

# DDCTF - 2018 解题过程

Author: pandaos

需要用脚本解题的都附有完整代码。

<https://github.com/Chenyuxin/writeup>

## 签到题

略

## misc2

```
data=bytearray.fromhex('d4e8elf4a0f7e1f3a0e6elf3f4a1a0d4e8e5a0e6ece1e7a0e9f3baa0c4c4c3d4c6fbb9e1e6b3e3b9e4b3b7b7e2b6b1e4b2b6b9e2b1b1b3b3b7e6b3b3b0e3b9b3b5e6fd')
flag = ''
for i in data:
    flag = flag + chr(i & 0x7f)
print flag
```

## misc3

题目给出了一张图片，文件特别大，不用想里面肯定加了附加数据，于是用binwalk扫描，扫描结果如下：

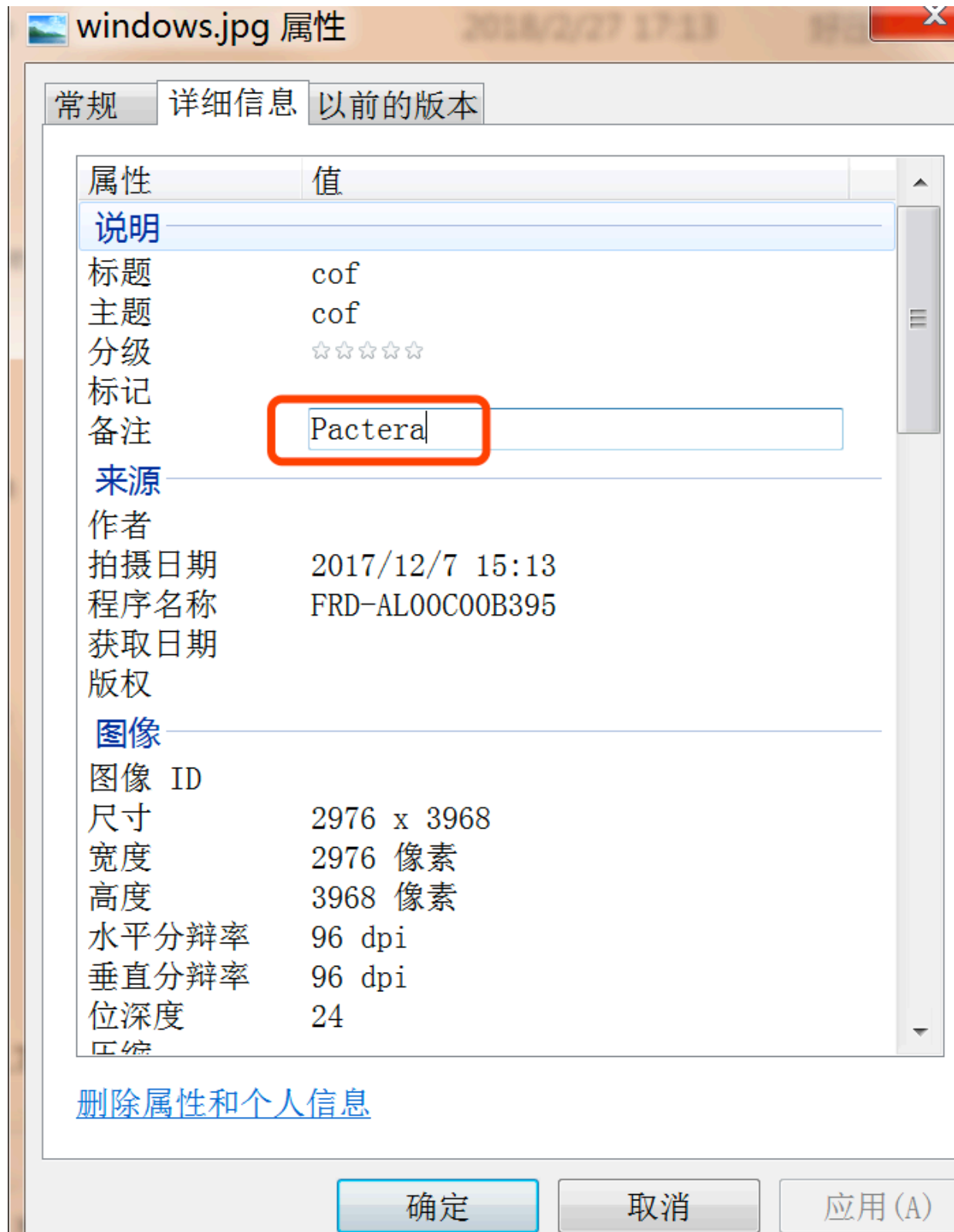
DECIMAL	HEXADECIMAL	DESCRIPTION
-----		
0	0x0	JPEG image data, JFIF standard 1.01
30	0x1E	TIFF image data, big-endian, offset of first image directory: 8
2825118	0x2B1B9E	Linux EXT filesystem, rev 2.0, ext4 filesystem data, UUID=57f8f4bc-abf4-0000-675f-946fc0f9c0f9
7236510	0x6E6B9E	Zip archive data, encrypted at least v2.0 to extract, compressed size: 18214, uncompressed size: 30500, name: file.txt
7254816	0x6EB320	End of Zip archive, footer length: 22

图片中隐藏了一个文件系统和压缩包。

```
binwalk -Me windows.jpg
```

循环提取后得到一个zip文件以及file.txt,此时的file.txt是空文件，因为提取出的zip是带有密码的。

查看Windows.jpg文件的属性，可以找到一些明文的字符串，可作密码测试解压，最终测试出Pactera为密码。



解压后的file.txt也很大，内容的字母重复频率很高，联想到题目中提到的频次，于是对该文件的每一个字符进行统计。

```

data = open('file.txt').read()
counter = dict()
for i in data:
    if counter.has_key(i):
        counter[i] += 1
    else:
        counter[i] = 1

cnt = len(counter)
flag = ''
for i in range(cnt):
    max_num = 0
    max_c = ''
    for x in counter:
        if counter[x] >= max_num:
            max_num = counter[x]
            max_c = x
    #print max_c
    flag = flag + max_c
    del(counter[max_c])
print flag

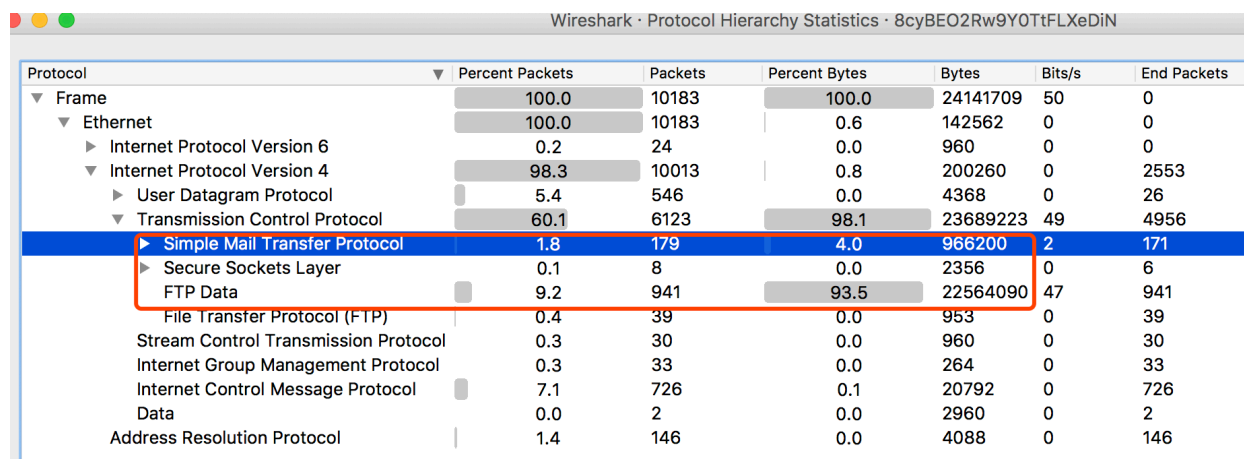
```

脚本的开头是DCTF，手动改为DDCTF即可。

## 流量分析

题目给出了pcap文件，于是拖入wireshark分析。

首先使用wireshark的protocol Hierarchy 统计功能统计协议的类型，这样能对数据包的整体感知。



Protocol	Percent Packets	Packets	Percent Bytes	Bytes	Bits/s	End Packets
▼ Frame	100.0	10183	100.0	24141709	50	0
▼ Ethernet	100.0	10183	0.6	142562	0	0
▶ Internet Protocol Version 6	0.2	24	0.0	960	0	0
▼ Internet Protocol Version 4	98.3	10013	0.8	200260	0	2553
▶ User Datagram Protocol	5.4	546	0.0	4368	0	26
▼ Transmission Control Protocol	60.1	6123	98.1	23689223	49	4956
▶ Simple Mail Transfer Protocol	1.8	179	4.0	966200	2	171
▶ Secure Sockets Layer	0.1	8	0.0	2356	0	6
FTP Data	9.2	941	93.5	22564090	47	941
File Transfer Protocol (FTP)	0.4	39	0.0	953	0	39
Stream Control Transmission Protocol	0.3	30	0.0	960	0	30
Internet Group Management Protocol	0.3	33	0.0	264	0	33
Internet Control Message Protocol	7.1	726	0.1	20792	0	726
Data	0.0	2	0.0	2960	0	2
Address Resolution Protocol	1.4	146	0.0	4088	0	146

通过该图可以发现，使用了tcp的上层协议有SMTP、SSL、FTP，于是重点分析这三类协议的数据。

## smtp

设置smtp协议为筛选条件，并对Length降序排列。

smtp						
No.	Time	Source	Destination	Protocol	Length	Info
9281	878.601203	172.17.0.3	172.17.0.2	SMTP	16450	C: DATA fragment, 16384 bytes
9279	878.601159	172.17.0.3	172.17.0.2	SMTP	16450	C: DATA fragment, 16384 bytes
9277	878.601110	172.17.0.3	172.17.0.2	SMTP	16450	C: DATA fragment, 16384 bytes
9275	878.601065	172.17.0.3	172.17.0.2	SMTP	16450	C: DATA fragment, 16384 bytes
9273	878.601017	172.17.0.3	172.17.0.2	SMTP	16450	C: DATA fragment, 16384 bytes
9271	878.600972	172.17.0.3	172.17.0.2	SMTP	16450	C: DATA fragment, 16384 bytes
9269	878.600917	172.17.0.3	172.17.0.2	SMTP	16450	C: DATA fragment, 16384 bytes
9267	878.600872	172.17.0.3	172.17.0.2	SMTP	16450	C: DATA fragment, 16384 bytes
9265	878.600821	172.17.0.3	172.17.0.2	SMTP	16450	C: DATA fragment, 16384 bytes
9263	878.600777	172.17.0.3	172.17.0.2	SMTP	16450	C: DATA fragment, 16384 bytes
9261	878.600733	172.17.0.3	172.17.0.2	SMTP	16450	C: DATA fragment, 16384 bytes
9259	878.600689	172.17.0.3	172.17.0.2	SMTP	16450	C: DATA fragment, 16384 bytes
9257	878.600637	172.17.0.3	172.17.0.2	SMTP	16450	C: DATA fragment, 16384 bytes
Frame 9189: 14546 bytes on wire (116368 bits), 14546 bytes captured (116368 bits)						
Ethernet II, Src: 02:42:ac:11:00:03 (02:42:ac:11:00:03), Dst: 02:42:ac:11:00:03 (02:42:ac:11:00:03)						

查看TCP Stream的数据流的文本数据，可以发现一段图片的base64编码数据。

```
--B_3598538194_700708737
Content-type: image/png; name="image001.png";
x-mac-creator="4F50494D";
x-mac-type="504E4766"
Content-ID: <image001.png@01D38B05.8079CE40>
Content-disposition: inline;
filename="image001.png"
Content-transfer-encoding: base64

iVBORw0KGGoAAAANSUHEUgAABIwAAAIISCAYAAACu6lSmAAAMKGIDQ1BJQ0MgUHJvZmlsZQAA
SImVVwdYU8kwnlUsKJDQAhGQEnoTpVepoUUQkCrYCEkgocSYEETsyKKCa0HFghVZVFwLYAs
NixYWBTS9WFBVRvXCzZU3iQBdPV7733vfN+5979nzpz5z7kz880AoB7NEYuzUA0AskU5kpjQ
Q0bEpGQm6SFAAAAIwBM4crhScUB0dASAMvt+p7y7Dr2hXLGXx/q5/b+KJo8v5QKARE0cypNy
syE+BADuxhVLcgAg9EC72cwcMcREyBJoSyBBiM3l0F2JPeQ4VYkjFD5xMSyIUwBQoXI4knQA
1OS8mLncdBhHbRnEDiKeUARxE8S+XAGHB/FniEdlZ0+HWN0aYuvU7+Kk/yNm6nBMDid9Gctz
UYhKkFAqzuLM+j/L8b8l00s2NIYZVKpAEHyjz1let8zp4XJMhfickDUyCmItik8KeQp/0X4i
kIXFD/p/4EpZsGaAAQBK5XGCwiE2gNhUlBUZMwj3TR0GsCGGtUfjhDns0GVfLCeZjMYH83j
S4NjhzBHohhL7lmsy4wPGIy5RcBnD8VsZBfEJSp5opdzHqMREktBfFeaGRs+6PM8X8CKHPKR
yGLkn0E/x0CaJCRG6Y0ZZ0uH8sK8BEJ25CC0yBHEhSn7Yl05HAU3XYgz+NKJEUM8efygYGVe
WAFfFD/IHysV5wTGDPpXir0iB/2xJn5WqNxCnGbNdd2qG9vDpxsynxxIM6JjlnYw7Uz000i
lRxwWxABWCAIMIEMaiqYDjKAsK2nvgd+KVtCAAdIQDrgA/tBy1CPREWLCD5jQT74CyI+ka73
C1S08kEuth8Ztiqf9iBN0Zqr6JEJnkCcDcJBFvyWKXqJhkdlAI+hRfjT6FzINQuqv00nG1N9
yEYMJgYRw4ghRBtch/ffvfeI+PSH6or74J5DvL75E54Q2gkPCdcInYRb04QFkh+YM8F40Ak5
haxml/p9drali0aKB+I+MD6MiTNwfWCPu8CRANa/OLYrtH7PVTac8bdaDsYi05BR8aivP9n6
```

将这些base64编码的字符串解码后保存为jpg，可以得到如下图片：

```
MIICXAIBAAKBgQDCm6vZmc1JrVH1AAyGuCuSSZ80+mIQiOUQCvN0HYbj8153JfSQ
LsJIhbRYS7+zZ1oXvPemWQDv/u/tzegt58q4ciNmcVnq1uKiygc6Q0tvT7oiSTyO
vMX/q5iE2iC1YUIHZEKX3BjjNDxrYvLQzPyGD1EY2DZIO6T45FNKYC2VDwIDAQAB
AoGAbtWUKUkx37lLfRq7B5sqjZVKdpBZe4tL0jg6cX5Djd3Uhk1inR9UXVNw4/y4
QGfzYqOn8+Cq7QSoBysHOeXSiPztW2cL09ktPgSlfTQyN6ELNGuiUOYnaTWYZpp/
QbRcZ/eHBu1VQLlk5M6RVs9BLI9X08RA17EcwumiRfWas6kCQQDvqC0dx12wIjwN
czILcoWlig2c2u71Nev9DrWjWHU8eHDuzCJWvOUAHIrkexddWEK2VHd+F13GBCOQ
ZCM4prBjAkEAz+ENahsEjBE4+7H1HdIaw0+goe/45d6A2ew0/1YH6dDZTAzTW9z9
kzV8uz+Mmo5163/JtvwYQcKF39DJGGtqZQJBAKa18XR16fQ9TFL64EQwTQ+tYBzN
+04eTWQCMH3haeQ/0Cd9XyHBUveJ42Be8/jeDcIx7dGLxZKajHbEAfBFnAsCQGq1
AnbJ4Z6opJCGu+UP2c8SC8m0bhZJDe1PRC8IKE28eB6SotgP61ZqaVmQ+HLJ1/wH
/5pfc3AmEyRdfyx6zwUCQCAH4SLJv/kprRz1a1gx8FR5tj4NeHEFFNEgq1gmiwmH
2STT5qZWzQFz8NRe+/otNOHBR2Xk4e8IS+ehIJ3TvyE=
```

MII 开头，让人联想到RSA密钥之类的东西，结合题目给出的提示，于是把这张图片的文本数据导出组合成pem文件。

私钥文件内容如下：

```
-----BEGIN RSA PRIVATE KEY-----
MIICXAIBAAKBgQDCm6vZmc1JrVH1AAyGuCuSSZ8O+mIQiOUQCvN0HYbj8153JfSQ
LsJIhbRYS7+zZloXvPemWQDv/u/tzegt58q4ciNmcVnqluKiygc6QOtvT7oiSTyO
vMX/q5iE2iClYUIHZEKX3BjjNDxrYvLQzPyGD1EY2DZIO6T45FNKYC2VDwIDAQAB
AoGAbtWUKUkx37lLfRq7B5sqjZVKdpBZe4tL0jg6cX5Djd3Uhk1inR9UXVNw4/y4
QGfzYqOn8+Cq7QSoBysHOeXSiPztW2cL09ktPgSlfTQyN6ELNGuiUOYnaTWYZpp/
QbRcZ/eHBulVQLlk5M6RVs9BLI9X08RAL7EcwumiRfWas6kCQQDvqC0dxl2wIjwN
czILcoWLi2c2u71Nev9DrWjWHU8eHDuzCJWvOUAHIrkexddWEK2VHd+F13GBCOQ
ZCM4prBjAkeAz+ENahsEjBE4+7H1HdIaw0+goe/45d6A2ewO/lyH6dDZTAzTW9z9
kzV8uz+Mmo5163/JtvwYQcKF39DJGGtqZQJBAKa18XR16fQ9TFL64EQwTQ+tYBzN
+04eTWQCMH3haeQ/0Cd9XyHBuVeJ42Be8/jeDcIx7dGLxZKajHbEAfBFnAsCQGq1
AnbJ4Z6opJCGu+UP2c8SC8m0bhZJDelPRC8IKE28eB6SotgP61ZqaVmQ+HLJ1/wH
/5pfc3AmEyRdfyx6zwUCQCAH4SLJv/kprRz1a1gx8FR5tj4NeHEFFNEgqlgmiwmH
2STT5qZWzQFz8NRe+/otNOHBR2Xk4e8IS+ehIJ3TvyE=
-----END RSA PRIVATE KEY-----
```

到了这里，联想到pcap中有ssl数据包，于是导入该密钥，最终成功解密ssl。

101...	3790624.058...	172.17.0.3	172.17.0.2	HTTP	169 GET / HTTP/1.1
101...	3790624.059...	172.17.0.2	172.17.0.3	HTTP	995 HTTP/1.1 200 OK (text/html)
101...	3790624.058...	172.17.0.2	172.17.0.3	TLSv1.2	308 New Session Ticket, Change Cipher
101...	3790624.018...	172.17.0.2	172.17.0.3	TLSv1.2	680 Server Hello, Certificate, Server

▶ Frame 10179: 995 bytes on wire (7960 bits), 995 bytes captured (7960 bits)					
▶ Ethernet II, Src: 02:42:ac:11:00:02 (02:42:ac:11:00:02), Dst: 02:42:ac:11:00:03 (02:42:ac:11:00:03)					
▶ Internet Protocol Version 4, Src: 172.17.0.2, Dst: 172.17.0.3					
▶ Transmission Control Protocol, Src Port: 443, Dst Port: 50216, Seq: 857, Ack: 541, Len: 929					
▶ Secure Sockets Layer					
▶ Hypertext Transfer Protocol					
▶ Line-based text data: text/html					

0270	6e 65 20 64 6f 63 75 6d	65 6e 74 61 74 69 6f 6e	ne docum entation
0280	20 61 6e 64 20 73 75 70	70 6f 72 74 20 70 6c 65	and sup port ple
0290	61 73 65 20 72 65 66 65	72 20 74 6f 0a 3c 61 20	ase refe r to.<a
02a0	68 72 65 66 3d 22 68 74	74 70 3a 2f 2f 6e 67 69	href="ht tp://ngi
02b0	6e 78 2e 6f 72 67 2f 22	3e 6e 67 69 6e 78 2e 6f	nx.org/" >nginx.o
02c0	72 67 3c 2f 61 3e 2e 3c	62 72 2f 3e 0a 43 6f 6d	rg</a>.< br/>.Com
02d0	6d 65 72 63 69 61 6c 20	73 75 70 70 6f 72 74 20	mercial support
02e0	69 73 20 61 76 61 69 6c	61 62 6c 65 20 61 74 0a	is avail able at.
02f0	3c 61 20 68 72 65 66 3d	22 68 74 74 70 3a 2f 2f	<a href= "http://
0300	6e 67 69 6e 78 2e 63 6f	6d 2f 22 3e 6e 67 69 6e	nginx.co m/">ngin
0310	78 2e 63 6f 6d 3c 2f 61	3e 2e 3c 2f 70 3e 0a 0a	x.com</a >.</p>..
0320	3c 70 3e 3c 65 6d 3e 54	68 61 6e 6b 20 79 6f 75	<p><em>T hank you
0330	20 66 6f 72 20 75 73 69	6e 67 20 6e 67 69 6e 78	for usi ng nginx
0340	2e 20 44 44 43 54 46 7b	30 36 62 38 34 66 66 64	. DDCTF{ 06b84ffd
0350	37 36 39 62 38 64 36 37	36 30 35 39 38 36 30 32	769b8d67 60598602
0360	36 65 38 33 38 61 33 35	7d 20 3c 2f 65 6d 3e 3c	6e838a35 } </em><
0370	2f 70 3e 0a 3c 2f 62 6f	64 79 3e 0a 3c 2f 68 74	/p>.</bo dy>.</ht
0380	6d 6c 3e 0a		ml>.

flag 显而易见了。

## misc4 安全通信

这是最有趣的一道题目。

题目给出一段程序：

```
#!/usr/bin/env python
import sys
import json
from Crypto.Cipher import AES
from Crypto import Random

def get_padding(rawstr):
    remainder = len(rawstr) % 16
    if remainder != 0:
        return '\x00' * (16 - remainder)
    return ''

def aes_encrypt(key, plaintext):
    plaintext += get_padding(plaintext)
    aes = AES.new(key, AES.MODE_ECB)
    cipher_text = aes.encrypt(plaintext).encode('hex')
    return cipher_text

def generate_hello(key, name, flag):
    message = "Connection for mission: {}, your mission's flag is:
    {}".format(name, flag)
    return aes_encrypt(key, message)

def get_input():
    return raw_input()

def print_output(message):
    print(message)
    sys.stdout.flush()

def handle():
    print_output("Please enter mission key:")
    mission_key = get_input().rstrip()

    print_output("Please enter your Agent ID to secure communications:")
    agentid = get_input().rstrip()
    rnd = Random.new()
    session_key = rnd.read(16)
```



```

flag = '<secret>'
print_output(generate_hello(session_key, agentid, flag))
while True:
    print_output("Please send some messages to be encrypted, 'quit' to
exit:")
    msg = get_input().rstrip()
    if msg == 'quit':
        print_output("Bye!")
        break
    enc = aes_encrypt(session_key, msg)
    print_output(enc)

if __name__ == "__main__":
    handle()

```

通过仔细推敲这段程序，有以下结论：

- 可以得到flag加密后的数据
- flag的相对位置可以通过agentid修改
- 可以无限次用相同密钥对已知数据加密并得到其对应的密文

按照传统的暴力枚举法肯定是不可取的，于是另辟蹊径。

```

message = "Connection for mission: {}, your mission's flag is:
{}".format(name, flag)

```

我想到一种方法，最坏情况尝试次数为  $n \times 95$ ， $n$  为 flag 长度。

## flag长度测定方法

通过不断的加agentid的字符数量，并检测密文【第一次输出】长度变化量来计算。最终测得当agentid有8个字符的时候，密文长度发现变化（抱歉实在没保存中间过程数据），最终可根据agentid的长度、变化后的密文长度等数据测算得flag长度为40。

## 暴力方法

基本思想是每轮大循环只猜一位flag的明文,每轮最多猜95个字符（可见字符范围）。

假设我们使用agenid把message的最后一个分组控制为这样

了：}`\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00`

我们也有它所对应的密文。

注意：}是你不知道的，那么要怎么才能知道第一位是}呢？

通过后面的二次输入猜。

`a\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00`

`b\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00`

c\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00

....

} \x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00

直到枚举到的密文和第一次输出的密文最后一组相同的时候就可以了，这时候枚举到的值就是该位的正确解。

同理，第二轮大循环就该测定}前面的数据呢，而}是已知数据。

尝试枚举：

a} \x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00

b} \x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00

c} \x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00

.....

这样迭代枚举，我们就可以逐渐测试出每一位的flag，每一次只需要把待测定的位置于第一位，已知位右移即可。

于是写如下脚本：

```
from pwn import *
mission_key = 'xxxxxxxxxxxxxxxx'

def e2(conn,p):
    conn.recvuntil("Please send some messages to be encrypted, 'quit' to exit:\n")
    conn.sendline(p)
    return conn.recv().strip()

def get_conn():
    conn = remote('116.85.48.103',5002)
    # auth
    conn.recvuntil('Please enter mission key:\n')
    conn.sendline(mission_key)
    return conn

def set_agendId(conn, agendId):
    conn.recvuntil('Please enter your Agent ID to secure communications:\n')
    conn.sendline(agendaId)
    return conn.recv().strip()

def get_padding(rawstr):
    remainder = len(rawstr) % 16
```



```

    if remainder != 0:
        return '\x00' * (16 - remainder)
    return ''

def guess():
    know_str = ''
    test_str = ''
    know_count = 0
    for x in range(40):
        conn = get_conn()
        hex1 = set_agendId(conn, '*' * (8 + x)) # }000000000
        bin1 = hex1.decode('hex')
        etext = bin1[96:112]
        ehex = etext.encode('hex')
        print 'target:' + ehex
        for i in range(32,127):
            test_str = chr(i) + know_str
            if len(test_str) <= 16:
                test_str = test_str+get_padding(test_str)
            else:
                test_str = test_str[0:16]

            test_hex = e2(conn,test_str)
            #print test_hex
            if test_hex == ehex:
                print chr(i)
                know_str = chr(i) + know_str
                break
        print know_str
        if know_str[0:5] == "DDCTF":
            print know_str

guess()

```

记得替换你的通信key。

## Baby mips

刚开始做这道题的时候对mips指令集一无所知，但是这道题涉及到的mips指令并不是很复杂，简单的进行学习即可上手。

显而易见可得sub\_403168是main函数，main大体流程是：先接收用户16个int数值输入，然后调用sub\_400420检查其输入的合法性，如果合法则按照一定格式输出flag，否则提示错误。

这道题的重点可以放到sub\_400420函数。该函数有一个参数，指向输入的16个int数组。

```

sw $v0, 0x410+var_3EC($fp)
jalx 0xA580BAC
li $v0, 0x85ED
sw $v0, 0x410+var_3E8($fp)
lw $v0, 0x410+var_3E8($fp)
xori $v0, 0xD1
sw $v0, 0x410+var_3E8($fp)
li $v0, 0x47C
sw $v0, 0x410+var_3E4($fp)
jal 0xD440BAC
li $v0, 0x7060
sw $v0, 0x410+var_3E0($fp)
lw $v0, 0x410+var_3E0($fp)
sra $v0, 4
sw $v0, 0x410+var_3E0($fp)
li $v0, 0xA9BC
sw $v0, 0x410+var_3DC($fp)
sc $at, 0x2EB($gp)
sdr $s1, 0x2EB($s5)

```

可以看出这个函数里面有很多无效指令，直接调试运行必定会GG，观察这些错误指令的hex值，发现它们都是以EB 02 固定模式开头。mips指令固定长4个字节，替换EB 02 XX XX为00 00 00 00即可，注意检查EB 02 的时候要与sub\_400420的地址4字节对齐，避免改到满足条件的数据部分。

于是大胆编写脚本修复：

```

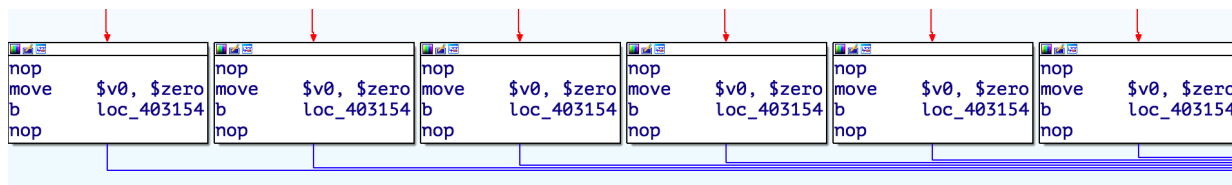
data = open('baby_mips', 'rb').read()
data = bytearray(data)
for i in range(0x420, len(data)):
    if data[i] == 0xeb and data[i+1] == 0x02:
        data[i] = 0
        data[i+1] = 0
        data[i+2] = 0
        data[i+3] = 0
open('mips2', 'wb').write(data)

```

修复后的sub\_400420果真可以被IDA正常分析了。

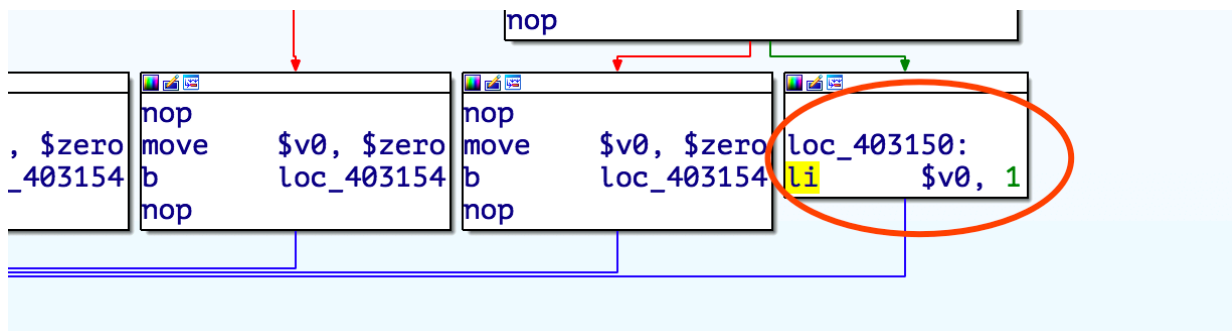
显而易见地，这个函数里面有十六个方程组，其实质就是要求我们解方程，由于方程太多了，就直接用angr暴力求解，注意约束边界。

注意，边界地址不同的bin可能不一样，需要自己抄写。



这种修改v0为0的分支地址都需要加到avoid列表里面。

要find的目标当然是v0置1的分支了。



```
from angr import *
import logging
import IPython
logging.getLogger('angr.manager').setLevel(logging.DEBUG)

p = Project('mips2')
state = p.factory.blank_state(addr=0x400420)

DATA_ADDR = 0xA0000
state.regs.a0 = DATA_ADDR
for i in range(16*4):
    vec = state.solver.BVS("c{}".format(i),8,explicit_name=True)
    cond = state.solver.And(vec>=32,vec<=126) # low byte
    state.memory.store(DATA_ADDR+i,vec)
    if i % 4 == 0:
        pass
        #state.add_constraints(cond)

sm = p.factory.simulation_manager(state)
res = sm.explore(find=0x403150,avoid=[0x403644,0x401940,0x0401ADC,0x401C74,
,0x401E10 ,0x401FA8,0x402144
,0x4022DC,0x402478,0x402610,0x4027A8,0x402940,0x402AD8,0x402C74,0x402E10,0x
402FA8,0x403144])
# 这些地址不同人的bin会不一样。
found = res.found[0]
mem = found.memory.load(DATA_ADDR,16*4)
print found.solver.eval(mem)
```

```

print '#####'
flag = ''
for i in range(16):
    v = found.memory.load(DATA_ADDR + 4*i,1)
    flag = flag + found.solver.eval(v,cast_to=str)
print flag

```

angr执行的对象也应该是修复后的mips elf。

## 黑盒破解

这道题逆向过程比较简单，拼接过程需要耐心。

这道题目的逻辑很清晰，黑盒的内部定义了几条含有特殊意义的指令，需要精心构造指令序列使生成Bingo字符串，既然是虚拟机类型的题目，就应该先去找handler，然后逐一摸清指令的含义。

定位到main函数后可以快速发现成功的分支：

```

5 |         if ( byte_603F00 )
5 |             printf("Success!\nYour flag is %s\n", &ptr);
7 |         else
3 |             puts("Failed!");
9 |     }

```

byte\_603F00 条件，因此查找byte\_603F00的所有引用。

```
Structure Data Unexplored External Symbol
IDA View-A Pseudocode-A Hex View
11 int v9; // [rsp+30h] [rbp-10h]
12 char v10; // [rsp+34h] [rbp-Ch]
13 unsigned __int64 v11; // [rsp+38h] [rbp-8h]
14
15 v11 = __readfsqword(0x28u);
16 *s = 0LL;
17 v8 = 0LL;
18 v9 = 0;
19 v10 = 0;
20 if ( !a1 )
21     return 0LL;
22 if ( *(a1 + 664) == 115 )
23 {
24     *(a1 + 664) = 0;
25     for ( i = 0; i <= 19; ++i )
26         s[i] = (*(a1 + 8) + *(a1 + 288) + i);
27     puts(s); // Binggo
28     v2 = strlen(s);
29     v5 = 0;
30     if ( sub_400C74(s, v2) == 76565983 )
31     {
32         v3 = strlen(s);
33         if ( sub_400BC6(s, v3) == 717892254 )
34         {
35             v4 = strlen(s);
36             if ( sub_400C1D(s, v4) == 437502268 )
37                 v5 = 1;
38         }
39     }
40     if ( v5 )
41         byte_603F00 = 1;
42     result = 1LL;
43 }
44 else
45 {
46     printf("%c\n", *(a1 + 665));
47     result = 1LL;
48 }
49 return result;
50 }
```

再查找个函数的引用，发现有一张函数地址表：

.data:0000000000603840	off_603840	dq offset sub_400DC1
.data:0000000000603848		dq offset sub_400E7A
.data:0000000000603850		dq offset sub_400F3A
.data:0000000000603858		dq offset sub_401064
.data:0000000000603860		dq offset sub_4011C9
.data:0000000000603868		dq offset sub_40133D
.data:0000000000603870		dq offset sub_4012F3
.data:0000000000603878		dq offset sub_4014B9
.data:0000000000603880		dq offset sub_400CF1
.data:0000000000603888		dq offset sub_400E22
.data:0000000000603890		dq offset sub_400FA0
.data:0000000000603898		dq offset sub_400EEA
.data:00000000006038A0		dq offset sub_40121D
.data:00000000006038A8		dq offset sub_4010F5

这个地址表有什么用呢？后面会说到。

继续分析main函数，可以找到指令解析函数（很明显的特征：循环、动态调用函数）

```

1 |__int64 __fastcall sub_401A48(__int64 a1)
2 |{
3 |    char v2; // [rsp+13h] [rbp-1Dh]
4 |    int i; // [rsp+14h] [rbp-1Ch]
5 |    signed int j; // [rsp+18h] [rbp-18h]
6 |
7 |    if ( a1 && a1 != -16 )
8 |    {
9 |        for ( i = 0; i < strlen((a1 + 16)); ++i )
10 |        {
11 |            v2 = *(a1 + i + 16);
12 |            if ( i + 1 != strlen((a1 + 16)) )
13 |                *(a1 + 664) = *(a1 + i + 1 + 16);
14 |            for ( j = 0; j <= 8; ++j )
15 |            {
16 |                if ( byte_603900[v2] == *(a1 + *(a1 + 4 * (j + 72LL) + 8) + 408) )
17 |                {
18 |                    *(a1 + 672) = *(a1 + 8 * (*(a1 + 4 * (j + 72LL) + 8) + 84LL) + 8);
19 |                    (*(a1 + 672))(a1); | // 调用对于指令执行函数
20 |                }
21 |            }
22 |        }
23 |    }
24 |    return 0LL;
25 |}

```

这个函数的参数是输入的Pascode的字符串，循环则是循环从passcode中读取指令并执行。

通过循环次数可知，一共有九条指令，又通过对j的待定可以测出v2以及对应的函数值。

v2就是输入的每一位。

\$8Ct0Eu#; 就是所有的指令序列了。

伪代码如下

```

r0 = 280 (相对于vm环境基址的偏移)
r1 = 288
r2 = 665
r3 = 664
r4 = 292

```

```

r5 = 280

sub_400DC1:  //$指令
    r2 = sbox[r1]

sub_400E7A: //8指令
    sbox[r1] = r2

sub_400F3A://C指令
    r2 = r2 + r3 - 33

sub_401064://t
    r2 = r2 - r3 + 33
    r2 += 1;

sub_4011C9: //0
    r1 += 1

sub_40133D://E  最终调用这个。
    check

sub_4012F3://u
    r1 -= 1

sub_4014B9://#
    sbox[r1] = input[r1 + r3 - 48] - 49;

sub_400CF1://;
for ( i = 0; r3 > i; ++i )
    r1 += 1
if (r3 - 16 > 89 )
    return 0LL;
sbox[r1] = input[r1 + r3 - 48] - 49;

```

E指令为检测结果合法性的指令，代码如下：



```

11  int v9; // [rsp+30h] [rbp-10h]
12  char v10; // [rsp+34h] [rbp-Ch]
13  unsigned __int64 v11; // [rsp+38h] [rbp-8h]
14
15  v11 = __readfsqword(0x28u);
16  *s = 0LL;
17  v8 = 0LL;
18  v9 = 0;
19  v10 = 0;
20  if ( !a1 )
21      return 0LL;
22  if ( *(a1 + 664) == 115 )
23  {
24      *(a1 + 664) = 0;
25      for ( i = 0; i <= 19; ++i )
26          s[i] = (*(a1 + 8) + *(a1 + 288) + i);
27      puts(s); // Binggo
28      v2 = strlen(s);
29      v5 = 0;
30      if ( sub_400C74(s, v2) == 76565983 )
31      {
32          v3 = strlen(s);
33          if ( sub_400BC6(s, v3) == 717892254 )
34          {
35              v4 = strlen(s);
36              if ( sub_400C1D(s, v4) == 437502268 )
37                  v5 = 1;
38          }
39      }
40      if ( v5 )
41          byte_603F00 = 1;
42      result = 1LL;
43  }
44  else
45  {
46      printf("%c\n", *(a1 + 665));
47      result = 1LL;
48  }
49  return result;
50 }

```

使用\$8Ct0Eu#; 序列构造一串指令，获得sbox[r1] = "Binggo"

sbox[] = PaF0!&Prv}H{ojDQ#7v=2o,wK@8Em/ry2w|n\_yWNz-sW5vs!>Q8?M>Ue\*}<&!c\_v?  
Wz98>79U}LKR&O-  
3,s2=)!ac'fm,v#WbU5Cv:M"WQa"i%BLT5<qBl#AK%60ixtn+Kn^Ontp]O9Sr(Nm27H}QM><iE,x)yN?  
ok]L##jE"-4i-  
t);AzD4"^IEC3TrZ%55)|3:.YLv{7(cnxA\_AsYRXyD&|3U\_z^yR/Kpqp6lT/pdv8tp,\^Y7T+oLsvf. 寻找一  
组函数调用顺序，使得sbox[r1] = "Binggo" 调用顺序用注释后面的符号表示，最后两个字节一定是  
Es。

## 被隐藏的真实

这道题目一共需要输入三组数据，才能得到最终的flag。

每一轮输入都需要经过get\_pwd 验证，每一轮的get\_pwd 细节又有所不同。

三轮输入共同的是：输入数据均为HEX字符串，输入后会转成byte array，转后的数据会被加入到sha1计算。

最终的flag是由三轮输入的数据拼接起来的sha1确定。

## 第一关

通过动态调试可以简单快速地跟踪到对应的函数，第一关实际就是将输入的数值与常量DEADBEEF进行比较，由于字节序的缘故，所以输入的时候要倒过来，EFBEADDE。

## 第二关

通过动态调试的方法，可以确定第二关get\_pwd调用了4046B9 处的函数。

```
1 // write access to const memory has been detected, the output may be wrong!
2 signed __int64 __fastcall sub_4046B9(Bitcoin *a1, __int64 a2)
3 {
4     Bitcoin::thiz = a1;
5     *((_QWORD *)Bitcoin::thiz + 4) = a2;
6     *((_QWORD *)Bitcoin::thiz + 5) = sub_4047E2;
7     qword_6D0A08[126] = (__int64)&dword_404A14;
8     b58e(*((const unsigned __int8 **)Bitcoin::thiz + 4), 0x15uLL, &buf);
9     JUMPOUT(memcmp(&buf, baddr, 0x23uLL), 0, &loc_40481E);
10    return 0xCAFEBAEELL;
11 }
```

s2: char[35]  
"1EaPk4jQBt7yWEyG6hRaFcnZwHga9tgp3J"

这个函数把输入的byte数组作为参数调用b58e函数，b58e是一个带校验的base58编码函数，最后把base58编码的结果与1EaPk4jQBt7yWEyG6hRaFcnZwHga9tgp3J比较，该结果是带签名的，所以应该把尾部的4字节签名去掉，最后解码得到HEX STR：

0094ea58adc3aa65f5c26f6ed27d3fdb0d093dc17f

## 第三关

40492E

这一关实际就是一个比特币的交易hash生成算法，已知hash值求输入。

暴力解肯定是很困难的。

题目给出的提示关键信息是1A1zP1eP5QGefi2DMPTfTL5SLmv7DivfNa

拿到百度上查一下，这是一个比特币地址，而且能查询到很多信息。

```
GetHash() =
0x000000000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce26f
hashMerkleRoot =
0x4a5e1e4baab89f3a32518a88c31bc87f618f76673e2cc77ab2127b7afdeda33b
txNew.vin[0].scriptSig = 486604799 4
0x736B6E616220726F662074756F6C69616220646E6F63657320666F206B6E697262206E6F2
0726F6C6C65636E61684320393030322F6E614A2F33302073656D695420656854
txNew.vout[0].nValue = 5000000000
```

```

txNew.vout[0].scriptPubKey =
0x5F1DF16B2B704C8A578D0BBAF74D385CDE12C11EE50455F3C438EF4C3FBCF649B6DE611FE
AE06279A60939E028A8D65C10B73071A6F16719274855FEB0FD8A6704 OP_CHECKSIG
block.nVersion = 1
block.nTime     = 1231006505
block.nBits     = 0x1d00ffff
block.nNonce    = 2083236893

CBlock(hash=0000000000019d6, ver=1, hashPrevBlock=000000000000000,
hashMerkleRoot=4a5e1e, nTime=1231006505, nBits=1d00ffff, nNonce=2083236893,
vtx=1)
    CTransaction(hash=4a5e1e, ver=1, vin.size=1, vout.size=1, nLockTime=0)
        CTxIn(COutPoint(000000, -1), coinbase
04ffff001d0104455468652054696d65732030332f4a616e2f32303039204368616e63656c6
c6f72206f6e206272696e6b206f66207365636f6e64206261696c6f757420666f722062616e
6b73)
            CTxOut(nValue=50.00000000, scriptPubKey=0x5F1DF16B2B704C8A578D0B)
            vMerkleTree: 4a5e1e

```

我们可以看到的是，这里的hash和题目中对比的hash是一模一样的，所以我们只需要根据hash生成算法来提供输入即可，所有的参数都能在互联网上查到，于是此题告破。