Operating Systems

Assignment #2

담당교수 : 김태석

강의 시간 : 수2

학부 : 컴퓨터정보공학부

학번 : 2017202088

이름 : 신해담

1. Introduction

- 특정 pid를 입력받아서 해당 pid에 대하여 파일에 관한 시스템콜을 추적하는 툴을 작성한다. ftrace 시스템콜을 hijack해서 새로운 ftrace 함수로 대체하고, open, read, write, lseek, close의 시스템콜을 hijack하여 새 함수로 대체해서 구현한다.

2. Analysis

- **ftracehooking.h**

  ftracehooking.c와 iotracehooking.c에서 사용할 헤더파일이다. 각종 헤더파일과 sys_call_table 파일의 권한을 변경하는 함수가 정의되어 있다.

```
// chmod syscall table
void make_rw(void *addr)
{
        unsigned int level;
        pte_t *pte = lookup_address((u64)addr, &level);

        if(pte->pte &~ _PAGE_RW)
                pte->pte |= _PAGE_RW;
}

// recover mod of syscall table
void make_ro(void *addr)
{
        unsigned int level;
        pte_t *pte = lookup_address((u64)addr, &level);

        pte->pte = pte->pte &~ _PAGE_RW;
}
```

- **ftracehooking.c**

  sys_ftrace를 hijack해서 새로운 ftrace 함수로 변경하는 모듈의 함수이다.

  End message를 가지는 end1, end2, end3을 전역변수로 가진다.

  또한 iotracehooking.c와 공유할 변수들도 선언되어 있다. savedPid는 ftrace 함수 호출 시 전달받은 pid값, fname은 open한 파일명, rw는 read, write한 bytes값, fcount는 각 파일 함수의 실행횟수를 값으로 저장한다.

```
#include "ftracehooking.h"

#define __NR_ftrace 336

void **syscall_table;

pid_t (*real_ftrace)(pid_t pid);

char end1[255] = {0, };
char end2[255] = {0, };
char end3[255] = {0, };

pid_t savedPid = 0;             // ftrace pid
char fname[255] = {0, };        // open file name
unsigned long rw[2] = {0, };    // read/write bytes
int fcount[5] = {0, };          // r/w/o/c/lseek count
// share with ioftracehooking.c
EXPORT_SYMBOL(savedPid);
EXPORT_SYMBOL(fname);
EXPORT_SYMBOL(rw);
EXPORT_SYMBOL(fcount);
```

sys_ftrace를 대체할 함수이다. 이미 저장된 pid가 존재하고, 전달받은 pid가 0이면 ftrace를 종료한다. task_struct에서 호출한 실행파일의 파일명을 얻어서 결과를 커널 메시지로 출력한다. 사용한 변수들도 0으로 초기화한다.

```
__SYSCALL_DEFINEx(1, _ftrace, pid_t, pid)
{
        // finish ftrace
        if(savedPid > 0 && pid == 0)
        {
                struct task_struct* task;
                task = get_current();

                // set result message
                sprintf(end1, "[2017202088] /%s file[%s] stats[x] read - %lu / w
ritten - %lu", task->comm, fname, rw[0], rw[1]);
                sprintf(end2, "open[%d] close[%d] read[%d] write[%d] lseek[%d]",
 fcount[0], fcount[1], fcount[2], fcount[3], fcount[4]);
                sprintf(end3, "OS Assignment2 ftrace [%d] End\n", savedPid);

                // print result
                printk(end1);
                printk(end2);
                printk(end3);

        // reset
        savedPid = 0;
        fname[0] = '\0';
        fcount[0] = 0;
        fcount[1] = 0;
        fcount[2] = 0;
        fcount[3] = 0;
        fcount[4] = 0;
        rw[0] = 0;
        rw[1] = 0;
}
```

전달받은 pid가 0보다 크면 ftrace를 시작한다. pid를 savedPid에 저장하고 ftrace start 메시지를 출력한다.

pid가 0보다 작으면 에러메시지를 출력하고 종료한다.

```
// start ftrace
else if(pid > 0)
{
        // save pid
        savedPid = pid;

        // print start message
        printk("OS Assignment2 ftrace [%d] Start\n", savedPid);

}
// negative pid
else
{
        printk("ftrace error: unexpected parameter [%d]\n", pid);
}
return pid;
```

- **iotracehooking.c**

  함수들의 원본함수를 사용하기 위해 함수포인터를 선언한다. ftracehooking.c와 공유하는 변수들을 선언한다.

```
#include "ftracehooking.h"

#define __NR_read 0
#define __NR_write 1
#define __NR_open 2
#define __NR_close 3
#define __NR_lseek 8

void **syscall_table;

asmlinkage ssize_t (*real_read)(unsigned int, char __user*, size_t) = NULL;
asmlinkage ssize_t (*real_write)(unsigned int, char __user*, size_t) = NULL;
asmlinkage long (*real_open)(const char __user*, int, umode_t) = NULL;
asmlinkage void (*real_close)(unsigned int) = NULL;
asmlinkage off_t (*real_lseek)(unsigned int, off_t, unsigned int) = NULL;

extern pid_t savedPid;
extern char fname[255];
extern unsigned long rw[2];
extern int fcount[5];
```

  ftrace_read, ftrace_write, ftrace_open, ftrace_close, ftrace_lseek로 시스템콜을 대체할 때, 새로 추가한 코드 뿐만 아니라 기존의 시스템콜의 작업을 수행하도록 시스템콜을 wrapping한다. read, write, open, close, lseek 각각이 수행될 때마다 함수를 호출한 pid와 ftrace중인 pid를 비교하여 같으면 fcount + 1 해준다.

  read와 write는 bytes를 계산해주어야 하기 때문에, rw에 +count 작업을 추가한다.

```
asmlinkage long ftrace_read(unsigned int fd, char __user* buf, size_t count)
{
        // if current pid is tracing process's pid
        struct task_struct* task;
        task = get_current();
        if(task->pid == savedPid)
        {
                // read count + 1
                fcount[2] += 1;

                // + read bytes
                rw[0] += count;
        }

        return real_read(fd, buf, count);
}
```

  open 시 열은 파일명을 얻기 위해 copy_from_user 작업을 추가해준다.

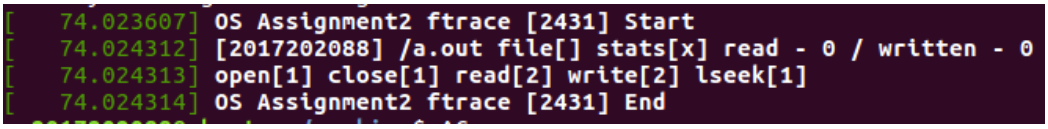
  user영역의 filename을 kernel영역의 fname으로 복사한다.

```
asmlinkage long ftrace_open(const char __user* filename, int flags, umode_t mode
)
{
        // if current pid is tracing process's pid
        struct task_struct* task;
        task = get_current();
        if(task->pid == savedPid)
        {
                // open count + 1
                fcount[0] += 1;
                // copy filename
                copy_from_user(fname, filename, 255);
        }

        return real_open(filename, flags, mode);
}
```

3. 실행결과

- make 후 생성된 두 모듈을 적재하고 out 테스트 파일을 실행시켰다.
  파일명, 함수 수행 횟수는 잘 출력되나, open file name, read/written bytes는 제대로 수행하지 못했다.



4. 결론 및 고찰

- 커널에서 tast_struct와 get_current()를 사용하면 현재 task의 다양한 정보에 접근할 수 있다. ftrace 구현하면서 task->comm으로 현재 프로세스의 파일명, task->pid로 현재 프로세스의 pid를 얻을 수 있었다.

- 커널에서 여러 모듈이 동일한 변수에 접근하게 하고싶다면, EXPORT_SYMBOL( )과 extern을 사용해서 구현할 수 있다.

- Kernel 영역과 user 영역의 값은 일반적인 사용법으로 접근할 수 없으므로, <asm/uaccess.h>에 정의된 교환함수를 사용해서 값을 교환해야 한다.

- Iotracehooking의 io함수를 SYSCALL_DEFINE으로 구현하면 컴파일이 되지 않거나 커널이 다운되는 문제가 있었다. SYSCALL_DEFINE 대신 asmlinkage 함수로 구현하니 문제가 해결되었다.

- ftracehooking의 ftrace를 asmlinkage 함수로 구현하니 parameter가 0으로 고정되는 문제가 있었다. SYSCALL_DEFINE으로 구현함으로써 해결할 수 있었다.

- open의 filename과 read/write의 count parameter의 값이 0으로 존재하는 문제가 발생했다. Filename의 경우 user 영역의 값이므로 copy_from_user를 사용하면 읽힐 것으로 기대했으나 마찬가지로 빈 값만 복사되었다. Read가 제대로 수행되는지 테스트 결과 파일을 제대로 읽어오는 것을 확인해서 실제 count, filename이 0인 것은 아니었다. Parameter의 값에 제대로 접근할 수 없는 어떤 문제가 있는 것으로 추정된다.

5. Reference

- Task_struct
  https://linuxholic.tistory.com/entry/%EB%A6%AC%EB%88%85%EC%8A%A4-Taskstruct-%EA%B5%AC%EC%A1%B0
- System call wrapping
  https://m.blog.naver.com/ite98/120200323032
- EXPORT_SYMBOL
  https://stackoverflow.com/questions/31197345/how-can-i-share-a-global-variable-between-two-linux-kernel-modules