M1522.000800 System Programming
Fall 2018

System Programming Kernel Lab Report

Woohyeon Baek
2017-15782

1. <lab> Kernel Lab

커널 랩의 목표는 리눅스 커널 프로그래밍을 익히고 커널 레벨의 프로그래밍과 사용자 레벨의 것의 차이를 인식하는 데에 있다. 이번 랩을 통해 커널 데이터의 구조를 자유자재로 이용하고 그것들의 처리 방식을 깊이 이해하고자 한다.

2. Implementation

버퍼 랩은 2개의 과제로 이루어져 있다. Process Tree Tracing과 Find Physical Address이다. 두 과제는 여러 리눅스 커널 API를 이용하여 모듈을 만들고, 그것을 커널에 올려 실행하는 방식으로 진행된다. Process Tree Tracing은 모듈을 커널에 올려서 파일을 통해 프로세스의 ID를 전달하면 부모 프로세스들의 ID가 파일로 출력되도록 하는 과제이다. Find Physical Address는 app이란 프로그램이 실행되어 힙 공간이 할당되면 app의 PID와 할당된 공간의 virtual memory 및 physical memory의 주소를 출력하는 과제이다.

2.1 Process Tree Tracing

커널 모듈은 dbfs_ptree.c 파일에 구현하였다. dbfs_module_init() 내부에는 ptree 디렉토리 생성을 위해 debugfs_create_dir(const char *name, struct dentry *parent)를 사용했다. 기본 경로인 /sys/kernel/debug 경로에 생성하기 위해 parent에는 NULL을 넣었다. input과 ptree 파일을 생성하기 위해 각각 debugfs_create_file(const char *name, umode_t mode, struct dentry *parent, void *data, const struct file_operations *fops)와 debugfs_blob(const char *name, umode_t mode, struct dentry *parent, struct debugfs_blob_wrapper *blob)를 이용했다. input 파일의 경우 PID를 쓰는 것만 가능해야 하므로 mode를 S_IWUSR로 주었으며 ptree 생성의 반환 값인 dir을 parent로 넘겨줌으로써 ptree 디렉토리 하위에 파일이 생성되도록 했다. 초기 data는 없으므로 NULL을, input 파일에 데이터가 쓰일 경우 작동될 file_operations 구조체를 넘겼다. ptree 파일의 경우 읽기만 가능하도록 S_IRUSR을, 마찬가지로 parent로 dir을, 그리고 파일 쓰기가 작동하도록 미리 만들어 둔 blob_wrapper 구조체를 넘겼다. blob_wrapper는 blob으로 생성된 파일에 쓰기를 할 버퍼의 역할을 한다. static한 buffer 공간을 생성하여 그 주소를 .data에 넘겼다. dbfs_module_exit()에서는 debugfs_remove_recursive(dir)을 이용하여 ptree 디렉토리 및 그 하위 파일들을 삭제하도록 했다.

input 파일의 file_operations는 write될 경우 PID를 읽고 그 프로세스의 부모 프로세스들을 init (1)부터 차례대로 출력해야 한다. 부모 프로세스의 정보를 담고 있는 task_struct를 구하기 위해 pid_task(find_get_pid(input_pid), PIDTYPE_PID)를 이용하여 얻었다. 부모의 프로세스를 추적하는 순서와 출력 형식의 순서가 반대이므로 print_pid_recursive라는 함수의 재귀호출을 통해 쉽게 구현했다. task_struct에는 real_parent라는 부모 프로세스의 task_struct의 주소를

가리키는 포인터가 있다. 가장 상위 프로세스의 PID는 항상 1이므로 PID가 1이 아니면 real_parent를 이용하여 재귀호출이 들어가도록 했으며, 재귀가 끝나면 현재의 프로세스의 정보를 출력하도록 했다.

make를 하여 커널 모듈을 삽입했으며 ps를 실행시켰더니 현재 실행중인 프로세스의 리스트는 다음과 같았다.

```
PID TTY          TIME CMD
 2839 pts/0    00:00:00 sudo
 2840 pts/0    00:00:00 su
 2843 pts/0    00:00:00 bash
 2982 pts/0    00:00:00 ps
```

sudo 권한에서 input에 2843을 넘긴 후 ptree를 출력했더니 결과는 다음과 같았다.

```
init (1)
xfce4-panel (2308)
xfce4-terminal (2483)
bash (2488)
sudo (2839)
su (2840)
bash (2843)
```

모듈이 제대로 작동하고 있음을 확인했다. 그리고 make clean을 통해 모듈을 삭제하니 ptree 폴더가 삭제됨을 확인했다.

2.2 Find Physical Address

커널 모듈은 dbfs_paddr.c에 구현했다. paddr 디렉토리와 output 파일을 생성 및 삭제는 2.1 Process Tree Tracing과 같은 방법으로 구현했다. app.c 파일의 코드를 보면, output 파일로 struct packet 타입인 pckt의 주소를 보냄을 알 수 있다. 따라서 모듈에서 output 파일에 write 작업이 일어났을 때의 작동 방식을 기술해야 한다. app.c에 있는 struct packet의 정의를 그대로 써서 output 파일을 통해 얻은 pckt의 주소를 struct packet 타입의 것으로 인식시켰다. 본인의 gentoo 리눅스는 x86-64 환경이었다. 이 경우 48비트의 virtual address를 이용하여 4차례에 걸쳐 page table을 참조한 뒤 52비트의 physical address를 얻게 된다. 이를 고려하여 하위 12비트는 virtual page offset의 의미인 vpo에, 나머지 36비트는 9비트씩 virtual page number의 의미인 vpn1, vpn2, vpn3, vpn4에 나누어 담았다. pgd_t 구조체 배열의 시작 값인 pgd는 task_struct 아래의 mm_struct에서 구할 수 있다. task_struct는 2.1 Process Tree Tracing에서와 동일한 방식으로 얻으면 된다. 포인터 타입의 pgd에 vpn1을 offset으로 더한 후 그 주소에 해당하는 pgd_t 구조체에서 pgd를 뽑아냈다. 이 값은 unsigned long이며 64비트 중 중간의 40비트에 pud의 주소에 대한 정보가 담겨 있다. 이를 mask하여 뽑아냈다. page table은 physical address에서 0번부터 점유하는 것이 아니다. 이들은 전체적으로 asm/pgtable.h에 정의되어 있는 PAGE_OFFSET이란 상수만큼 shift되어 있다. 따라서 뽑아낸 주소에 PAGE_OFFSET을 더한 값이 pud의 주소가 된다. 이것을 pud_t * 로 강제 캐스팅하고 위 절차를 반복하여 차례로 pmd

pte도 구했다. pte에서도 offset인 vpn4를 더하고 중간의 40비트를 뽑아낸 뒤 12비트의 vpo를 더해 52비트의 physical address를 얻었다.

make를 하여 커널 모듈을 삽입했으며 sudo 권한으로 app 파일을 실행시켰다. 결과는 다음과 같았다.

```
pid: 4626  vaddr: 17c0010  paddr: 6e30f010
```

gentoo의 제한 메모리인 2GB, 즉 0x80000000 미만인 것으로 보아 physical address가 올바르게 출력된 것으로 보였다.

3. Conclusion

커널 모듈을 올리고 파일을 통해 이 모듈과 통신을 하면서 커널의 각종 정보들을 얻을 수 있었다. 예컨대 사용자의 레벨에서는 얻을 수 없는 모든 부모 프로세스들의 PID나 physical address와 같은 것처럼 말이다. 이 과정에서 사용자 레벨과 커널 레벨의 프로그래밍에서 차이가 있음을 깨달았다. 또한 커널 모듈을 어떻게 작성하며 debugfs는 어떻게 사용하는지 등을 어느 정도 익힐 수 있었다. 처음 배우는 개념들이라 다소 생소하고 API를 어떻게 사용해야 하는지 문서를 찾느라 시간이 많이 소요되었던 것 같다. 그러나 그만큼 배운 것도 많아 도움이 되었다.