ICS-LAB5

Link/链接

哈尔滨工业大学
计算机科学与技术学院

2021年5月21日, Friday

一、实验基本信息

- **实验类型：综合型实验**
- **实验目的**
  - 理解链接的作用与工作步骤
  - 掌握ELF结构、符号解析与重定位的工作过程
  - 熟练使用Linux工具完成ELF分析与修改
- **实验指导教师**
  - 任课教师：郑贵滨
  - 实验室教师：
- **人数与分组**
  - 一人一组

- **实验学时：3**
- **实验学分：3，本次实验按100分计算，折合成总成绩的3分。**
- **实验地点：G712、G709**
- **实验环境与工具：**
  - X64 CPU；2GHz；2G RAM；256GHD Disk 以上
  - Windows7 64位以上；VirtualBox/Vmware 11以上；Ubuntu 16.04 LTS 64位/优麒麟 64位；
  - Visual Studio 2010 64位以上；GDB/OBJDUMP；DDD/EDB等
- **学生实验准备：禁止准备不合格的学生做实验**
  - 个人笔记本电脑
  - 实验环境与工具所列明软件
  - 参考手册: Linux环境下的命令；GCC手册；GDB手册
  - http://docs.huihoo.com/c/linux-c-programming/ C汇编Linux手册
  - http://csapp.cs.cmu.edu/3e/labs.html CMU的实验参考
  - http://www.linuxidc.com/ http://cn.ubuntu.com/ http://forum.ubuntu.org.cn/

二、实验要求

- **学生应穿鞋套进入实验室**
- **进入实验室后在签到簿中签字**
- **实验安全与注意事项**
  - 禁止使用笔记本电脑以外的设备
  - 学行生不得自行开关空调、投影仪
  - 学生不得自打开窗户
  - 不得使用实验室内的其他实验箱、示波器、导线、工具、遥控器等
  - 认真阅读消防安全撤离路线
  - 突发事件处理：第一时间告知教师，同时关闭电源插排开关。
- **遵守学生实验守则，爱护实验设备，遵守操作规程，精心操作，注意安全，严禁乱拆乱动。**
- **实验结束后要及时关掉电源，对所用实验设备进行整理，设备摆放和状态恢复到原始状态。**
- **桌面整洁、椅子归位，经实验指导教师允许后方可离开**

三、实验预习

- 上实验课前，必须认真预习实验指导书（PPT或PDF）
- 了解实验的目的、实验环境与软硬件工具、实验操作步骤，复习与实验有关的理论知识。
- 请按顺序写出ELF格式的可执行目标文件的各类信息。
- 请按照内存地址从低到高的顺序，写出Linux下X64内存映像。
- 请运行“LinkAddress -u 学号 姓名” 按地址顺序写出各符号的地址、空间。并按照Linux下X64内存映像结构，标出其所属各区。
  - gcc -m64 -o LinkAddress linkaddress.c
- 请按顺序写出LinkAddress从开始执行到main前/后执行的子程序的名字。(gcc与objdump/GDB/EDB)

四、实验内容与步骤

- **1.环境建立**
  - Windows下Visual Studio 2010 64位
  - Windows下 OllyDbg（Windows下的破解神器OD）
  - Ubuntu下安装EDB（OD的Linux版---有源程序！）
  - Ubuntu下GDB调试环境、OBjDUMP等
- **2.获得实验包**
  - 从实验教师处获得下 linklab.tar
  - 也可以从课程QQ群下载，也可以从其他同学处获取。
  - 每人的包都不相同，一定要注意，
  - CMU无此实验，HIT增加
- **3.Ubuntu下ELF文件分析: readelf 看帮助**
  - readelf -*h*　读取分析ELF文件头
  - readelf -*s*　符号表（分析符号与动态符号)　　-*x* 看字节 -*p*看字符串
  - readelf -*a*　看所有信息　　可练习！ readelf -*r* 等

readelf <option(s)> elf-file(s)

-*a* –all 等同于同时使用：-h -l -S -s -r -d -V -A -I

-*h* --file-header 显示ELF文件头

-*l* --program-headers 显示程序头

-*S* --section-headers 显示节头

-*t* --section-details 显示节详细信息

-*s* –syms 显示符号表（symbol table）

-*r* –relocs 显示重定位信息

-*d* –dynamic 显示动态节（dynamic section）

-*x* --hex-dump=<number|name>

以字节形式显示输出<number|name>指定节的内容

-*p* --string-dump=<number|name>

以字符串形式显示输出<number|name>指定节的内容

-*R* --relocated-dump=<number|name>

以重定位后的字节形式显示输出<number|name>指定节内容

- **4.运行LinkAddr程序，看输出结果**
  - 排序一下输出的各个符号。
  - 查看内存：argv 与 env 典型的char** 或char*[]
- **5.GDB/EDB打开linkAddr**
  - 调试程序，查看从最开始运行、到main函数、直至退出前，所有运行过的函数。按顺序列出。
- **6.查看变量与函数的地址**

  以下函数在对应的调用指令中的地址是何时确定确定的、谁确定的？列出各符号的地址、内容
  - 函数useless、showpointer、main；
  - 函数__printf_chk、puts；malloc、free；exit、printf
  - 符号__environ、global、argc、argv的地址与值怎么确定的

7. linkbomb实验包分析

- 实验数据包： linklab.tar
- 解压命令 $ tar vxf linklab.tar
- 数据包中包含下面3个文件：
  - main.o： 主程序的可重定位目标模块-实验中无需修改
  - phase1.o …phasen.o：各阶段实验所针对的二进制可重定位目标模块，需在相应实验阶段中予以修改或补充
- 实验目标程序运行
  - 在实验中的每一阶段n，按照阶段的目标要求修改相应可重定位二进制目标模块phase*n*.o后，使用如下命令生成可执行程序linkbomb：
  - $ gcc -o linkbomb[n] main.o phase[n].o [-no-pie]
  - ./linkbomb 运行后会输出信息，如你的 学号

8. LinkBomb程序框架

```
// main.c
#include <stdio.h>
#include "config.h"
void (*phase)();
int main( int argc, const char* argv[] ) {
    if ( phase )
        (*phase)();
    else
        printf("To run lab, please link the relevant
object module with the main module.\n");
    return 0;
}
```

```
// phase[n].c

void do_phase() {

    … // 该阶段具体工作
}

void (*phase)() = do_phase;
```

注释： 各阶段phase[n].c中的全局函数指针变量phase是经初始化的“强”符号，在将phase[n].o模块与main.o链接后，前者中的phase变量定义将取代后者中的同名“弱”符号（变量），因此相应阶段中完成具体功能的do_phase函数将被调用执行。

9.实验任务

- 每个实验阶段考察ELF文件组成与程序链接过程的不同方面知识
  - 阶段1：全局变量⇔数据节
  - 阶段2：指令⇔代码节
  - 阶段3：符号解析
  - 阶段4：switch语句与重定位
  - 阶段5：重定位

实验阶段1

- **实验步骤：**

1）使用readelf和objdump工具，首先确定printf（具体为puts）输出函数的第2个调用参数对应的字符串地址（在.data节中）

注意：printf(“%s\n”, s)；会编译成 puts(s)函数调用，注意s为字符串，应该在数据段　可知输出字符串起始地址在.data节中偏移量为 xx 的位置

2）使用readelf或objdump工具，查看.data节中的字符串内容并与未修改的phase1.o链接后程序输出的字符串比较，确定该字符串为修改的目标

3）使用hexedit或自己写程序，将该字符串前若干字符替换为目标学号中的字符（其后应有一个0x00字节，表示字符串结束）

```
$ gcc -m32 -o linkbomb1 main.o phase1.o
$ ./linkbomb1
你的学号
```

实验阶段2

- phase2.c程序框架

```
static void OUTPUT_FUNC_NAME( const char *id ) {
// 该函数名对每名学生均不同
    if( strcmp(id,MYID) != 0 ) return;
       printf("%s\n", id);
}
void do_phase()  {// 在代码节中预留存储位置供插入必要指令

asm("nop\n\tnop\n\tnop\n\tnop\n\tnop\n\tnop\n\tnop\n\tnop\n\tnop\n\t");

asm("nop\n\tnop\n\tnop\n\tnop\n\tnop\n\tnop\n\tnop\n\tnop\n\tnop\n\t");
}
```

实验阶段2

- **实验内容：**

修改二进制可重定位目标文件“phase2.o”的代码节内容，使其与main.o链接后能够运行输出（且仅输出）自己的学号：

```
$ gcc -m32 -o linkbomb2 main.o phase2.o
$ ./linkbomb2
学号
```

- **实验提示：**
  - ✓ 检查反汇编代码，定位模块中的各组成函数并推断其功能作用
  - ✓ 修改入口函数do_phase()中的机器指令（用自己指令替换函数体中的nop指令）以获得期望的输出（学号的ASCII编码）

实验阶段2

- **实验提示：**

- ✓ 使用objdump工具，定位phase2.o代码节中printf函数调用点（汇编层面对应puts)，猜测字符串输出函数(rRlVNhXm函数)。
- ✓ 使用objdump工具，分析do_phase函数的反汇编指令
- ✓ *编写类似phase2.c的程序，并在do_phase中用赋初值的局部字符串数组名为参数调用输出函数，编译成目标文件，查看其反汇编，了解指令特点*
- ✓ 根据上一步的信息，修改phase2.o: 构造调用输出函数（通过相对PC的偏移量）的机器指令，并替换do_phase函数中预留的nop指令

注：输出函数为static类型，可通过偏移量直接调用跳转（无需重定位）

实验阶段3

- phase3.c程序框架

```
char PHASE3_CODEBOOK[256];

void do_phase(){
    const char char cookie[] = PHASE3_COOKIE;
    for( int i=0; i<sizeof(cookie)-1; i++ )
        printf( "%c", PHASE3_CODEBOOK[ (unsigned char)(cookie[i]) ] );
    printf( "\n" );
}
```

实验阶段3

- **实验内容：**

创建生成一个名为“phase3_patch.o”的二进制可重定位目标文件，使其与main.o、phase3.o链接后能够运行和输出（且仅输出）自己的学号：

```
$ gcc –m32 -o linkbomb3 main.o phase3.o phase3_patch.o
$ ./linkbomb3
学号
```

- **实验提示：**

  - ✓ 模块入口函数do_phase()依次遍历一个COOKIE字符串（由一组互不相同的英文字母组成，且总长度与学号字符串相同）中的每一字符，并通过一个映射数组将该字符的不同可能ASCII编码取值映射为输出字符。
  - ✓ 了解并利用符号解析规则。

17

实验阶段3

- **实验提示：**

✓ 1)分析do_phase函数反汇编指令，获知COOKIE字符串（保存于栈帧中的局部字符数组中）的组成内容和起始地址

✓ 2) 定位循环结构　根据cookie中字符的使用，定位映射数组的引用位置结合重定位记录，确定映射数组的变量名

✓ 3)通过符号表，发现该数组为一未初始化变量（类型为COM，长度为256字节）

✓ 4）要改变程序输出（为学号），必须改变该映射数组的内容，因此，可利用强弱全局符号的解析规则，在patch模块中定义同名且按输出要求正确初始化映射关系的数组变量——从而在链接时替换对原数组的引用

实验阶段4

- phase4.c程序框架

```
void do_phase(){
    const char cookie[] = PHASE4_COOKIE;
    char c;
    for ( int i = 0; i < sizeof(cookie)-1; i++ ) {
        c = cookie[i];
        switch (c) {
            // 每个学生的映射关系和case顺序建议不一样
            case 'A': { c = 48; break; }
            case 'B': { c = 121; break; }
              …
            case 'Z': { c = 93; break; }
        }
        printf("%c", c);
    }
}
```

实验阶段4

- **实验内容**

修改二进制可重定位目标文件“phase4.o”中相应节中的数据内容（注意不允许修改.text节的内容），使其与main.o链接后能够运行输出（且仅输出）自己的学号：

```
$ gcc –m32 -o linkbomb4 main.o phase4.o
$ ./linkbomb4
学号
```

实验阶段4

- **实验提示**
  - 模块入口函数do_phase()依次遍历一个COOKIE字符串（由一组互不相同的大写英文字母组成，且总长度与学号字符串相同）中的每一字符，并使用一个switch语句将该字符的不同可能ASCII编码取值映射为输出字符。
  - 了解掌握switch语句的机器表示的各个组成部分及其特定重定位数据组成。

实验阶段4

- **实验步骤**

1）通过分析do_phase函数的反汇编程序获知COOKIE字符串（保存于栈帧中的局部字符数组中）的组成内容

2）确定switch跳转表在.rodata节中的偏移量

3）定位COOKIE中每一字符’c’在switch跳转表中的对应表项（索引为’c’-0x41），将其值设为输出目标学号中对应字符的case首指令的偏移量

实验阶段5：不建议

- **实验内容：**

修改二进制可重定位目标文件“phase5.o”的重定位节中的数据内容（不允许修改.text节的内容），补充完成其中被清零的一些重定位记录（分别对应于本模块中需要重定位的符号引用），使其与main.o链接后能够正确输出（且仅输出）自己学号的编码结果：

```
$ gcc –m32 -o linkbomb5 main.o phase5.o
$ ./linkbomb5
学号编码后字符串
```

- **实验提示：**

如果实验中对缺失重定位信息的恢复不完整或不正确的话，链接生成linkbomb程序时可能不报错，但运行程序可能得到以下结果之一：

- 出现“Segmentation fault”出错信息——原因？ “如果未对相关引用进行必要的重定位会发生什么？”
- 输出“Welcome to this small lab of linking. To begin lab, please link the relevant object module(s) with the main module. ”——提示模块未链接。可能原因：虽然按上述步骤在生成linkbomb程序时实际已链接进phase4.5模块，但某个重要的重定位记录未正确设置。
- 输出不正确的编码结果

实验阶段5

- phase5.c程序框架

```
const int TRAN_ARRAY[] = {… …};
const char FDICT[] = FDICTDAT;
char BUF[] = MYID;
char CODE = PHASE5_COOKIE;

int transform_code( int code, int mode ) {
   switch( TRAN_ARRAY [mode] & 0x00000007 )
      case 0:
         code = code & (~ TRAN_ARRAY[mode]);
         break;
      case 1:
         code = code ^ TRAN_ARRAY[mode];
         break;
      … …
   }
   return code;
}
```

```
void generate_code( int cookie ) {
   int i;
   CODE = cookie;
   for( i=0; i<sizeof(TRAN_ARRAY)/sizeof(int);
i++ )
      CODE = transform_code( CODE, i );
}

int encode( char* str ) {
   int i, n = strlen(str);
   for( i=0; i<n; i++ ) {
      str[i] = (FDICT[str[i]] ^ CODE) & 0x7F;
      if( str[i]<0x20 || str[i]>0x7E ) str[i] = ' ';
   }
   return n;
}

void do_phase() {
   generate_code(PHASE5_COOKIE);
   encode(BUF);
   printf("%s\n", BUF);
}
```

- 上列绿色标出（以及如switch的跳转表等）的符号引用的对应重定位记录中随机选择若干个被置为全零。
- 涉及的重定位记录可能位于.text, .rodata等不同重定位节中

实验阶段5

- **实验步骤**

1）对照phase5.o的反汇编程序及已有重定位记录，定位每一空重定位记录可能对应的符号引用。

2）对每一待处理的符号引用，按照下列重定位记录结构，构造其二进制表示（8字节块）。

```
1   typedef struct {
2       int offset;      /* Offset of the reference to relocate */
3       int symbol:24,   /* Symbol the reference should point to */
4           type:8;      /* Relocation type */
5   } Elf32_Rel;
```

3）使用hexedit或编程将生成的重定位记录写入到相应被清空的记录位置中。

10.实验结果提交

- 将修改完成的各阶段模块（phase1.o, phase2.o, phase3_patch.o, phase4.o, phase5.o）和未改动的main.o、phase3.o模块一起用tar工具打包：

  tar cvf <学号>.tar main.o phase1.o phase2.o phase3.o phase3_patch.o phase4.o phase5.o

  - 注意：TAR文件中一定不要包含任何目录结构

- 将结果tar文件重命名为“学号.tar”后提交

五、实验报告格式

- 按照实验报告模板所要求的格式与内容提交。
- 实验后 1周内 提交至课代表并打包给授课教师。
- 本次实验成绩按100分计
  - 按时上课，签到5分
  - 按时下课，不早退5分
  - 课堂表现：10分，不按操作规程、非法活动扣分。
  - 实验报告：80分。具体参见实验报告各环节的分值
- 学生提交1个压缩包即可，课代表提交1个包
- 在实验报告中，对每一任务，用文字详细描述分析与攻击过程，栈帧内容要截图标注说明。
- 注意：及时记录每一步的地址、变量、函数、参数、数据结构、算法等等。以方便实验报告的撰写。