**운영체제 2차 과제**

**~ Task 자료구조와 LSM의 이해 및 응용 ~**

**2013210111 남세현**

**개발환경**

*VMWare에서 페도라 Core 14, 커널은 2.6.35.6*

**copy\_from\_user와 copy\_to\_user**

두 함수는 커널과 유저프로세스 사이에 데이터를 복사할 때 사용하는 함수입니다.

이 함수가 왜 필요한지 알아보았습니다, 한 사이트에서는

*Why can't you just call, say, memcpy?* Two reasons. One, the kernel is capable of writing to any memory. User process's can't. copy\_to\_user needs to check dst to ensure it is accessible and writable by the current process. Two, depending on the architecture, you can't simply copy data from kernel to user-space. You might need to do some special setup first, invalidate certain caches, or use special operations.

라고 하였는데, 즉 커널은 아무 메모리에 접근이 가능하지만 유저 프로세스는 그러하지 못하고, 아키텍처상 유저와 커널사이의 데이터 카피는 할 수 없다는 것입니다.

두 함수의 사용법은 memcpy와 기본적으로 동일하기 때문에 특별히 작성하지 않겠습니다.

**시스템콜 후킹**

Hooking이란 말 그대로, 중간에 시스템콜의 처리를 가로채는 것 입니다.

이번 숙제에서 Hook.c파일에서 task kill을 가로채는 것 도 시스템 콜 후킹에 해당됩니다.

후킹을 하는 방법은 크게 2가지가 있는데, 이번 숙제에서 한 것처럼 프로그램 실행 전에 물리적으로 값을 수정해서 후킹하는 방법과, 런타임 도중 이벤트를 후킹하는 방법이 있습니다.

본 보고서 작성요령에서는 자세한 설명을 쓰라고 했는데, 자세하게 쓰라는 것이 구현 예를 쓰라는건지, 아니면 무엇을 원하는 것인지 정확하게 알 수 없고, 위 설명과 본 숙제 정도면 충분히 후킹이 무엇인지 감을 잡을 수 있다고 판단되기 때문에 설명은 마치겠습니다.

**MAC ( Madatory Access Control )**

한국어로 강제적 접근 통제 라고 하며, 쉽게 예시를 들어 설명하면, 보안 레벨 요건이 만족되는 사람에게 해당 부분을 보여주고, 낮은 레벨의 보안 수준에게는 노출되지 않도록 접근을 제한하는 방법입니다. 루프가 객체의 보안레벨과 사용자의 보안레벨을 설정할 수 있습니다. 이 모델은 미국 국방부의 스타일이라고 합니다.

**LSM ( Linux Security Modules )**

LSM은 리눅스가 단일 정책에 편중을 두는 것을 방지하기 위한, 여러 보안 모델을 적용할 수 있도록 만든 프레임워크입니다. 리눅스 2.6에서 4개의 표준이 채택되었고, 그 중 하나는 지금 페도라에서 사용하고 있는 SELinux입니다.

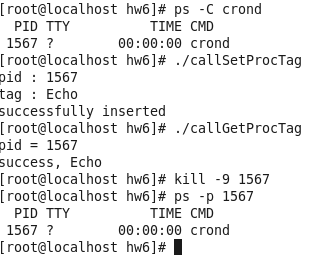
LSM은 위에서 설명한 MAC을 제대로 지원하기 위해 구현이 되었습니다. LSM은 시스템 콜이 중요한 커널 내부 객체를 사용하는 모든 부분에 후크를 넣었습니다.

**수정 및 작성한 부분과 그 이유**

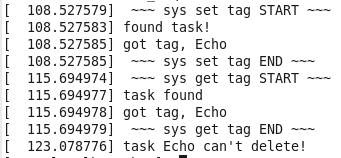
1. 우선 (kernel)/arch/x86/kernel/syscall\_table\_32.S 와 (kernel)/include/linux/syscall.h 에 sys\_get\_tag와 sys\_set\_tag를 추가하였습니다. Syscall 함수로 시스템 콜을 호출하기 위함입니다.
2. Task\_struct에 tag를 추가하기 위해 (kernel)/include/linux/sched.h 에서 struct task\_struct에 char\* tag를 추가하였습니다.
3. 1.에서 추가한 sys\_get\_tag와 sys\_set\_tag의 구현은 (kernel)/2nd\_assign/setProcTagSysCall.c, (kernel)/2nd\_assign/getProcTagSysCall.c 에 작성하였습니다. 같은 폴더에 Makefile을 두어 컴파일 되어 만들어진 object 파일이 커널 컴파일에 추가되도록 설정했습니다.
4. sys\_set\_tag는 맨 처음 입력받은 pid를 통해find\_task\_by\_vpid함수로 task를 찾아냅니다. 그 이후 입력받은 문자열의 길이 + 1 바이트를 kmalloc으로 할당했습니다. +1을 한 이유는 맨 마지막에 NULL문자를 넣기 위함입니다. 할당에 성공하면 copy\_from\_user 함수를 이용해서 할당한 메모리에 문자열을 복사합니다. Task 구조체의 char\* tag 포인터가 할당된 메모리를 가리키도록 합니다. 물론 그 전에 tag가 가리키고 있던 메모리가 있으면 kfree를 이용해 해제하도록 합니다.
5. Sys\_get\_tag는 입력받은 pid로 task를 찾아내고, task의 tag를 내보냅니다.
6. “Echo”라는 태그를 가진 task는 kill되지 않도록 하기 위해서 (kernel)/security/selinux/hooks.c의 task\_kill을 수정했습니다. Task->tag가 “Echo”이면 task\_kill이 되지 않도록 하였습니다.
7. 마지막으로 (kernel)/include/linux/init\_task.h에서 tag = NULL로 초기화되도록 하고, (kernel)/Makefile에서 2nd\_assign를 포함하도록 셋팅한 후 커널을 컴파일했습니다.

**실행 화면**

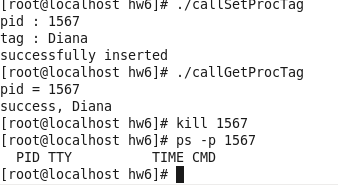
1. **Task의 tag가 “Echo”인 경우**



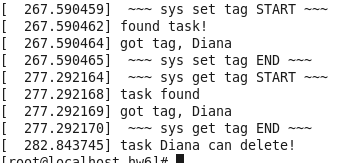
**-dmesg**



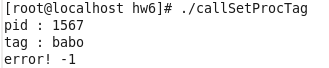
1. **Task의 tag가 “Echo”가 아닌 경우**



**-dmesg**



1. **존재하지 않은 PID의 tag를 설정하려고 했을 때**



**-dmesg**

