信息学院

实验报告书

课程名称： 嵌入式操作系统

实验项目： 系统调用劫持

实验地点：  信息学院

专业班级： 20计科3班

学生姓名： 胡博文 学号： 2052312

指导教师： 冯国富

2023年06月 07日

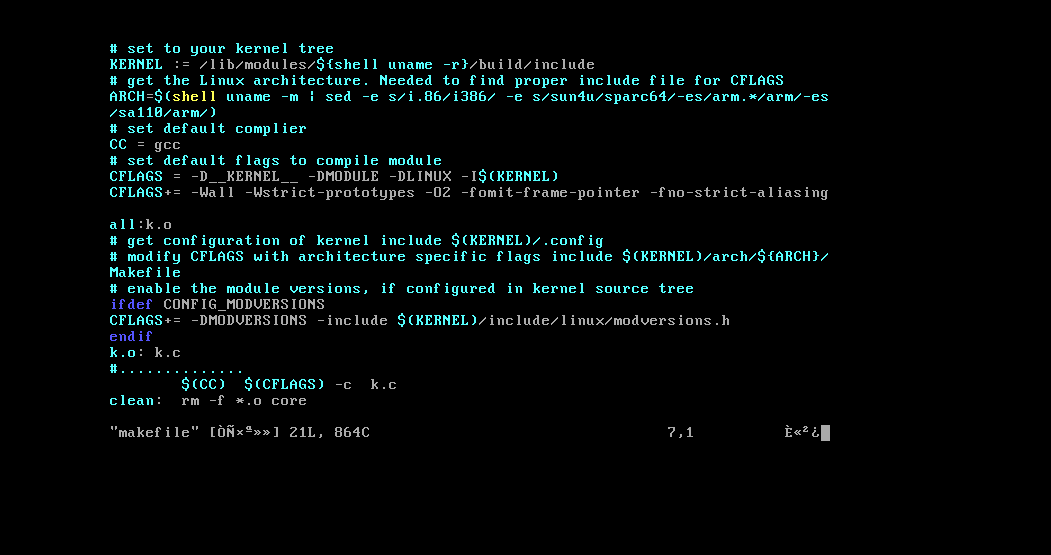
**实验二 系统调用劫持**

**【目的与要求】**

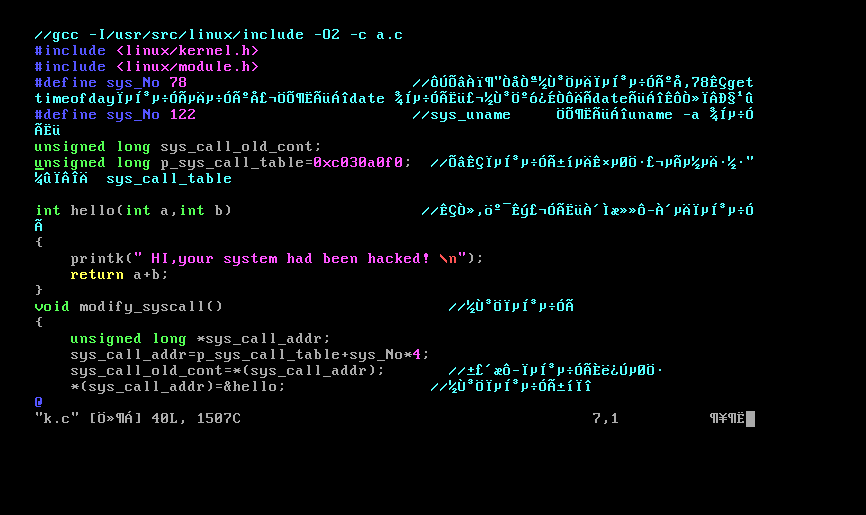
1. 目的
2. 熟悉linux基础命令
3. 对系统调用有初步了了解
4. 学会对系统调用进行基本的劫持操作
5. 要求
6. 用vi命令编写makefile、k.c、user.c文件
7. 通过gcc编译并运行user.c文件
8. 在root目录下使用make命令
9. 实现对gettimeofday系统调用的劫持

**【实验内容】**

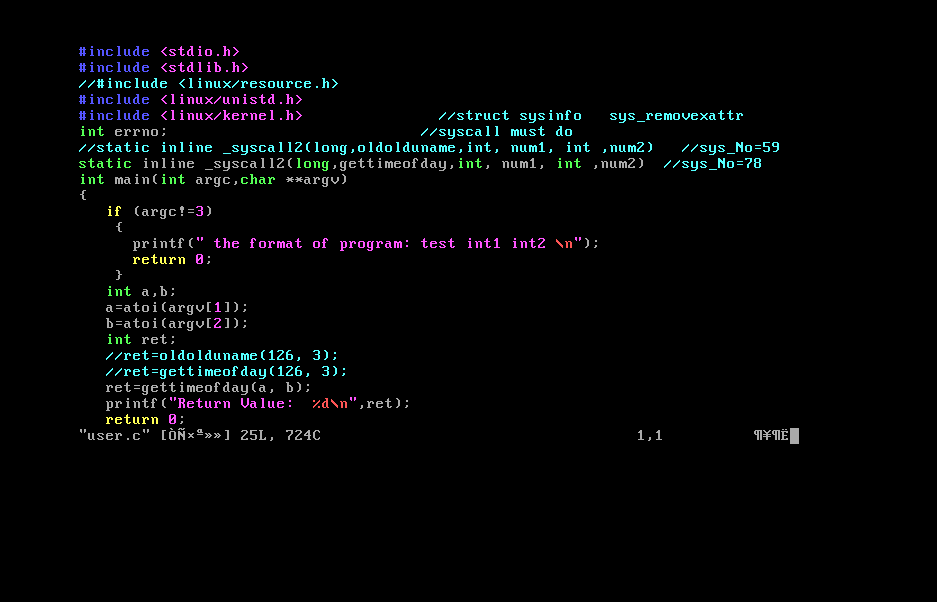
1. 用vi命令编写makefile



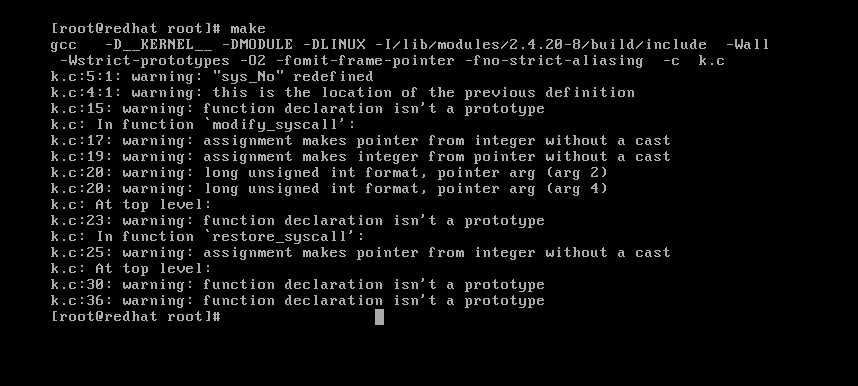
1. 使用vi命令编写k.c文件



1. 使用vi命令编写user.c



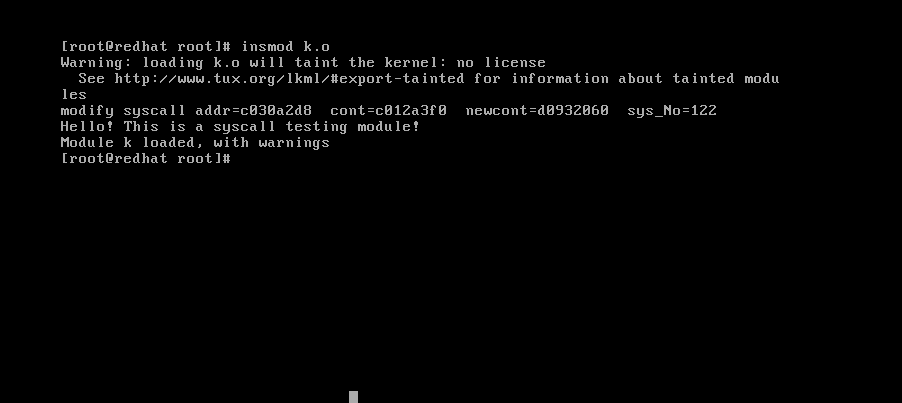
1. 执行make命令



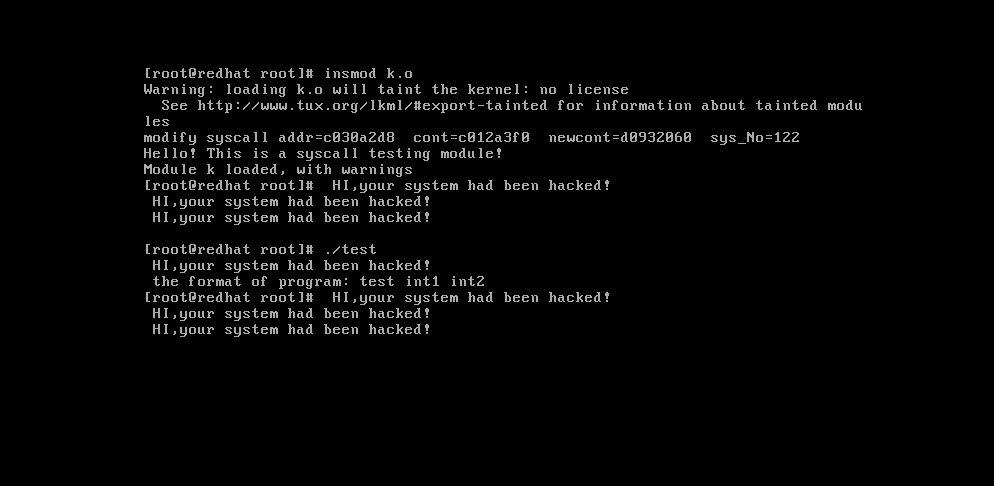
1. 使用gcc编译user.c



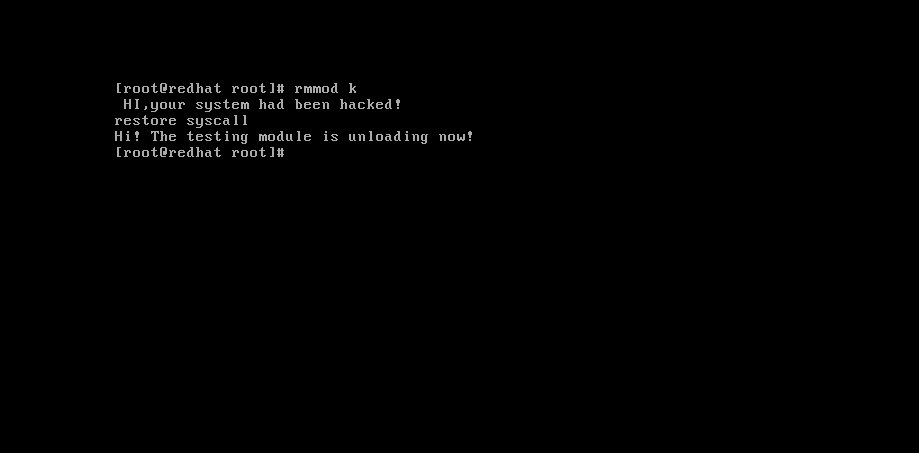
1. 使用insmod将k.o 嵌入内核



1. 调用test



1. 使用rmmod卸载模块



【**实验代码**】

**k.c**

//gcc -I/usr/src/linux/include -O2 -c a.c

#include <linux/kernel.h>

#include <linux/module.h>

#define sys\_No 78 //在这里定义要劫持的系统调用号,78是gettimeofday系统调用的调用号，终端命令date 就调用它，劫持后可以拿date命令试一下效果

#define sys\_No 122 //sys\_uname 终端命令uname -a 就调用它

unsigned long sys\_call\_old\_cont;

unsigned long p\_sys\_call\_table=0xc030a0f0; //这是系统调用表的首地址，得到的方法见下文 sys\_call\_table

int hello(int a,int b) //是一个函数，用它来替换原来的系统调用

{

printk(" HI,your system had been hacked! \n");

return a+b;

}

void modify\_syscall() //劫持系统调用

{

unsigned long \*sys\_call\_addr;

sys\_call\_addr=p\_sys\_call\_table+sys\_No\*4;

sys\_call\_old\_cont=\*(sys\_call\_addr); //保存原系统调用入口地址

\*(sys\_call\_addr)=&hello; //劫持系统调用表项

printk("modify syscall addr=%lx cont=%lx newcont=%lx sys\_No=%d\n",sys\_call\_addr,sys\_call\_old\_cont,&hello,sys\_No);

}

void restore\_syscall() ////恢复系统调用

{

unsigned long \*sys\_call\_addr;

sys\_call\_addr=p\_sys\_call\_table+sys\_No\*4;

\*(sys\_call\_addr)=sys\_call\_old\_cont; ////恢复系统调用表项

printk("restore syscall \n");

}

int init\_module()

{

modify\_syscall();

printk("Hello! This is a syscall testing module! \n");

return 0;

}

void cleanup\_module()

{

restore\_syscall();

printk("Hi! The testing module is unloading now! \n");

}

**user.c**

#include <stdio.h>

#include <stdlib.h>

//#include <linux/resource.h>

#include <linux/unistd.h>

#include <linux/kernel.h> //struct sysinfo sys\_removexattr

int errno; //syscall must do

//static inline \_syscall2(long,oldolduname,int, num1, int ,num2) //sys\_No=59

static inline \_syscall2(long,gettimeofday,int, num1, int ,num2) //sys\_No=78

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

{

if (argc!=3)

{

printf(" the format of program: test int1 int2 \n");

return 0;

}

int a,b;

a=atoi(argv[1]);

b=atoi(argv[2]);

int ret;

//ret=oldolduname(126, 3);

//ret=gettimeofday(126, 3);

ret=gettimeofday(a, b);

printf("Return Value: %d\n",ret);

return 0;

}

**Makefile**

# set to your kernel tree

KERNEL := /lib/modules/${shell uname -r}/build/include

# get the Linux architecture. Needed to find proper include file for CFLAGS

ARCH=$(shell uname -m | sed -e s/i.86/i386/ -e s/sun4u/sparc64/-es/arm.\*/arm/-es/sa110/arm/)

# set default complier

CC = gcc

# set default flags to compile module

CFLAGS = -D\_\_KERNEL\_\_ -DMODULE -DLINUX -I$(KERNEL)

CFLAGS+= -Wall -Wstrict-prototypes -O2 -fomit-frame-pointer -fno-strict-aliasing

all:k.o

# get configuration of kernel include $(KERNEL)/.config

# modify CFLAGS with architecture specific flags include $(KERNEL)/arch/${ARCH}/Makefile

# enable the module versions, if configured in kernel source tree

ifdef CONFIG\_MODVERSIONS

CFLAGS+= -DMODVERSIONS -include $(KERNEL)/include/linux/modversions.h

endif

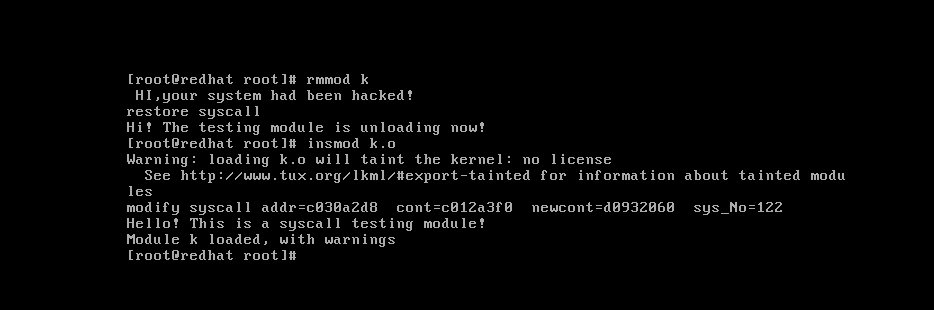
k.o: k.c

#..............

$(CC) $(CFLAGS) -c k.c

clean: rm -f \*.o core

【**运行结果】**



**【实验分析】:**

通过insmod和rmmod将自己的模块嵌入到内核中，并且找到系统调用表更改其某个系统调用的指向，让其指到自己所写的内核模块，从而完成对系统调用的挟持

**【心得体会】:**

学会了更广泛的linux指令的用法，也从实际操作中一定程度上理解了linux 内核态和用户态的概念，大致了解了权限管理作用和意义。