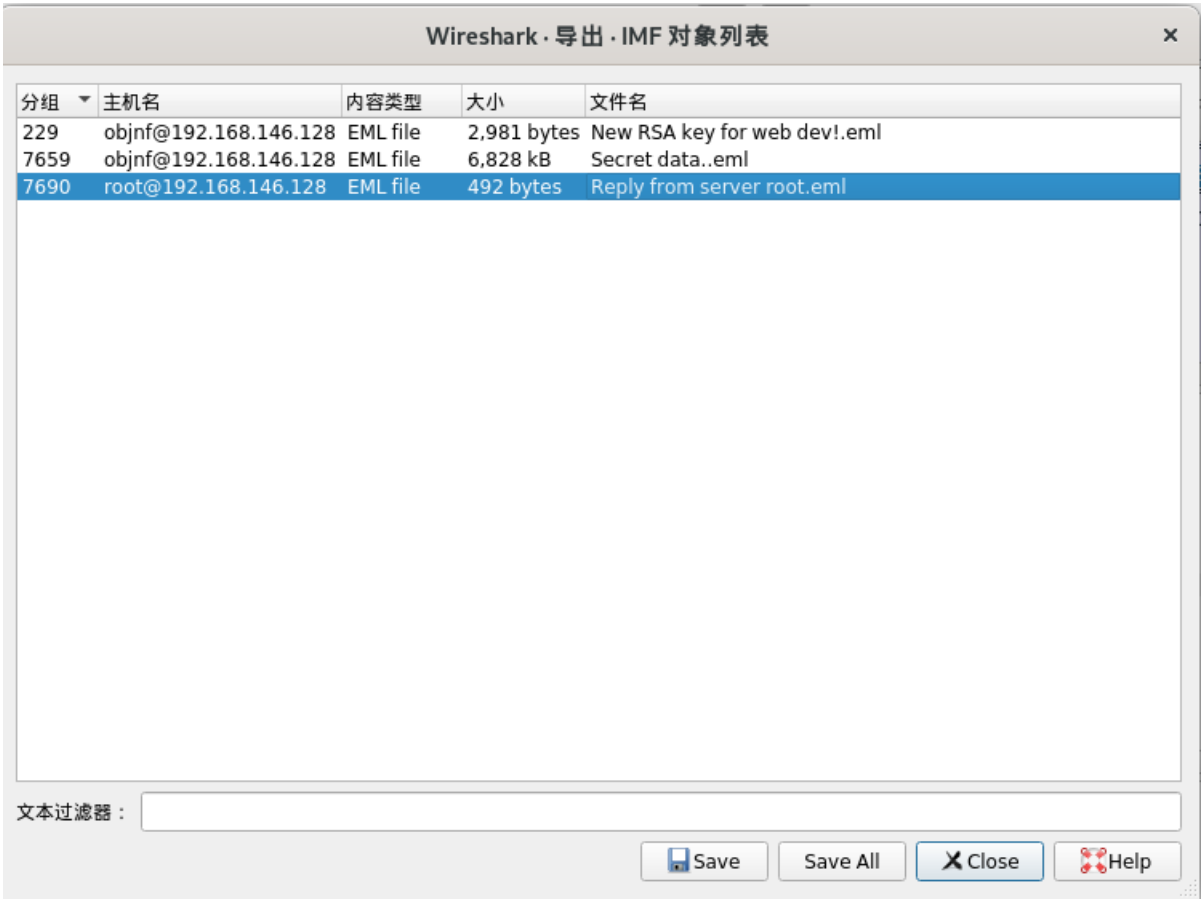
Misc - Good Video（未做完）
只差5分钟…

题目：

```
Nothing but a pcapng.
【修复补给】http://oss-east.zhouweitong.site/hgamefinal/flag.png.zip，懂的人自然懂
```

一开始没有修复补给，只有一个含pcapng的压缩包

丢到wireshark里面，发现主要是SMTP的包，其中还有一些TELNET的包
一番乱点之后在文件→导出对象→IMF…中找到了几个奇怪的包



经过分析，第一个包内含有一个RSA私钥文件，第二个包内是一个压缩文件。两者都是通过base64来传输的
导出base64文本文件后通过一个简单的脚本来恢复文件

```
import base64
strs=open("C:\\hgame\\final\\out.txt", "r").read()
res = base64.b64decode(strs)
f = open("C:\\hgame\\final\\1.zip", "wb+")
f.write(res)
```

之后打开压缩文件，提示压缩文件被损坏。经过WinRAR修复之后得到了一个加密的压缩包。尝试通过处理伪加密的方法来处理压缩包，结果解压文件时校验错误，说明压缩包确实有密码

打开RSA私钥文件，里面的内容没有异常
在wireshark当中继续搜寻邮件相关的包，发现了一些登录邮箱时的认证信息

```
192.168.146.1       SMTP      120 S: 250-192.168.146.128 | SIZE 20480000 | AUTH LOGIN | HELP
192.168.146.128     SMTP       66 C: AUTH LOGIN
192.168.146.1       SMTP       72 S: 334 VXNlcm5hbWU6
192.168.146.128     SMTP       84 C: User: b2JqbmZAMTkyLjE2OC4xNDYuMTI4
192.168.146.1       SMTP       72 S: 334 UGFzc3dvcmQ6
192.168.146.128     SMTP       64 C: Pass: MTIzNDU2
192.168.146.1       SMTP       74 S: 235 authenticated.
192.168.146.128     SMTP       99 C: MAIL FROM:<objnf@192.168.146.128> SIZE=2981
192.168.146.1       SMTP       62 S: 250 OK
192.168.146.128     SMTP       86 C: RCPT TO:<root@192.168.146.128>
```

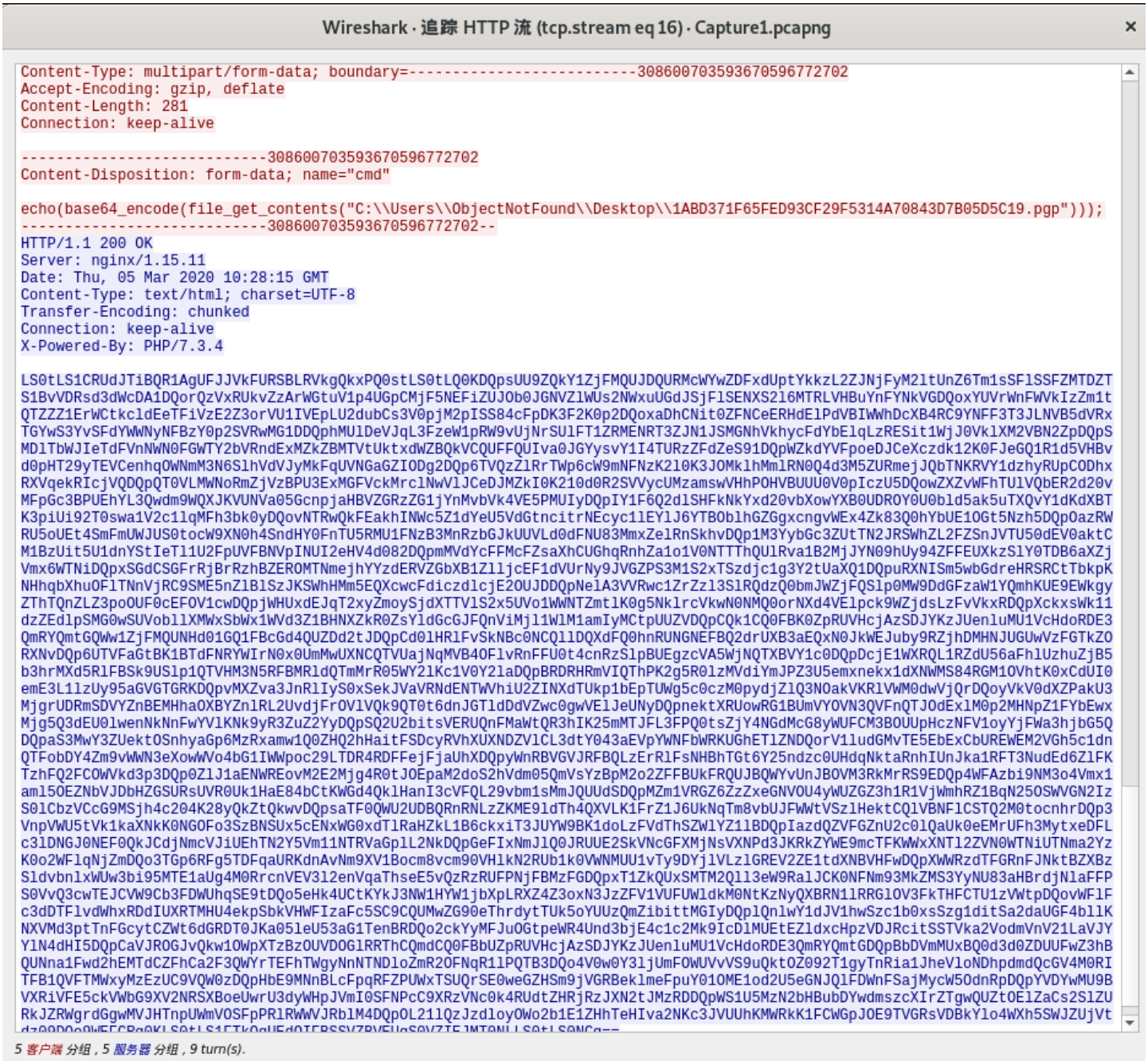
将这些文本用于解压，均提示密码错误
之后搜索http包，没有结果
后来在浏览包的时候偶然发现有TLS包

```
ICMPv6      86 Neighbor Solicitation for fe80::250:56ff:fec0:2222 from 00:50:56:c0:00:08
TCP         66 3074 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM=1
TCP         66 443 → 3074 [SYN, ACK] Seq=0 Ack=1 Win=8192 Len=0 MSS=1460 WS=256 SACK_PERM=1
TCP         60 3074 → 443 [ACK] Seq=1 Ack=1 Win=1051136 Len=0
TLSv1.2    290 Client Hello
TLSv1.2   1093 Server Hello, Certificate, Server Hello Done
TLSv1.2    372 Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message
TLSv1.2    312 New Session Ticket, Change Cipher Spec, Encrypted Handshake Message
TLSv1.2    663 Application Data, Application Data, Application Data
TLSv1.2   1658 Application Data
TCP         60 3074 → 443 [ACK] Seq=1164 Ack=2902 Win=1051136 Len=0
ICMPv6      86 Neighbor Solicitation for fe80::250:56ff:fec0:2222 from 00:50:56:c0:00:08
ICMPv6      86 Neighbor Solicitation for fe80::250:56ff:fec0:2222 from 00:50:56:c0:00:08
ARP         42 Who has 192.168.146.2? Tell 192.168.146.128
ARP         60 192.168.146.2 is at 00:50:56:ec:17:a9
ICMPv6     102 Router Advertisement
TLSv1.2    679 Application Data, Application Data, Application Data
```

之前做含TLS包题目的时候是通过浏览器的日志文件导入密钥，从而解密包，然而此题中并没有这样的日志文件（或者说暂时没找到）
而刚才已经获得了一个RSA文件，结合相关邮件内容中提到“web dev”，该文件也许可以用来解密TLS包
依次进入编辑→首选项→协议(Protocols)→TLS(或SSL)→RSA keys list，新建一个条目，在Key File一栏当中加入刚才的RSA私钥文件并保存，之后wireshark中便出现了http包

```
TLSv1.2    290 Client Hello
TLSv1.2   1093 Server Hello, Certificate, Server Hello Done
TLSv1.2    372 Client Key Exchange, Change Cipher Spec, Finished
TLSv1.2    312 New Session Ticket, Change Cipher Spec, Finished
HTTP       663 POST /shell.php HTTP/1.1
HTTP      1658 HTTP/1.1 200 OK  (text/html)
TCP         60 3074 → 443 [ACK] Seq=1164 Ack=2902 Win=1051136 Len=0
ICMPv6      86 Neighbor Solicitation for fe80::250:56ff:fec0:2222 from 00:50:56:c0:00:08
ICMPv6      86 Neighbor Solicitation for fe80::250:56ff:fec0:2222 from 00:50:56:c0:00:08
ARP         42 Who has 192.168.146.2? Tell 192.168.146.128
ARP         60 192.168.146.2 is at 00:50:56:ec:17:a9
ICMPv6     102 Router Advertisement
HTTP       679 POST /shell.php HTTP/1.1
```

追踪HTTP流，最后得到了一个pgp文件（仍然需要先base64解码）

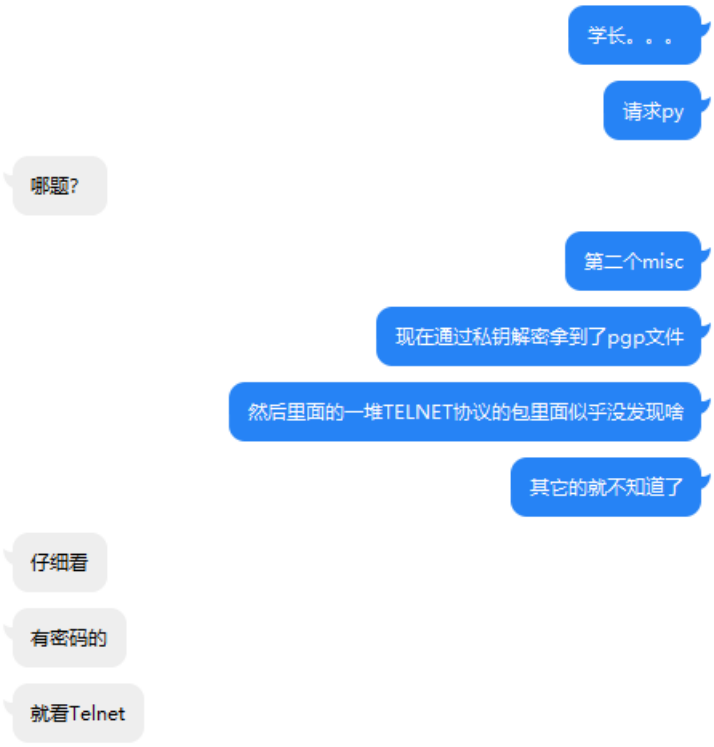


之前听说过pgp可用于消息加密和数字签名，结合加密压缩文件中的“flag.png.gpg”，这个pgp文件应该是用来解密flag.png的（pgp(pretty good privacy)是一套用于加密或验证的系统(或者规范？)，而gpg(GnuPG, Gnu Privacy Guard)是基于OpenPGP标准开发的自由软件）

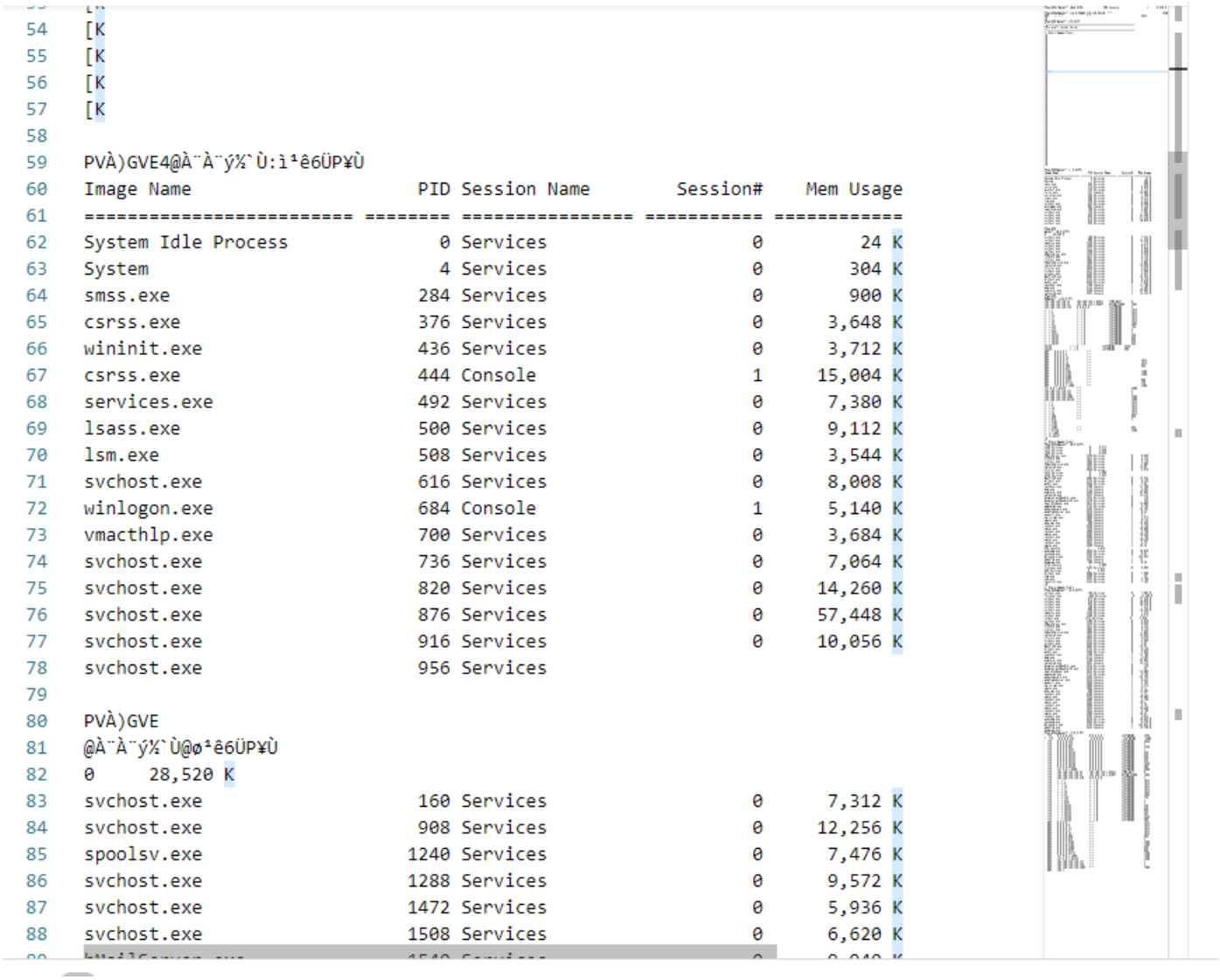
然而压缩文件密码的问题还是没有解决掉

考虑到pcapng文件中有好几个telnet包，于是尝试从当中提取信息。当中有若干个包仅含单个字符，拼接后大概是netstat -ano,exit,tasklist之类的cmd命令（此时是按大小排序，挖坑）。而之后较大的telnet包中则含有这些命令的运行结果，但是有很多乱码

之后实在找不到思路了，于是py了一下出题人



把几个较大的包中的内容整理了一下，得到以下结果：



```
54  [K
55  [K
56  [K
57  [K
58
59  PVÀ)GVE4@À¨À¨ý½`Ù:ì¹ê6ÜP¥Ù
60  Image Name                     PID Session Name        Session#    Mem Usage
61  ========================= ======== ================ =========== ============
62  System Idle Process              0 Services                   0         24 K
63  System                           4 Services                   0        304 K
64  smss.exe                       284 Services                   0        900 K
65  csrss.exe                      376 Services                   0      3,648 K
66  wininit.exe                    436 Services                   0      3,712 K
67  csrss.exe                      444 Console                    1     15,004 K
68  services.exe                   492 Services                   0      7,380 K
69  lsass.exe                      500 Services                   0      9,112 K
70  lsm.exe                        508 Services                   0      3,544 K
71  svchost.exe                    616 Services                   0      8,008 K
72  winlogon.exe                   684 Console                    1      5,140 K
73  vmacthlp.exe                   700 Services                   0      3,684 K
74  svchost.exe                    736 Services                   0      7,064 K
75  svchost.exe                    820 Services                   0     14,260 K
76  svchost.exe                    876 Services                   0     57,448 K
77  svchost.exe                    916 Services                   0     10,056 K
78  svchost.exe                    956 Services
79
80  PVÀ)GVE
81  @À¨À¨ý½`Ù@ø¹ê6ÜP¥Ù
82  0     28,520 K
83  svchost.exe                    160 Services                   0      7,312 K
84  svchost.exe                    908 Services                   0     12,256 K
85  spoolsv.exe                   1240 Services                   0      7,476 K
86  svchost.exe                   1288 Services                   0      9,572 K
87  svchost.exe                   1472 Services                   0      5,936 K
88  svchost.exe                   1508 Services                   0      6,620 K
```

然而并没有什么发现

之后百度"wireshark telnet"，得知telnet的登陆过程中其密码也是以单个字符的方式发送出去的。经过一番寻找和拼接，找到了登录时提交的"hgame_final"和"WelcomeToHgameFinal"两个字符串

（因为按大小排序后"login"和"password"两个包出现在后面，难以找到其对应的输入信息）

经测试，后者可以解压提取出的压缩文件，但是解压过程中却提示压缩包已损坏。此时使用【修复补给】当中提供的压缩包进行解压，得到了flag.png.gpg

之后命令行下用HTTP流中的pgp文件解密出flag.png

```
gpg --import 1.pgp
gpg -o out.png -d flag.png.gpg
gpg --list-keys
gpg --delete-secret-keys 1ABD371F65FED93CF29F5314A70843D7B05D5C19
gpg --delete-keys 1ABD371F65FED93CF29F5314A70843D7B05D5C19
```

先在kali中直接打开文件，发现CRC error，binwalk识别到zlib数据，但是改名为zip文件后仍然无法打开文件（其实zlib和zip不是一个东西）。后来把文件转移到Windows下打开，显示正常，用StegSolve把大部分功能都用了一遍，无果

（然后比赛就结束了）

冷静下来后回忆起之前看到过png文件的实际尺寸可能和属性中的尺寸不一样，于是在网上搜索处理方法。用16进制编辑器将文件头中的长度数据度变大之后即可看到flag

hgame{Re4LlY_gO0D_P4ck3ts_92252043}

最终flag：hgame{Re4LlY_gO0D_P4ck3ts_92252043}（没能提交到平台上，不知道有没有打错字）

提前装了sage和z3，看了一大堆线性反馈移位寄存器，对着BM算法想了很久，结果对第二道Crypto还是束手无策
毕竟是final

2020.03.07