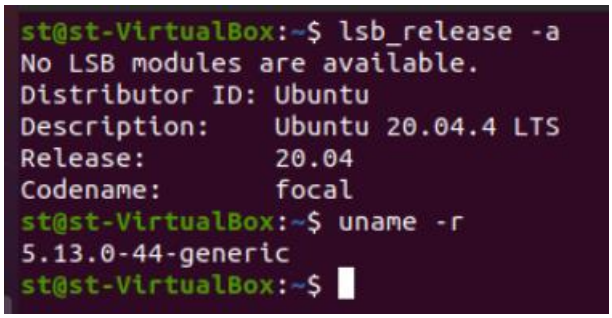
Lab 06 – Kernel

정성태 2018-15515

과제 수행 환경



Part 1 – Tracing Parent Process Tree

커널이 관리하는 task_struct 데이터를 이용해 특정 PID를 가진 process에서 출발해 root까지 거슬러 올라가는 작업이다.

먼저 sscanf()를 호출해 user buffer에 들어 있는 input PID 값을 가져온다. 그러고 나면 이 PID를 find_get_pid() 함수에 전달해 PID 구조체를 얻은 후, 이를 다시 pid_task() 함수에 전달해서 task_struct 구조체를 얻을 수 있게 된다.

그러고 나면 task_struct에 담긴 parent 정보를 이용해 root process까지 거슬러 올라가면 된다. 이때 추적 결과를 마지막에 한 번에 출력해야 하므로 따로 저장할 곳이 필요해지는데, 여기에 커널의 linked list 구조를 이용했다. 각 process의 정보를 담을 node 구조체를 정의한 후 process tree를 한 단계 올라갈 때마다 list에 새로운 node를 삽입해 정보를 저장하도록 했다.

root node에 도달하고 나면 추적은 종료된다. 이제 list에 저장되어 있는 정보들을 한데 모아 output buffer로 출력해주면 된다. 이때는 list_for_each_safe() 매크로를 사용해 list의 node를 하나씩 순회하는 방식을 썼다.

새로 read 동작을 정의해 output buffer의 내용을 다시 user buffer로 읽어올 필요가 있었다. 이때는 simple_read_from_buffer()라는 함수를 사용했다.

또한 입출력을 위한 input과 ptree 파일을 생성하도록 module init 함수를 적절히 수정해주었다.

```
root@st-VirtualBox:/sys/kernel/debug/ptree# ps
    PID TTY          TIME CMD
   2630 pts/2    00:00:00 sudo
   2633 pts/2    00:00:00 su
   2634 pts/2    00:00:00 bash
   2647 pts/2    00:00:00 ps
root@st-VirtualBox:/sys/kernel/debug/ptree# echo 2634 >> input
root@st-VirtualBox:/sys/kernel/debug/ptree# cat ptree
systemd (1)
systemd (963)
tmux: server (2166)
bash (2175)
sudo (2630)
su (2633)
bash (2634)
```

실행 결과 정상 작동함을 확인했다.

Part 2 – Virtual-to-Physical Address Translation

여러 level로 이루어진 page table을 반복적으로 참조하는 과정을 통해 virtual address를 physical address로 변환하는 작업이다.

Part 1에서와 유사하게 먼저 user buffer에서 입력을 읽어 buffer에 저장한다. 그런 다음에는 이 입력을 적절히 fetch 하는 과정이 필요하다. 테스트 프로그램을 참고하면 입력은 packet 구조체에 담겨 있으며, 이는 PID, virtual address, physical address를 순서대로 포함하고 있음을 알 수 있다. 따라서 입력을 char array로 받은 뒤 각 byte를 읽는 과정을 통해 PID와 VA를 추출할 수 있다. PID와 pid_task() 함수를 이용해 task_struct를 불러오면 준비가 완료된다.

이제 VA를 가지고 page table을 반복적으로 참조해 PA를 얻어낼 수 있다. 현재 버전의 linux 커널에서는 pgd – p4d – pud – pmd – pte에 이르는 총 5단계의 참조가 수행되어야 한다. 마지막에 얻은 PTE를 가지고 PA를 얻어낼 수 있게 된다.

이제 이 결과를 다시 packet 구조체의 형식에 맞추어 출력해주면 완료된다. 각 byte의 내용을 작성한 뒤 simple_read_from_buffer()를 호출해 user buffer로 결과를 내보내면 완료된다.

```
root@st-VirtualBox:/home/st/Desktop/kernellab-handout/paddr# make
make -C /lib/modules/5.13.0-44-generic/build M=/home/st/Desktop/kernellab-handout/paddr modu
les;
make[1]: Entering directory '/usr/src/linux-headers-5.13.0-44-generic'
  CC [M]  /home/st/Desktop/kernellab-handout/paddr/dbfs_paddr.o
  MODPOST /home/st/Desktop/kernellab-handout/paddr/Module.symvers
  CC [M]  /home/st/Desktop/kernellab-handout/paddr/dbfs_paddr.mod.o
  LD [M]  /home/st/Desktop/kernellab-handout/paddr/dbfs_paddr.ko
  BTF [M] /home/st/Desktop/kernellab-handout/paddr/dbfs_paddr.ko
Skipping BTF generation for /home/st/Desktop/kernellab-handout/paddr/dbfs_paddr.ko due to un
availability of vmlinux
make[1]: Leaving directory '/usr/src/linux-headers-5.13.0-44-generic'
gcc -o app app.c;
sudo insmod dbfs_paddr.ko
root@st-VirtualBox:/home/st/Desktop/kernellab-handout/paddr# ./app
[TEST CASE]    PASS
```

테스트 결과 정상 작동함을 확인했다.

Review

어려웠던 점

- 커널 프로그래밍 자체를 처음 해보는 것이었기 때문에 모듈의 등록과 해제를 비롯한 새로운 개념을 이해하기가 어려웠다.
- 일반적인 파일이 아닌 새로운 방식의 입출력을 구현하는 부분이 어려웠다.

새로 알게 된 점

- ptree를 구현하는 과정에서 실제로 실행 중인 process에 관한 정보가 어떤 식으로 저장되는지를 더 구체적으로 알게 되었다.
- 커널 프로그래밍을 통해 일반적인 사용자 프로그램에서는 접근하기 어려운 정보까지도 다룰 수 있다는 사실을 알게 되었다.