2019. 11. 21.

Operating System

Assignment # 3

박 철 수 교수님
컴퓨터정보공학부
2015722087
김 민 철

Introduction

이번 과제는 하나의 프로세스의 가상 메모리 영역에 적재되어 있는 여러 가지 정보를 커널 메세지로 출력하는 System Call 을 작성하여 기존의 System Call 인 add 함수를 Hooking 하여 동작을 확인하는 것이다. 이번 과제를 통해 가상 메모리 영역에 대해 알아보며 관련된 구조체와 함수를 학습하는 것이 목적이다.

Reference

[1] dentry

https://kldp.org/node/87172

http://esos.hanyang.ac.kr/tc/david/i/entry//file-%EA%B5%AC%EC%A1%B0%EC%B2%B4%EC%97%90%EC%84%9C-file-name-%EC%B6%9C%EB%A0%A5%ED%95%98%EA%B8%B0#_post_62

[2] vm_area_struct

https://docs.huihoo.com/doxygen/linux/kernel/3.7/structvm__area__struct.html#a6a52d5cbd80393379dd437be3700d2fb

[3] task_struct

https://linuxholic.tistory.com/entry/%EB%A6%AC%EB%88%85%EC%8A%A4-Taskstruct-%EA%B5%AC%EC%A1%B0

[4] dentry_path_raw

https://www.linuxquestions.org/questions/linux-kernel-70/how-to-find-the-complete-file-path-using-struct-file-in-linux-kernel-modules-4175501975/

https://stackoverflow.com/questions/14151521/what-does-the-function-dentry-path-raw-do/28668363

Conclusion

-Analysis

```
os2015722087@ubuntu:~/OS/assignment3$ make test
gcc -o test test.c
os2015722087@ubuntu:~/OS/assignment3$ ./test
7 add 4 = 11
```

우선 기존의 System Call 인 Add System Call 을 test 하는 코드를 작성하여 동작을 확인하였다. test 프로세스에 대한 정보가 위치하는 가상 메모리 주소와 데이터, 코드, 힙 영역의 주소, 그리고 원본 파일의 전체 경로를 출력하는 동작을 수행하도록하는 함수를 Hooking하여 다시 test 해보고 결과를 확인한다.

```
os2015722087@ubuntu:~/OS/assignment3$ sudo insmod file_varea.ko
os2015722087@ubuntu:~/OS/assignment3$ make testwrapping
gcc -o test test.c -DWRAPPING
os2015722087@ubuntu:~/OS/assignment3$ ./test
7 add 4 = 0
os2015722087@ubuntu:~/OS/assignment3$ make
make -C /lib/modules/4.19.67-2015722087/build SUBDIRS=/home/os2015722087/OS/assignment3 modules
make[1]: Entering directory '/home/os2015722087/Downloads/linux-4.19.67'
  CC [M]  /home/os2015722087/OS/assignment3/file_varea.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /home/os2015722087/OS/assignment3/file_varea.mod.o
  LD [M]  /home/os2015722087/OS/assignment3/file_varea.ko
make[1]: Leaving directory '/home/os2015722087/Downloads/linux-4.19.67'
os2015722087@ubuntu:~/OS/assignment3$ sudo insmod file_varea.ko
os2015722087@ubuntu:~/OS/assignment3$ ./test
7 add 4 = 0
```

Hooking 이 된 이후에는 test 해보면 결과 값이 0 으로 출력되는데, 이는 커널 메시지를 출력할 Hooking 된 함수는 0 값을 반환하기 때문이다.

Hooking 할 파일인 file_varea 를 make 하고, 모듈을 적재한 뒤에 Test 해보니 0 으로 return 되는 것을 확인하였다.

```
os2015722087@ubuntu:~/OS/assignment3$ dmesg grep | tail -n -13
[ 1863.745838] init_[2015722087]
[ 1865.764464] ######### Loaded files of a process 'test(11574)' in VM ##########
[ 1865.764468] mem[400000~401000] data[600e10~601048] code[400000~4007d4] heap[2260000~2260000] /home/os2015722087/OS/assignment3/test
[ 1865.764470] mem[600000~601000] data[600e10~601048] code[400000~4007d4] heap[2260000~2260000] /home/os2015722087/OS/assignment3/test
[ 1865.764471] mem[601000~602000] data[600e10~601048] code[400000~4007d4] heap[2260000~2260000] /home/os2015722087/OS/assignment3/test
[ 1865.764472] mem[7fa19ed28000~7fa19eee8000] data[600e10~601048] code[400000~4007d4] heap[2260000~2260000] /lib/x86_64-linux-gnu/libc-2.23.so
[ 1865.764473] mem[7fa19eee8000~7fa19f0e8000] data[600e10~601048] code[400000~4007d4] heap[2260000~2260000] /lib/x86_64-linux-gnu/libc-2.23.so
[ 1865.764475] mem[7fa19f0e8000~7fa19f0ec000] data[600e10~601048] code[400000~4007d4] heap[2260000~2260000] /lib/x86_64-linux-gnu/libc-2.23.so
[ 1865.764476] mem[7fa19f0ec000~7fa19f0ee000] data[600e10~601048] code[400000~4007d4] heap[2260000~2260000] /lib/x86_64-linux-gnu/libc-2.23.so
[ 1865.764477] mem[7fa19f0f2000~7fa19f118000] data[600e10~601048] code[400000~4007d4] heap[2260000~2260000] /lib/x86_64-linux-gnu/ld-2.23.so
[ 1865.764478] mem[7fa19f317000~7fa19f318000] data[600e10~601048] code[400000~4007d4] heap[2260000~2260000] /lib/x86_64-linux-gnu/ld-2.23.so
[ 1865.764479] mem[7fa19f318000~7fa19f319000] data[600e10~601048] code[400000~4007d4] heap[2260000~2260000] /lib/x86_64-linux-gnu/ld-2.23.so
[ 1865.764480] ############################################################
```

이후 dmesg 명령어를 사용하여 커널 메시지를 출력한 결과 정상적으로 동작한 것을 확인하였다. 앞에서부터 가상 메모리 주소, 데이터 주소, 코드 주소, 힙 주소, 원본 파일의 전체 경로를 출력하였다.

```
struct task_struct *p_t;          // task struct pointer t.
struct mm_struct *p_mm;           // mm struct pointer mm.
struct vm_area_struct *p_mmap;    // vm area struct pointer p_mmap
```

사용한 구조체 포인터로는 이렇게 세 가지가 있는데, 가장 먼저 task_struct 는 '프로세스의 속성 정보'를 표현하는 구조체이다. 그 내부에는 '프로세스의 메모리 맵 정보'를 가지고 있는 mm_struct 가 있는데 이 주소를 가리키는 포인터도 같이 사용한다. 또 mm_struct 내부에는 '가상 메모리에 대한 정보'를 가지고 있는 vm_area_struct 가 있는데, 이 주소를 가리키는 포인터까지 총 세 가지를 사용한다.

```
p_t = pid_task(find_vpid(pid), PIDTYPE_PID);     // pointer t => pid's task struct
p_mm = get_task_mm(p_t);                         // pointer mm => process pid's mm struct
p_mmap=p_mm->mmap;                               // pointer mmap => mm's vm area struct
```

인자로 받은 pid 의 프로세스의 속성 정보를 포인트하기 위해 pid_task() 함수를 사용하였다. 이 함수에서는 pid 를 인자로 받아 해당 pid 의 프로세스의 task_struct 의 주소를 반환한다. 이 주소 값을 p_t 가 받아 사용한다.

이후에는 get_task_mm() 함수를 이용하여 p_mm 이 해당 task_struct 내부의 mm_struct 를 가리키도록 하였는데, 이 함수는 task_struct 의 주소를 인자 값으로 받아 해당 구조체 내부의 mm_struct 의 주소를 반환해주는 함수이다.

이후 해당 mm_struct 내부의 vm_area_struct 의 주소를 p_mmap 포인터가 가리키게 하였다.

```
// put the full path of the file
pname = dentry_path_raw(p_mmap->vm_file->f_path.dentry, str, 99);
```

vm_area_struct 내부에는 vm_file 이라는 구조체의 주소가 존재하는데, 이는 구조체 file 형식이다. 이 구조체에는 파일이 어떤 형식으로 Open 되었는지에 대한 정보를 담고있는데 이 구조체 내부의 f_path 라는 구조체의 주소가 존재한다. 이 구조체는 구조체 path 형식으로 내부에 dentry 라는 구조체의 주소를 또 갖고있는데, 이는 directory entry 의 약자이다. 디렉토리에 접근을 빠르게 하기 위한 구조체로 사용된다.
이 구조체를 인자로 dentry_path_raw() 함수를 사용하여 원본 파일의 전체 경로를 구하였는데, 이 함수는 file system 의 root 부터 원본 파일까지의 모든 경로를 반환하는 함수이다.

```
// print information of process to kernel message
printk(KERN_INFO "mem[%ld~%ld] data[%ld~%ld] code[%ld~%ld] heap[%ld~%ld] %s",p_mmap->vm_start,
p_mmap->vm_end, p_mm->start_data, p_mm->end_data, p_mm->start_code, p_mm->end_code, p_mm->
start_brk, p_mm->brk, pname);
```

가상 메모리 주소, 데이터 주소, 코드 주소, 힙 주소, 원본 파일의 전체 경로를 출력하는 함수이다. 커널 메시지로 출력하였으며 가상 메모리 주소는 vm_area_struct 내부의 vm_start, vm_end 를 사용하였으며, 나머지는 mm_struct 내부의 [start_data, end_data], [start_code, end_code], [start_brk, brk] 변수를 사용하여 출력하였다. Pname 은 위의 줄에서 dentry_path_raw 를 통해 구한 원본 파일의 전체 경로이다.

```
if(p_mmap->vm_next == NULL) // not exist next.
    break;
else // point next mmap
    p_mmap = p_mmap->vm_next;
```

vm_area_struct 내부의 vm_next 포인터의 값이 NULL 이면 더 이상 다음 vm_area_struct 가 존재하지 않는다는 의미이고, NULL 이 아니라면 다음 vm_area_struct 가 존재하는 것이므로 한번 더 반복문을 수행하도록 하였다.

-Conclusion

이번 과제는 이론으로만 학습했던 Linux 운영체제에서 프로세스를 관리할 때, 메모리 정보를 관리하는 방법, 데이터가 저장된 구조를 실제로 코드로 작성해보고 결과를 확인해보는 과제였다. 지난 과제인 Assignment 1-3 에서 하였던 Wrapping 과제를 수정하여 진행하였는데, 이번 과제를 수행하게 되면서 지난 과제의 복습까지 하게 되었고, 프로세스와 관련된 정보들이 어떤 구조로 저장이 되는지 확인하였다. 여러 가지 구조체 task_struct, mm_struct, vm_area_struct 에 어떤 정보가 저장이 되어 있는지, 필요한 정보를 찾기 위해 어떤 함수를 사용해야하는지 공부하고 직접 코드로 작성하면서 결과물을 확인하게 되니, 이론으로만 학습할 때 보다 더 이해가 잘된 것 같다.