题目：7. ELF文件注入

摘要:

在Linux中下修改一个现有的elf可执行程序（显然没有源代码！譬如vi或其他任何自己编写的可执行程序）。让该程序运行后先执行一个特别的附件功能（附加功能是：创建或打开一个指定文件写入helloworld的字符串）后再继续运行该程序，否则程序结束。注意：附加功能嵌入到了原来的程序中，不是一个独立程序！要求了解elf文件格式，另外建议使用汇编编程。

关键词:

ELF 文件、文件注入

团队信息:

组员:

姓名	学号	班级	职务
赵楠	U201617007	软件1601	组长
胡兴球		软件1604	组员

班级：软件1602

注：本组原组长为软件1602班的杜恩博，但由于途中杜恩博由于身体原因暂时休学，故不在参与大作业，故此组长有赵楠代理，但由于原申报班级为软件1602班，这里不做修改

完成日期：2018.05.06

备注:

- 电子版和源程序发到老师指定的邮箱 OSCourse@163.com。
- 邮件附件和主题注明 4 个要素: “学号-姓名-题号-题目不超过 15 字的简短文字描述”。
- 发送邮件时,不得采用 QQ 超大附件或云附件或网盘存储方式发送!!
- 源文件注意要拷贝完整,不要遗漏不在工程主目录下的第三方库或头文件。老师若编

译不了工程,算题目没有完成。

作业概述

在Linux中下修改一个现有的elf可执行程序（显然没有源代码！譬如vi或其他任何自己编写的可执行程序）。让该程序运行后先执行一个特别的附件功能（附加功能是：创建或打开一个指定文件写入helloworld的字符串）后再继续运行该程序，否则程序结束。注意：附加功能嵌入到了原来的程序中，不是一个独立程序！要求了解elf文件格式，另外建议使用汇编编程。

1. 功能需求

概述：

在一个已有的 elf 可执行程序中嵌入一段代码，从而让该程序运行之后先指向一个自己写的特殊附加功能，之后再指向原二进制可执行二进制文件的功能。

具体：

- 修改一个 ELF 可执行程序

  测试用elf可执行程序功能：

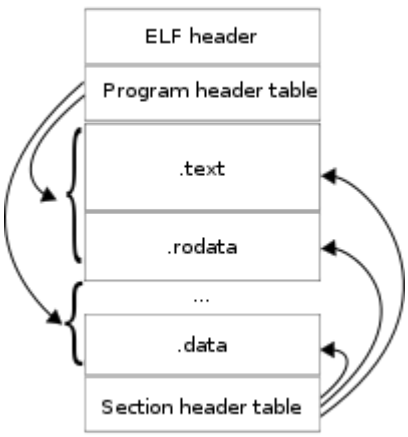
  在屏幕上打印 `This is the program, whick will be injected.\n` 。
- 让该程序运行后先执行一个特别的附件功能

  这里程序中的附加功能为：

  创建或者打开当前目录下的 helloworld 文件，并向其中写入 helloworld 字符串
- 附加功能执行之后执行ELF文件原来的功能
- 不能干扰ELF文件原来的功能
- 环境要求：Linux
- 要求了解elf文件格式
- 建议使用汇编编程

2. 设计思路

ELF 文件结构：



注：ELF文件格式详见文件：**ELF文件详解**

1. 存储相关原始数据，如原文件入口地址
2. (也可以最后修改)修正 ELF 头部 中的 e_shoff ，增加 PAGESIZE 大小（操作系统页式系统，一页默认4k）
3. 修正 第一个程序头部表，第一个头部特殊对待，因为要插入自己注入的程序，所以要把第一个头部扩容，把 p_filesz 和 p_memsz 增加 PAGESIZE 大小或者 注入程序的大小
4. 修正 ELF 头部 中的 e_entry ，指向 p_vaddr + p_filesz
5. 修正程序头部表偏移地址p_offset ，增加 PAGESIZE 大小
6. 修正节区 sh_offset ，增加 PAGESIZE 大小
7. 修正注入程序机器码，如：修正数据段存储地址（从新的elfh.e_entry开始计算程序首地址），最后要加上跳转指令，即跳转到原来的e_entry（刚开始记录的）
8. 首先存储原来目标节区头到末尾的数据，然后插入修正后的注入程序的机器码，之后通过插入0的方式让插入区块扩充到 PAGESIZE （4k）
9. 再把存储的数据接在后面插入

3. 开发环境下载、安装和配置

开发环境：

Linux内核：4.16.0-999-generic

Linux发行版本：Ubuntu16.04 LTS 64位

gcc 版本：5.4.0 20160609

汇编格式：AT&T format

注：汇编代码为64位环境代码，不是32位环境代码，不保证32位环境下支持

安装：

均为Linux自带工具，无需额外安装，只需要有Linux运行环境即可

工程文件目录及简介：

elf_inject

|- helloworld.o helloworld.s 编译之后的二进制文件

|- helloworld.o.txt 经过objdump命令反汇编之后带解释的机器码和汇编代码混合，编译查看机器码对应的功能

|- helloworld.s 嵌入的功能实现汇编代码

|- main.c elf文件嵌入的功能实现代码

|- main main.c 编译之后的可执行程序

|- test.c 用于生成测试elf文件

|- test test.c 编译之后的可执行程序

配置：

1. 编译汇编文件

```
gcc -c helloworld.s # 默认生成 helloworld.o
```

2. 反编译

```
objdump -s -d helloworld.o > helloworld.o.txt
```

> 注：此处helloworld.o.txt为机器码和汇编代码夹杂，用汇编作为旁注便于学习机器码对应的汇编代码

3. 编译 .c 文件

```
gcc main.c -o main
gcc test.c -o test
```

> 注：具体使用以及测试请见 **5. 运行和测试过程**

4. 程序的难点或核心技术分析

程序难点：

1. 深入理解 ELF 文件格式
2. 学习并实践 64 位环境下AT&T format 汇编文件操作
3. 利用 C 程序定位 ELF 文件中各表段并精确读取
4. 修改插入程序的机器码的地址并在最后实现跳转
5. 修改程序头部表和节区头部表
6. 正确插入注入的机器码

汇编代码分析：

文件：helloworld.s

整体思路：

首先将rax, rbx, rcx, rdx等寄存器pushq入栈，进行现场保存，最后在popq出栈，防止对外界环境进行污染。汇编代码实现的功能比较简单，只是打开或者创建helloworld文件，之后写入helloworld字符串即可。这里直接通过系统调用号来实现，比如创建文件使用系统调用号 sys_create，写入文件使用系统调用号 sys_write。

核心代码：

写入文件：

```
movq $4, %rax    # 系统调用号(sys_write)  写入文件
movq $content, %rcx  # content 为写入内容  (地址)
movq $10, %rdx   # 10为字符串的大小
```

> 注：rax, rbx, movq, pushq, popq等均为64位环境的汇编代码，相应的32位的为eax, ebx等

嵌入文件功能实现C代码分析：

文件：main.c (注释很多，可直接阅读)

总体过程：

1. 记录原始数据，并执行相应前提判断

2. 修改程序头部表
3. 修改节区头部表
4. 计算注入机器码的中相应的地址，并修改机器码
5. 插入机器码

具体分析：

- 读取elf文件头部（其他类似）

```
read(old_file, &elf_ehdr, sizeof(elf_ehdr));
```

- 修改程序头部表

  由于要插入一段自己的功能机器码在程序中，所以这里需要程序头部表项的偏移地址增加一页（4k），同时记录新的程序入口虚拟地址。特别的由于我的注入程序是插在最开始的节区，所以这里第一个程序头部表的偏移地址无需修改，但是需要修改其 p_filesz 和 p_memsz ，由于增加了自己的注入代码（4k对齐）

  代码：

```
// 增加 p_offset 一页大小 4k
elf_phdr.p_offset += PAGESIZE;

// 寻找并更新 程序头部
lseek(old_file, elf_ehdr.e_phoff + i * elf_ehdr.e_phentsize, SEEK_SET);
write(old_file, &elf_phdr, sizeof(elf_phdr));

// 增加 p_filesz 和 p_memsz 一页大小 4k
elf_phdr.p_filesz += PAGESIZE;
elf_phdr.p_memsz += PAGESIZE;
```

- 修改节区头部表

  类似修改程序头部表，这里只需将除第一个节区头部表外的表项中记录的偏移地址增加一页即可，特别的第一个节区表项的偏移地址无修改，但是需要修改sh_size 节区大小，增加一页4k（第一个节区插入了注入代码，4k对齐）

  代码：

```
// 第一个节区增加一页
elf_shdr.sh_size += PAGESIZE;

// 节区偏移地址增加一页
elf_shdr.sh_offset += PAGESIZE;
```

- 插入机器码：

  计算新的地址：

```
for (int i = 0; i < 4; i++)
{
    addr[i] = temp % 256;
    temp /= 256;
}
```

32位地址分别保存8位到addr[4]数组中，然后直接将机器码中的相应地址修改位数组中的值即可；

机器码只需插入功能执行部分的机器码，但是由于插入之后程序的数据段的地址修改，故需要计算新的数据段地址；

```
// 数据段的地址, 73 为数组中程序数据段的相对位置
int data_entry = elf_ehdr.e_entry + 73;
int data_addr[4];
cal_addr(data_entry, data_addr);
```

最后在程序功能执行完了之后执行跳转指令，跳转回 elf 可执行文件原来的入口点。

真正插入：

首先保存从目标节区头到程序末尾的数据，之后插入自己的机器码之后，再将机器码扩充到一页(4k)(不足4k用0补全)，再将之前保存的数据接着注入的机器码后面插入即可。这样整个代码注入就完成了。

相关代码：

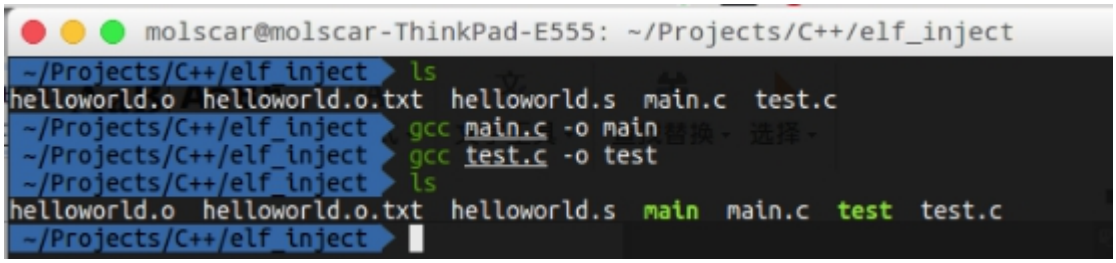
```
// 存储原程序从节区末尾到目标节区头的数据
lseek(old_file, old_phsize, SEEK_SET);
read(old_file, data, file_stat.st_size - old_phsize);

// 插入注入代码到原 elf 文件中
lseek(old_file, old_phsize, SEEK_SET);
write(old_file, inject_code, inject_size);

// 扩充到一页
char tmp[PAGESIZE] = {0};
memset(tmp, PAGESIZE - inject_size, 0);
write(old_file, tmp, PAGESIZE - inject_size);

 // 再将原始的数据接在注入代码后面插入
write(old_file, data, file_stat.st_size - old_phsize);
```

5. 运行和测试过程

1. 编译 main.c 和 test.c

```
gcc main.c -o main  # (elf文件注入功能实现)
gcc test.c -o test   # (测试用elf文件)
```

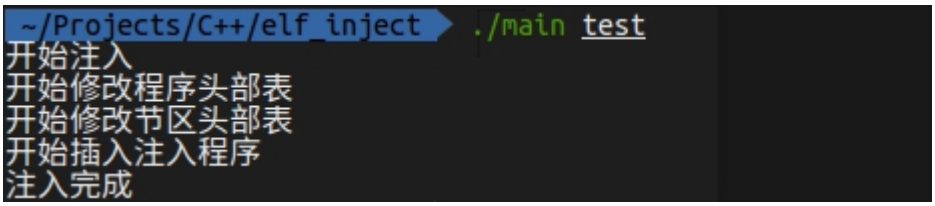
下图可以看到，编译无warning，无error



2. Elf文件注入

```
./main test
```

可以看到屏幕打印的注释为嵌入代码的过程注释，最后显示注入完成，说明程序一直执行到最后嵌入完成都没有如何问题。

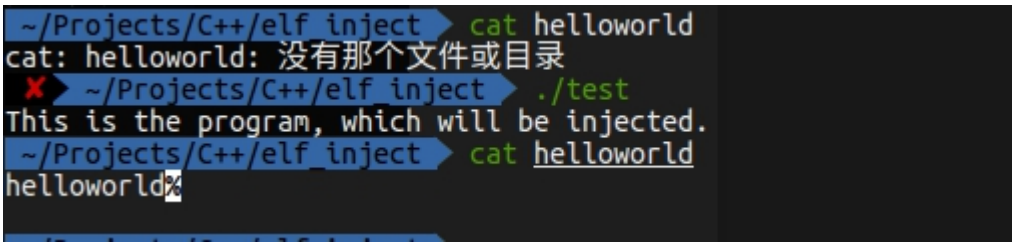


3. 成果查看

```
./test
cat helloworld
```

下图中，可以看到

- 在执行被修改过的elf二进制文件之前是没有helloworld文件的，执行test 可执行程序之后，就自动生成了helloworld文件，内容为helloworld，说明代码嵌入的功能成功实现。
- 执行test 可执行程序之后，屏幕上打印了 “This is the program, which will be injected.”，说明成功执行了 test执行文件的原来功能，即保存了原来程序的功能。



6. 编程参考网址

IBM Linux汇编语言开发指南

汇编学习-使用文件

AT&T汇编helloworld实例

汇编语言入门教程

王爽的 汇编语言

linux 系统调用号表

elf 文件格式解析

elf增加一个可执行段以注入代码的一些思考