【jarvisoj刷题之旅】pwn题目Tell Me Something的writeup

原创

iqiqiya 于 2018-10-14 21:22:30 发布 762 收藏 1
分类专栏： 我的pwn之路 ------jarvisojCTF 我的CTF之路 我的CTF进阶之路 文章标签： 【jarvisoj刷题之旅】pwn题目Tell Me Som pwn题目Tell Me Something的writeup Tell Me Something的writeup

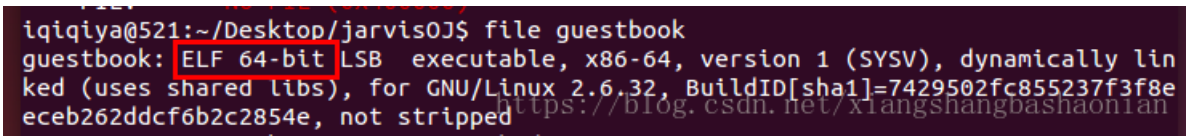

我的pwn之路 同时被 3 个专栏收录
8 篇文章 0 订阅
订阅专栏


------jarvisojCTF
2 篇文章 0 订阅
订阅专栏


我的CTF之路
92 篇文章 5 订阅
订阅专栏

题目信息：

file一下 发现是64位的ELF



checksec检查下安全性

```
iqiqiya@521:~/Desktop/jarvisOJ$ checksec guestbook
[*] '/home/iqiqiya/Desktop/jarvisOJ/guestbook'
    Arch:     amd64-64-little
    RELRO:    No RELRO
    Stack:    No canary found
    NX:       NX enabled
    PIE:      No PIE (0x400000)
```

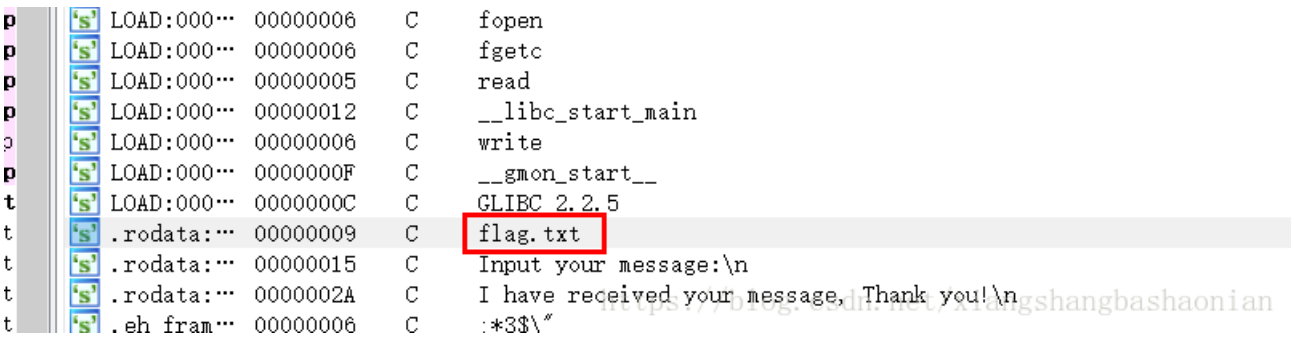

objdump -t 文件名　可以查看符号表

```
iqiqiya@521:~/Desktop/jarvisOJ$ objdump -t guestbook

guestbook:     file format elf64-x86-64

SYMBOL TABLE:
0000000000400200 l    d  .interp 0000000000000000              .interp
000000000040021c l    d  .note.ABI-tag 0000000000000000              .note.ABI-tag
000000000040023c l    d  .note.gnu.build-id 0000000000000000              .note.gnu.build-id
0000000000400260 l    d  .gnu.hash 0000000000000000              .gnu.hash
0000000000400280 l    d  .dynsym 0000000000000000              .dynsym
0000000000400328 l    d  .dynstr 0000000000000000              .dynstr
0000000000400378 l    d  .gnu.version 0000000000000000              .gnu.version
0000000000400388 l    d  .gnu.version_r 0000000000000000              .gnu.version_r
00000000004003a8 l    d  .rela.dyn 0000000000000000              .rela.dyn
00000000004003c0 l    d  .rela.plt 0000000000000000              .rela.plt
0000000000400450 l    d  .init 0000000000000000              .init
0000000000400470 l    d  .plt 0000000000000000              .plt
00000000004004e0 l    d  .text 0000000000000000              .text
0000000000400704 l    d  .fini 0000000000000000              .fini
0000000000400710 l    d  .rodata 0000000000000000              .rodata
0000000000400768 l    d  .eh_frame_hdr 0000000000000000              .eh_frame_hdr
00000000004007b0 l    d  .eh_frame 0000000000000000              .eh_frame
00000000006008d8 l    d  .init_array 0000000000000000              .init_array
00000000006008e0 l    d  .fini_array 0000000000000000              .fini_array
00000000006008e8 l    d  .jcr 0000000000000000              .jcr
00000000006008f0 l    d  .dynamic 0000000000000000              .dynamic
0000000000600ac0 l    d  .got 0000000000000000              .got
0000000000600ac8 l    d  .got.plt 0000000000000000              .got.plt
0000000000600b10 l    d  .data 0000000000000000              .data
0000000000600b20 l    d  .bss 0000000000000000              .bss
0000000000000000 l    d  .comment 0000000000000000              .comment
0000000000000000 l    df *ABS* 0000000000000000              crtstuff.c
00000000006008e8 l     O .jcr 0000000000000000              __JCR_LIST__
0000000000400550 l     F .text 0000000000000000              deregister_tm_clones
0000000000400590 l     F .text 0000000000000000              register_tm_clones
00000000004005d0 l     F .text 0000000000000000              __do_global_dtors_aux
0000000000600b20 l     O .bss 0000000000000001              completed.6661
00000000006008e0 l     O .fini_array 0000000000000000              __do_global_dtors_aux_fini_array_entry
00000000004005f0 l     F .text 0000000000000000              frame_dummy
00000000006008d8 l     O .init_array 0000000000000000              __frame_dummy_init_array_entry
0000000000000000 l    df *ABS* 0000000000000000              pwn100.c
0000000000000000 l    df *ABS* 0000000000000000              crtstuff.c
00000000004008d0 l     O .eh_frame 0000000000000000              __FRAME_END__
00000000006008e8 l     O .jcr 0000000000000000              __JCR_END__
0000000000000000 l    df *ABS* 0000000000000000
00000000006008e0 l       .init_array 0000000000000000              __init_array_end
00000000006008f0 l     O .dynamic 0000000000000000              _DYNAMIC
00000000006008d8 l       .init_array 0000000000000000              __init_array_start
0000000000600ac8 l     O .got.plt 0000000000000000              _GLOBAL_OFFSET_TABLE_
0000000000400700 g     F .text 0000000000000002              __libc_csu_fini
```

```
0000000000000000  w      *UND* 0000000000000000              _ITM_deregisterTMCloneTable
0000000000600b10  w      .data 0000000000000000              data_start
0000000000000000       F *UND* 0000000000000000              write@@GLIBC_2.2.5
0000000000600b20 g       .data 0000000000000000              _edata
0000000000400704 g     F .fini 0000000000000000              _fini
0000000000000000       F *UND* 0000000000000000              fgetc@@GLIBC_2.2.5
0000000000000000       F *UND* 0000000000000000              read@@GLIBC_2.2.5
0000000000000000       F *UND* 0000000000000000              __libc_start_main@@GLIBC_2.2.5
0000000000600b10 g       .data 0000000000000000              __data_start
0000000000000000  w      *UND* 0000000000000000              __gmon_start__
0000000000600b18 g     O .data 0000000000000000              .hidden __dso_handle
0000000000400710 g     O .rodata 0000000000000004            _IO_stdin_used
0000000000400690 g     F .text 0000000000000065              __libc_csu_init
0000000000600b28 g       .bss 0000000000000000              _end
0000000000400526 g     F .text 0000000000000000              _start
0000000000600b20 g       .bss 0000000000000000              __bss_start
00000000004004e0 g     F .text 0000000000000046              main
0000000000000000       F *UND* 0000000000000000              fopen@@GLIBC_2.2.5
0000000000000000  w      *UND* 0000000000000000              _Jv_RegisterClasses
0000000000400670 g     F .text 000000000000001e              readmessage
0000000000600b20 g     O .data 0000000000000000              .hidden __TMC_END__
0000000000000000  w      *UND* 0000000000000000              _ITM_registerTMCloneTable
0000000000400450 g     F .init 0000000000000000              _init
0000000000400620 g     F .text 000000000000004a              good_game
```

这里可以看到good_game很可疑

IDA 分析一下也可以看到flag.txt，read,read等



分别双击这个Input your message 和flag.txt 的引用　然后F5

```
1 int good_game()
2 {
3   FILE *v0; // rbx
4   int result; // eax
5   char buf; // [rsp+Fh] [rbp-9h]
6
7   v0 = fopen("flag.txt", "r");
8   while ( 1 )
9   {
10    result = fgetc(v0);
11    buf = result;
12    if ( (_BYTE)result == -1 )
13      break;
14    write(1, &buf, 1uLL);
15  }
16  return result;
17 }
```


那么我们就可以判断出good_game()这个函数的作用就是打开flag.txt并读取内容输出

main()是没有调用它的　也就是说程序正常是无法执行good_game()的

但是可以看到main()中有调用read()，那么我们就可以通过向缓冲区中输入大于 buf 长度的数据，这样多余的数据就会溢出，并覆盖栈上的一些数据，那么我们就将main()函数的返回地址改成good_game()的入口地址，这样就可以在main()执行完之后输出flag.txt内容了

双击main()中的v4 就可以查看main函数的栈帧

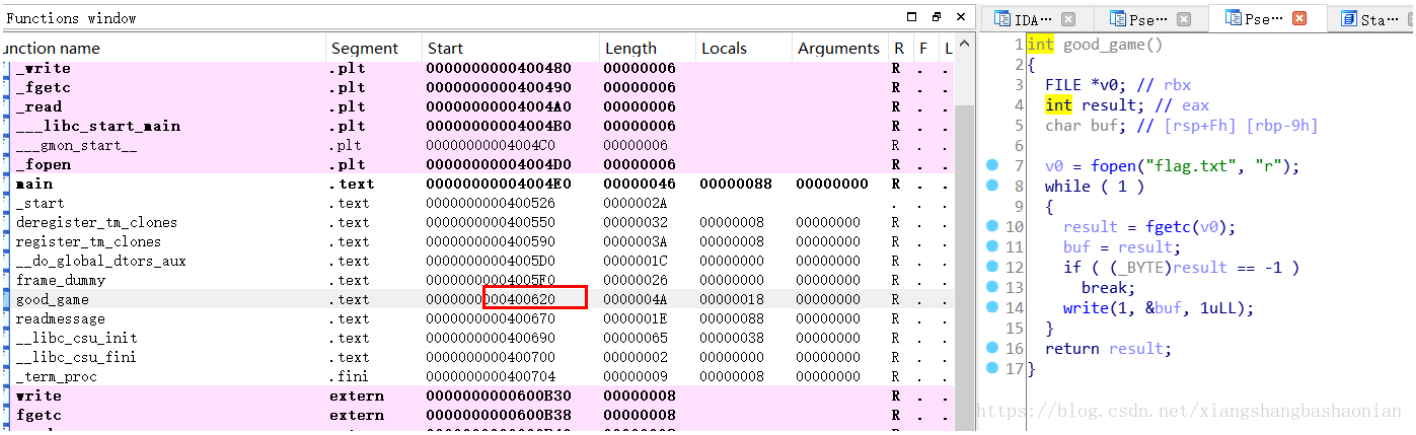
```
-0000000000000088 ; D/A/*   : change type (data/ascii/array)
-0000000000000088 ; N       : rename
-0000000000000088 ; U       : undefine
-0000000000000088 ; Use data definition commands to create local variables and function arguments.
-0000000000000088 ; Two special fields " r" and " s" represent return address and saved registers.
-0000000000000088 ; Frame size: 88; Saved regs: 0; Purge: 0
-0000000000000088 ;
-0000000000000088
-0000000000000088                 db ? ; undefined
-0000000000000087                 db ? ; undefined
-0000000000000086                 db ? ; undefined
-0000000000000085                 db ? ; undefined
```



```
-0000000000000012                 db ? ; undefined
-0000000000000011                 db ? ; undefined
-0000000000000010                 db ? ; undefined
-000000000000000F                 db ? ; undefined
-000000000000000E                 db ? ; undefined
-000000000000000D                 db ? ; undefined
-000000000000000C                 db ? ; undefined
-000000000000000B                 db ? ; undefined
-000000000000000A                 db ? ; undefined
-0000000000000009                 db ? ; undefined
-0000000000000008                 db ? ; undefined
-0000000000000007                 db ? ; undefined
-0000000000000006                 db ? ; undefined
-0000000000000005                 db ? ; undefined
-0000000000000004                 db ? ; undefined
-0000000000000003                 db ? ; undefined
-0000000000000002                 db ? ; undefined
-0000000000000001                 db ? ; undefined
+0000000000000000  r              db 8 dup(?)
+0000000000000008
+0000000000000008 ; end of stack variables
SP+000000000000006F
```

可以看到，用于缓冲用户输入的字符数组距离 `main` 的返回地址有 0x88 个字节
那么我们需要输入 0x88 (136) 个字节的数据，再输入 good_game 的地址，这样程序返回的时候就会跳转到 good_game 去执行

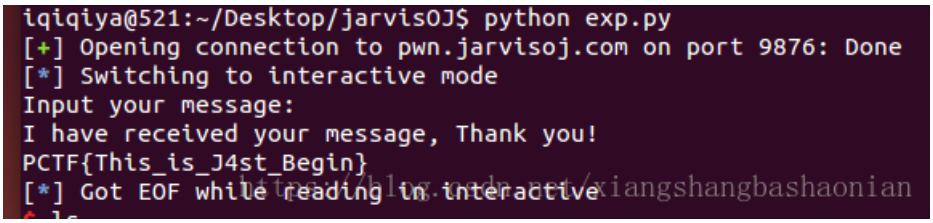
通过 ida 来查看 good_game 的地址



需要转换成小端序的形式，可以利用zio这个库

我直接用pwntools

脚本如下：



参考链接：https://www.jianshu.com/p/16901e0e51e4