登录

Pwn with File结构体（二）

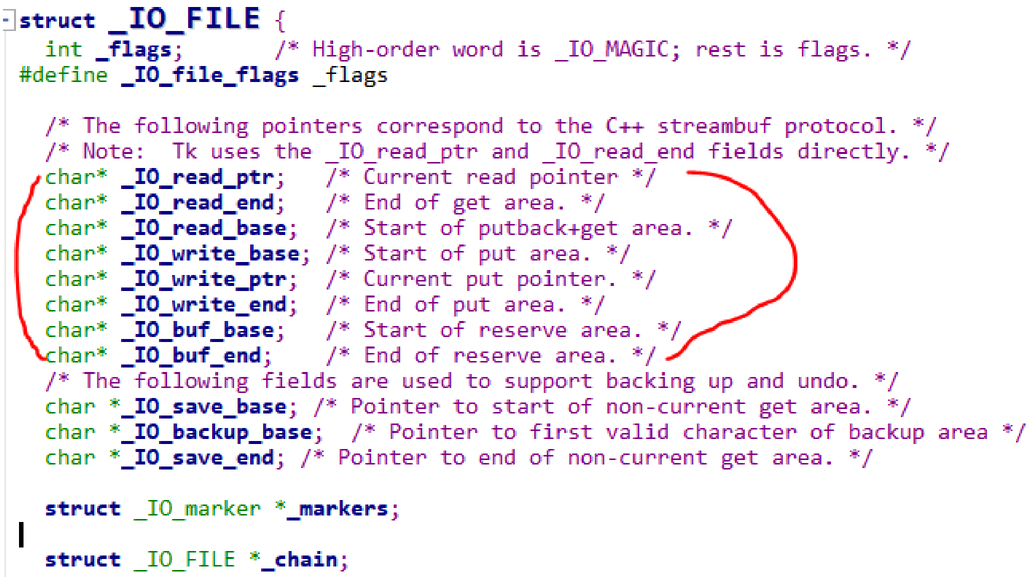
hackedbylh / 2017-12-15 09:48:21 / 浏览数 3420 技术文章 技术文章 顶(3) 踩(0)

前言

最新版的 `libc` 中会对 `vtable` 检查，所以之前的攻击方式，告一段落。下面介绍一种，通过修改 `_IO_FILE` 实现任意地址读和任意地址写的方式。

正文

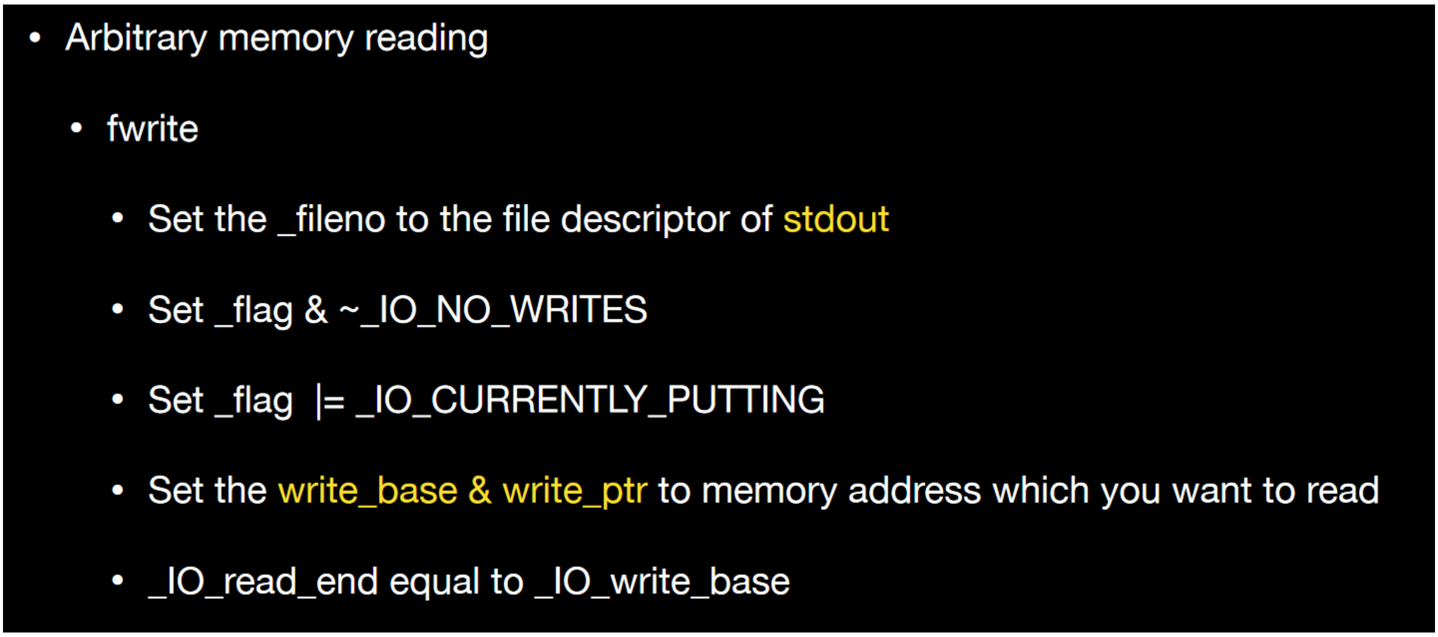
`_IO_FILE` 通过这些指针，来读写数据。



```
struct _IO_FILE {
  int _flags;           /* High-order word is _IO_MAGIC; rest is flags. */
#define _IO_file_flags _flags

  /* The following pointers correspond to the C++ streambuf protocol. */
  /* Note:  Tk uses the _IO_read_ptr and _IO_read_end fields directly. */
  char* _IO_read_ptr;   /* Current read pointer */
  char* _IO_read_end;   /* End of get area. */
  char* _IO_read_base;  /* Start of putback+get area. */
  char* _IO_write_base; /* Start of put area. */
  char* _IO_write_ptr;  /* Current put pointer. */
  char* _IO_write_end;  /* End of put area. */
  char* _IO_buf_base;   /* Start of reserve area. */
  char* _IO_buf_end;    /* End of reserve area. */
  /* The following fields are used to support backing up and undo. */
  char *_IO_save_base; /* Pointer to start of non-current get area. */
  char *_IO_backup_base;  /* Pointer to first valid character of backup area */
  char *_IO_save_end; /* Pointer to end of non-current get area. */

  struct _IO_marker *_markers;

  struct _IO_FILE *_chain;
```

如果我们修改了它们，然后通过一些文件读写函数时，我们就能实现 任意地址读写。

任意地址读

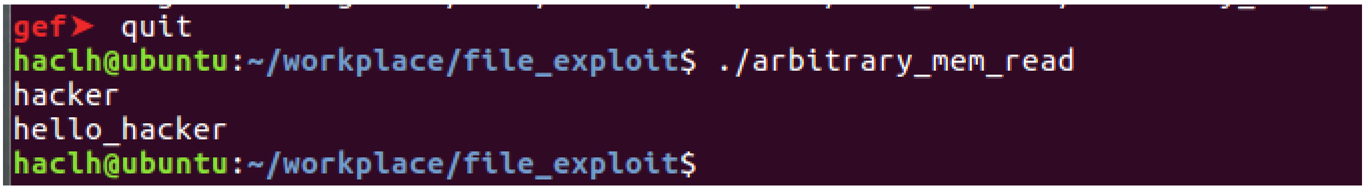


- Arbitrary memory reading
  - fwrite
    - Set the _fileno to the file descriptor of stdout
    - Set _flag & ~_IO_NO_WRITES
    - Set _flag |= _IO_CURRENTLY_PUTTING
    - Set the write_base & write_ptr to memory address which you want to read
    - _IO_read_end equal to _IO_write_base

代码示例

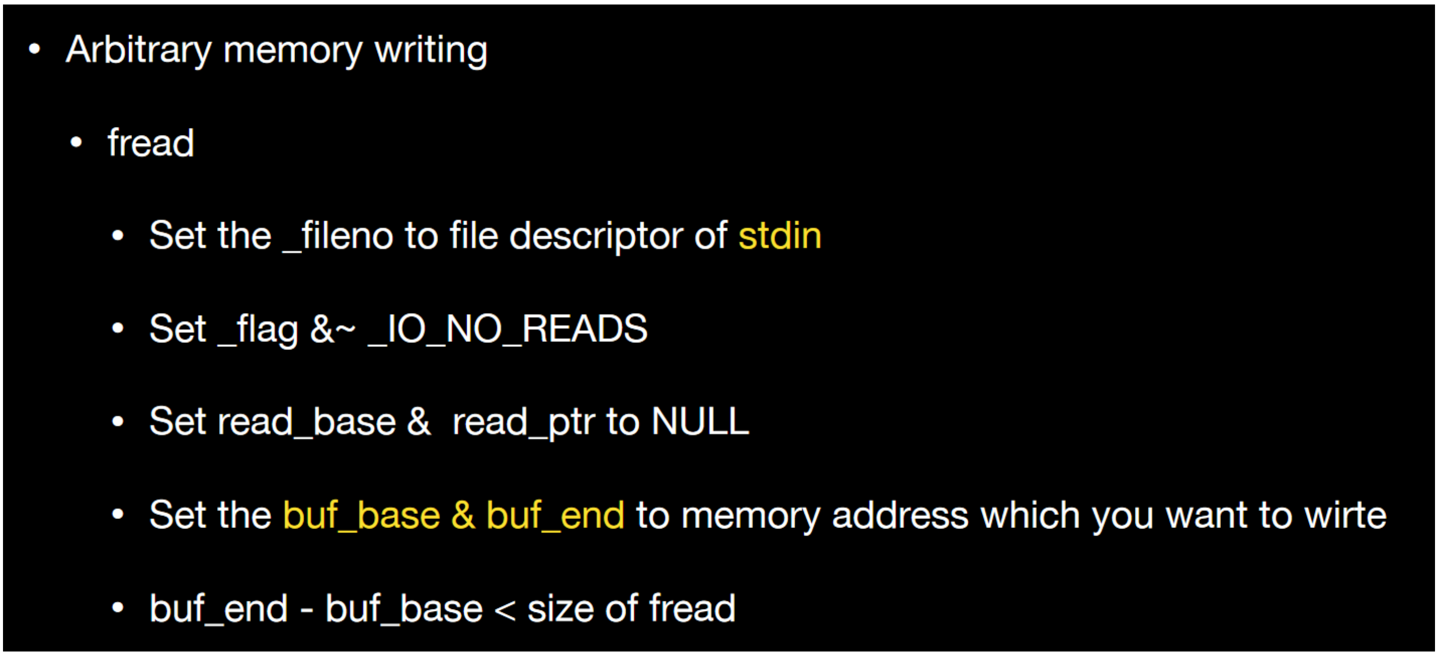
```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char * argv[])
{
   FILE *fp;
   char *msg = "hello_file";
```

```
    char *buf = malloc(100);
    read(0, buf, 100);
    fp = fopen("key.txt", "rw");

    // ■■ flag ■■ check
    fp->_flags &= ~8;
    fp->_flags |= 0x800;

    // _IO_write_base write■■■■■■■■ _IO_write_ptr  write■■■■■■■
    fp->_IO_write_base = msg;
    fp->_IO_write_ptr = msg + 6;

    //■■■■
    fp->_IO_read_end = fp->_IO_write_base;

    // write ■■■ ■■■■■■ 1 --> ■■■■
    fp->_fileno = 1;
    fwrite(buf, 1, 100, fp);

    return 0;
}
```



```
gef➤  quit
haclh@ubuntu:~/workplace/file_exploit$ ./arbitrary_mem_read
hacker
hello_hacker
haclh@ubuntu:~/workplace/file_exploit$
```

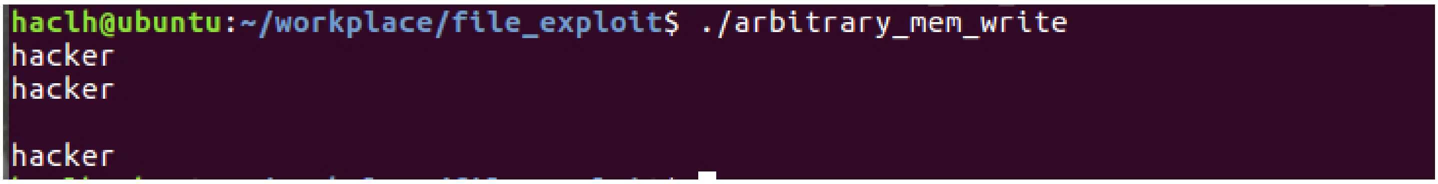
任意地址写



- Arbitrary memory writing
  - fread
    - Set the _fileno to file descriptor of stdin
    - Set _flag &~ _IO_NO_READS
    - Set read_base & read_ptr to NULL
    - Set the buf_base & buf_end to memory address which you want to wirte
    - buf_end - buf_base < size of fread

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char * argv[])
{
    FILE *fp;
    char msg[100];

    char *buf = malloc(100);
    fp = fopen("key.txt", "rw");

    // ■■ flag ■■ check
    fp->_flags &= ~4;

    // _IO_buf_base buffer ■■■■■■ _IO_buf_end  buffer ■■■■■
```

```
   // fread ■■■■■■ [_IO_buf_base, _IO_buf_end] ■■■ buffer
   // ■■■■■■■ buffer
   fp->_IO_buf_base = msg;
   fp->_IO_buf_end = msg + 100;

   // ■■ ■■■■■■ 0---> stdin, ■■■■■■■■
   fp->_fileno = 0;
   fread(buf, 1, 6, fp);

   puts(msg);
   puts(buf);

   return 0;
}
```



```
haclh@ubuntu:~/workplace/file_exploit$ ./arbitrary_mem_write
hacker
hacker

hacker
```

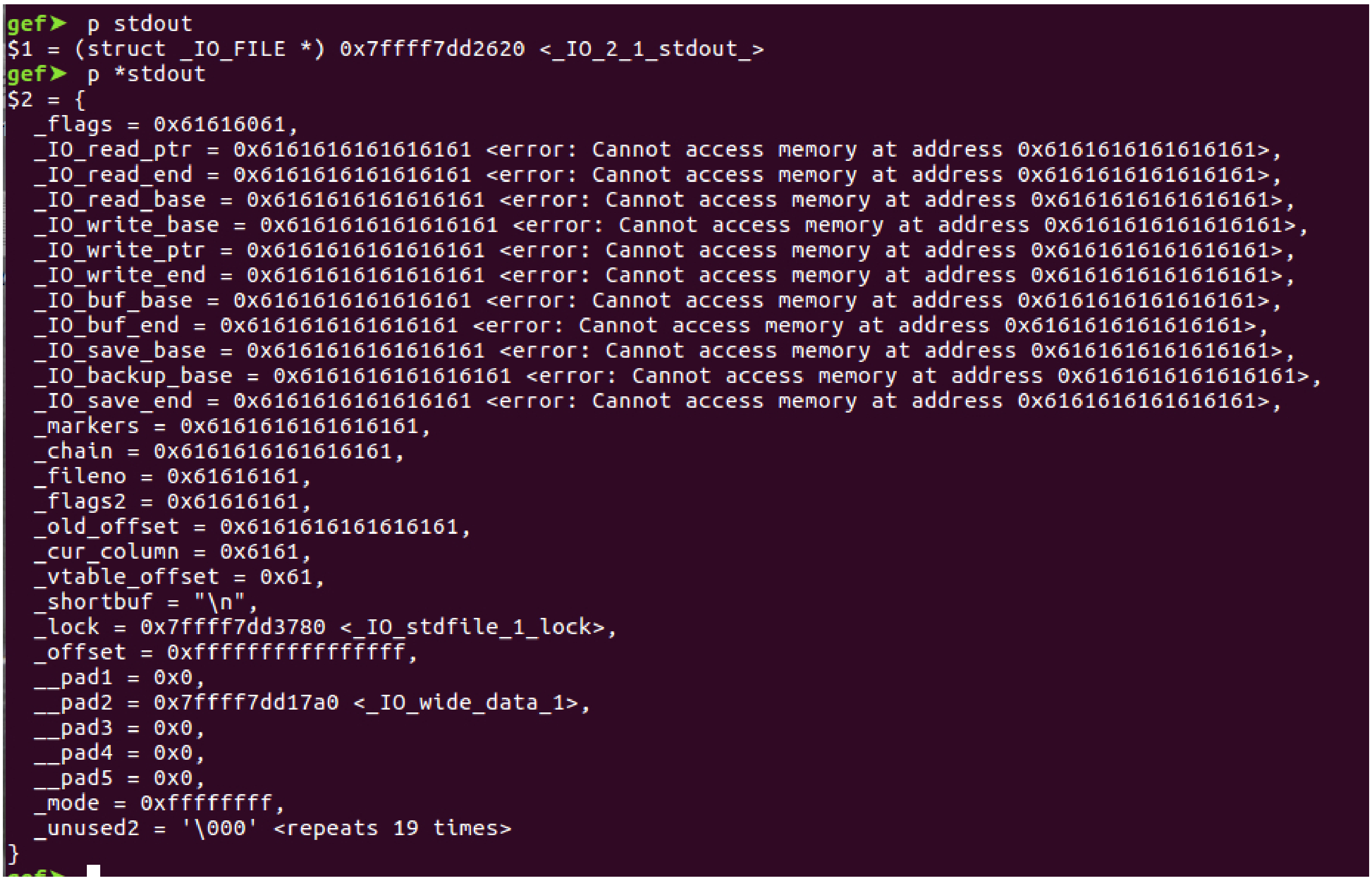
利用 stdin / stdout 任意地址写/ 读

puts, scanf 等一批系统函数默认使用的 stdin , stdout ,stderr 等结构体进行操作，通过修改这些结构体的内容，可以更方便的实现任意地址读，任意地址写。

stdin 也是 _IO_FILE 结构体

```
#include <stdio.h>
#include <stdlib.h>


int global_val = 0xaabbccdd;


int main(int argc, char * argv[])
{
   FILE *fp;
   int var;

   fp = stdin;

   fp->_flags &= ~4;

   fp->_IO_buf_base = stdout;
   fp->_IO_buf_end = stdout + 100;


   scanf("%d",&var);

   printf("0x%x\n", global_val);

   return 0;
}
```

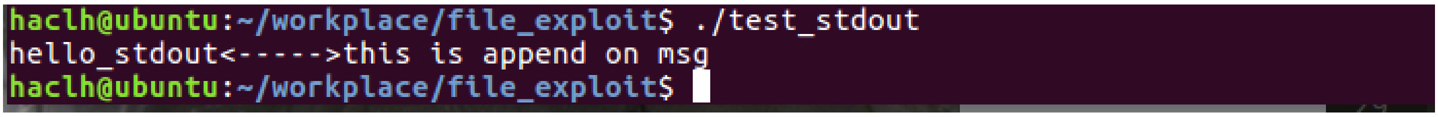
运行之



```
gef➤  p stdout
$1 = (struct _IO_FILE *) 0x7ffff7dd2620 <_IO_2_1_stdout_>
gef➤  p *stdout
$2 = {
  _flags = 0x61616061,
  _IO_read_ptr = 0x6161616161616161 <error: Cannot access memory at address 0x6161616161616161>,
  _IO_read_end = 0x6161616161616161 <error: Cannot access memory at address 0x6161616161616161>,
  _IO_read_base = 0x6161616161616161 <error: Cannot access memory at address 0x6161616161616161>,
  _IO_write_base = 0x6161616161616161 <error: Cannot access memory at address 0x6161616161616161>,
  _IO_write_ptr = 0x6161616161616161 <error: Cannot access memory at address 0x6161616161616161>,
  _IO_write_end = 0x6161616161616161 <error: Cannot access memory at address 0x6161616161616161>,
  _IO_buf_base = 0x6161616161616161 <error: Cannot access memory at address 0x6161616161616161>,
  _IO_buf_end = 0x6161616161616161 <error: Cannot access memory at address 0x6161616161616161>,
  _IO_save_base = 0x6161616161616161 <error: Cannot access memory at address 0x6161616161616161>,
  _IO_backup_base = 0x6161616161616161 <error: Cannot access memory at address 0x6161616161616161>,
  _IO_save_end = 0x6161616161616161 <error: Cannot access memory at address 0x6161616161616161>,
  _markers = 0x6161616161616161,
  _chain = 0x6161616161616161,
  _fileno = 0x61616161,
  _flags2 = 0x61616161,
  _old_offset = 0x6161616161616161,
  _cur_column = 0x6161,
  _vtable_offset = 0x61,
  _shortbuf = "\n",
  _lock = 0x7ffff7dd3780 <_IO_stdfile_1_lock>,
  _offset = 0xffffffffffffffff,
  __pad1 = 0x0,
  __pad2 = 0x7ffff7dd17a0 <_IO_wide_data_1>,
  __pad3 = 0x0,
  __pad4 = 0x0,
  __pad5 = 0x0,
  _mode = 0xffffffff,
  _unused2 = '\000' <repeats 19 times>
}
gef➤  
```

成功修改 stdout 结构体

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char * argv[])
{
   FILE *fp;
   char *msg = "hello_stdout";

   char *buf = malloc(100);

   fp = stdout;

   // ■■ flag ■■ check
   fp->_flags &= ~8;
   fp->_flags |= 0x800;

   // _IO_write_base write■■■■■■■■ _IO_write_ptr  write■■■■■■■
   fp->_IO_write_base = msg;
   fp->_IO_write_ptr = msg + 12;

   //■■■■
   fp->_IO_read_end = fp->_IO_write_base;

   // write ■■■ ■■■■■■ 1 --> ■■■■
   fp->_fileno = 1;
   puts("<----->this is append on msg ");

   return 0;
}
```



```
haclh@ubuntu:~/workplace/file_exploit$ ./test_stdout
hello_stdout<----->this is append on msg
haclh@ubuntu:~/workplace/file_exploit$ 
```

成功读到了， msg 的内容。

参考：

https://www.slideshare.net/AngelBoy1/play-with-file-structure-yet-another-binary-exploit-technique

点击收藏 | 0 关注 | 2

上一篇：域渗透技巧讨论--远程登录 下一篇：CVE-2017-15944 POC

1. 0 条回复
   - 动动手指，沙发就是你的了！

登录 后跟帖

先知社区

现在登录

热门节点

技术文章

社区小黑板

目录

RSS 关于社区 友情链接 社区小黑板