운영체제 실습

[Assignment#2]

Class : 목34

Professor : 최상호 교수님

Student ID : 2021202003

Name : 강준우

Introduction

이번 과제에서는 1번째 과제에서 만든 리눅스 환경을 바탕으로 연장 구현을 목표로 한다. 과제에서는 순서대로 시스템콜 접근 , 후킹 기능 구현, 파일 입출력 트레이싱을 포커싱 하여 구현하게 된다. 과정에서 시스템콜 테이블과 시스템콜 함수를 수정하게 되고 이후 내부 기능들을 변형하여 원하는 과정을 얻고자 한다.

결과화면

먼저 2번째 과제를 하기 앞서 Grub 에서 nokaslr 설정을 하도록 한다.

```
root@ubuntu:/home/os2021202003#  sudo cat /proc/kallsyms |
ffffffff82200260 R sys_call_table
ffffffff82201220 R ia32_sys_call_table
root@ubuntu:/home/os2021202003# sudo cat /boot/System.map-
ffffffff82200260 R sys_call_table
ffffffff82201220 R ia32_sys_call_table
root@ubuntu:/home/os2021202003#
```

해당 두 주소가 동일한것을 확인할 수 있다.

Assignment 2-1

해당 과제에서는 os_ftrace라는 라는 시스템콜을 생성한다.

시스템콜의 336번 테이블에 해당 수정을 가하고

이후 시스템콜이 호출될 때 로그가 나오도록 수정한다.

Syscall_64.tbl 파일은 시스템의 콜 테이블을 정의하고 있는 파일입니다.

커널 시스템 콜번호와 시스템 콜 함수가 하나씩 연결되어 있는 모습입니다.

해당 파일에서 336번을 os ftrace 시스템 콜을 호출하게 만들었습니다.

```
331     common  pkey_free                 __x64_sys_pkey_free
332     common  statx                     __x64_sys_statx
333     common  io_pgetevents             __x64_sys_io_pgetevents
334     common  rseq                      __x64_sys_rseq
336 common  os_ftrace       __x64_sys_os_ftrace
# don't use numbers 387 through 423, add new calls after the last
# 'common' entry
424     common  pidfd_send_signal         __x64_sys_pidfd_send_signal
```

syscalls.h 파일은 시스템 콜 함수들의 프로토타입을 선언하는 파일로 콜 호출이 가능하게 합니다.

```
/*
 * Not a real system call, but a placeholder for syscalls which are
 * not implemented -- see kernel/sys_ni.c
 */
asmlinkage long sys_ni_syscall(void);

asmlinkage long sys_os_ftrace(pid_t pid);
#endif /* CONFIG_ARCH_HAS_SYSCALL_WRAPPER */


/*
 * Kernel code should not call syscalls (i.e., sys_xyzyyz()) directly.
```

이후 os_ftrace 경로를 리눅스 Makefile 목록에 추가합니다.

```
  GNU nano 4.8
ifdef need-config
include include/config/auto.conf
endif

ifeq ($(KBUILD_EXTMOD),)
# Objects we will link into vmlinux / subdirs we n
init-y          := init/
drivers-y       := drivers/ sound/
drivers-$(CONFIG_SAMPLES) += samples/
net-y           := net/
libs-y          := lib/
core-y          := usr/
core-y      += os_ftrace/
virt-y          := virt/
endif # KBUILD_EXTMOD

# The all: target is the default when no target is
# command line.
# This allow a user to issue only 'make' to build
# Defaults to vmlinux, but the arch makefile usual
all: vmlinux

CFLAGS_GCOV     := -fprofile-arcs -ftest-coverage
```

```
 GNU nano 4.8                                          os_ftrace.c
#include <linux/kernel.h>
#include <linux/syscalls.h>
#include <linux/types.h>

SYSCALL_DEFINE1(os_ftrace, pid_t, pid)
{
    printk(KERN_INFO "ORIGINAL ftrace( ) called! PID is [ %d ]\n", pid);
    return 0;
}
```

과제에서 요구하는 출력을 위한 로그를 남기는 코드를 작성해줍니다.

```
 GNU nano 4.8
obj-y += os_ftrace.o
```

os_ftrace.c 파일을 컴파일한 오브젝트 파일을 추가하도록 Makefile을 구현합니다.

이외에 .built-in.a.cmd, .os_ftrace.o.cmd, built-in.a, modules.builtin, modules.order 파일들이 make하는 과정에서 생성되었습니다.

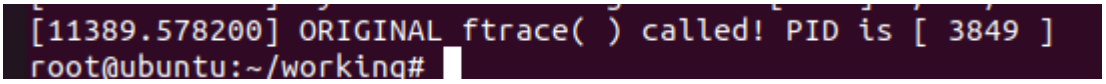
테스트를 위해서 root 경로에 working 디렉토리를 만들고 test 코드를 작성했습니다.

```
#include <stdio.h>
#include <unistd.h> // for getpid()
#include <sys/syscall.h> // for syscall()
#include <sys/types.h> // for pid_t

int main(void) {
    long r = syscall(336, getpid());
    printf("syscall(336) => %ld\n", r);
    return 0;
}
```

해당 코드를 수행하면 syscall(336)=> 0을 확인할 수 있고

이후 dmesg로 마지막 로그를 보면



이렇게 출력된것을 확인 가능합니다.

Assignment 2-2

해당 과제에서는 2-1에서 제작한 os_ftrace를 wrapping하여 또다른 로그가 남겨지게 유도한다.

Os_ftracehooking.c 코드 제작을 통해서 후킹을 구현했다.

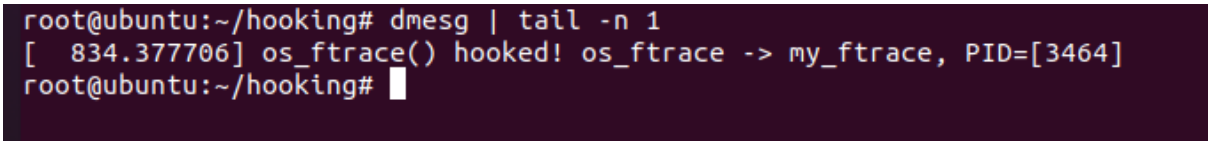
해당 코드에서는 __SYSCALL_DEFINEx(1, _my_ftrace, pid_t, pid)을 통해서 시스템콜을 재정의하여 출력을 조정했다. kallsyms_lookup_name("sys_call_table")로 커널의 시스템콜 테이블 주소를 찾는다. 이후 make_rw()로 테이블 수정을 가능하게 아였고 336번 내용을 __x64_sys_my_ftrace로 변경하였다. hook_exit()을 통해서 기존의 포인터를 저장하여 복구 과정을 구현해두었다.

Hooking이라는 경로를 만들고 스켈레톤 코드를 바탕으로 코드를 재구성하였다.

Make하는 과정에서 정상적으로 잘 후킹되는 모습이다. 추가로 Makefile 하는 과정에서는 다음과 같은 파일들이 생성되었습니다.

os_ftracehooking.mod.c modules.order os_ftracehooking.ko os_ftracehooking.mod.o Module.symvers os_ftracehooking.mod os_ftracehooking.o

이후 기존의 2-1에서의 테스트 코드를 다시 수행한뒤에



정상적으로 후킹이 성공한 모습입니다.

Assignment 2-3

해당 과제에서는 시스템콜을 추격하는 기능을 구현합니다.

ftracehooking.c는 sys_call_table[336]을 후킹해 os_ftrace 대신 my_ftrace가 실행되도록 설정한다. my_ftrace는 전달된 pid가 0보다 크면 추적을 시작하고, 0이면 측정된 데이터를 요약하여 출력한다. iotracehooking.c는 openat, read, write, lseek, close 시스템콜을 각각 ftrace_openat 등으로 대체한다. 각 후킹 함수는 추적 중인 pid와 일치할 경우 호출 횟수와 바이트 수를 카운트한다. 모듈 언로드 시에는 시스템콜 테이블을 원상 복구하여 커널 안정성을 유지한다. System call address, About Open( ), Check glibc version 룰 확인한다.

```
root@ubuntu:/usr/src/linux-5.4.282/hooking# cat /boot/System.map-$(uname -r) | grep "__x64_sys_openat"
ffffffff8133f620 T __x64_sys_openat
ffffffff8324d3e0 t _eil_addr___x64_sys_openat
root@ubuntu:/usr/src/linux-5.4.282/hooking# man open 2
No manual entry for 2
(Alternatively, what manual page do you want from section 2?)
For example, try 'man man'.
root@ubuntu:/usr/src/linux-5.4.282/hooking# getconf -a | grep glibc
GNU_LIBC_VERSION                    glibc 2.31
```

커널 모듈은 __x64_sys_openat 을 후킹해야 올바르게 작동하는 모습이다.

```
C library/kernel differences
    Since  version 2.26, the glibc wrapper function for open() employs the openat() system call, rather than
    the kernel's open() system call.  For certain architectures, this is also true in glibc versions  before
    2.26.
```

유저 프로그램이 open() 함수를 호출하더라도 glibc(라이브러리)는 실제로 커널의 open() 시스템콜이 아니라 openat() 시스템콜을 대신 호출한다고 언급한다.

```
[  435.942930] OS Assignment 2 ftrace [3916] Start
[  435.942935] [2021202003] a.out test file[abc.txt] stats [x] read - 20 / written - 26
[  435.942937] open[1] close[1] read[4] write[5] lseek[9]
[  435.942939] OS Assignment 2 ftrace [3916] End
```

Start / End : os_ftrace(pid) / os_ftrace(0) 호출 시점 추적 구간이 정상적으로 구분됨

read - 20 / written - 26 :
총 읽기/쓰기 바이트 수 test 프로그램의 read(4×5) + write(5×5+‘HELLO’6)=정상 계산

open[1], close[1] :파일 open/close 횟수 abc.txt 한 번 열고 닫음

read[4], write[5], lseek[9] :시스템콜 호출 횟수 루프 내 I/O 동작이 모두 카운트됨

고찰

이번 과제를 통해서 운영체개에서 하는 모든 커널 부분들도 매우 신중하게 설계가 되어있음을 알게 되었습니다. 해당 과제에서는 추가로 2-1부터 2-2까지 수정된 내용에

대해서 추가 후킹까지 수행하면서 우리가 조심스러운 부분까지 처리하는것을 확인할 수 있었습니다. 또한 과제에서 `exit_mod` 를 사용해서 만약 해당 수행이 문제를 일으키거나 삭제된다면 하는 가정까지 관리해야하는 사실을 알게 되었습니다. 또한 make하는 과정에서 다양한 파일들이 생성되는데 이에 대해서 추가로 알아보고자 하는 계기가 되었습니다.

Reference

실습자료 이용했습니다. 감사합니다.