دانشگاه صنعتی شریف

دانشکده‌ی مهندسی کامپیوتر

گزارش کار آزمایشگاه

آزمایشگاه سیستم‌های عامل

گزارش آزمایش شماره ۸

(آشنایی با توابع سیستمی)

شماره‌ی گروه: ۲۰

گروه: ارشیا یوسف‌نیا (۴۰۱۱۱۰۴۱۵)

محمدعارف زارع زاده (۴۰۱۱۰۶۰۱۷)

استاد درس: دکتر بیگی

تاریخ: تابستان ۱۴۰۴

فهرست مطالب

لیست تصاویر

۱ آزمایش ۱

۱.۱ توضیح کد

در آدرس source code/part1/sysaddr.c می‌توان کد کامل ماژول هسته را مشاهده کرد. در ادامه، بخش‌های مهم آن را توضیح می‌دهیم.

در [۱] که در گیتهاب درس به عنوان راهنما داده شده بود، نحوه‌ی کلی نوشتن ماژول سطح هسته و اجرای آن را توضیح می‌دهد. در نوشتن کد این بخش، از اصول و مثال‌های آن کمک می‌گیریم.

برای مشاهده‌ی توابع سطح سیستم، می‌توان از syscall_table که با کمک تابع kallsyms_lookup_name می‌توان آن را پیدا کرد، استفاده کرد. برای مشاهده‌ی آدرس و نام و سایر اطلاعات آنها از تابع kallsyms_lookup می‌توان استفاده کرد. از آنجایی که در نسخه‌های جدید لینوکس آنها به طور مستقیم وجود ندارند، باید آنها را با استفاده از kprobe دریافت کرد [۲].

با روشی مشابه [۲] تابعی می‌سازیم که با کمک kprobe مواردی که می‌خواهیم را به ما دهد. آن را در شکل ۱ می‌توان مشاهده کرد.

```c
static void *get_symbol_addr_by_kprobe(const char *name)
{
    struct kprobe kp = { .symbol_name = name };
    void *addr = NULL;
    int ret = register_kprobe(&kp);

    if (ret == 0) {
        addr = kp.addr;
        unregister_kprobe(&kp);
    } else {
        pr_warn("kprobe register failed for %s (err %d)\n", name, ret);
        addr = NULL;
    }
    return addr;
}
```

شکل ۱: تابع کمکی دریافت خواسته‌ها با کمک kprobe

سپس در تابع __init ابتدا مواردی که می‌خواهیم را با کمک این تابع کمکی دریافت کرده (نیمه‌ی اول تابع)، سپس نام و آدرس توابع سیستمی را دریافت می‌کنیم (نیمه‌ی دوم). نیمه‌ی اول این تابع در شکل ۲ و نیمه‌ی دوم آن در شکل ۳ قابل مشاهده است. در ادامه، این دو نیمه را توضیح می‌دهیم.

```c
void *tbl = NULL;
unsigned long *syscall_table = NULL;
int i;

pr_info("sysaddr: init\n");

my_kallsyms_lookup_name = (kallsyms_lookup_name_t)get_symbol_addr_by_kprobe("kallsyms_lookup_name");
pr_info("sysaddr: kallsyms_lookup_name at %p\n", my_kallsyms_lookup_name);

my_kallsyms_lookup = (kallsyms_lookup_t)get_symbol_addr_by_kprobe("kallsyms_lookup");
pr_info("sysaddr: kallsyms_lookup at %p\n", my_kallsyms_lookup);

tbl = (void *)my_kallsyms_lookup_name("sys_call_table");
pr_info("sysaddr: found symbol 'sys_call_table' at %p\n", tbl);



syscall_table = (unsigned long *)tbl;
pr_info("sysaddr: walking syscall table at %p (printing up to %d entries)\n", syscall_table, MAX_SYSCALLS);
```

شکل ۲: نیمه‌ی اول تابع __init در فایل sysaddr.c

```
for (i = 0; i < MAX_SYSCALLS; ++i) {
    unsigned long entry = syscall_table[i];
    unsigned long symsize = 0, offset = 0;
    char *modname = NULL;
    char namebuf[KSYM_NAME_LEN] = {0};

    if (!entry)
        continue;

    const char *symname = my_kallsyms_lookup(entry, &symsize, &offset, &modname, namebuf);
    if (symname && symname[0]) {
        pr_info("sysaddr: nr=%3d entry=%p  name=%s +0x%lx  module=%s\n",
                i, (void *)entry, symname, offset, modname ? modname : "kernel");
    }
}
pr_info("sysaddr: done\n");
return 0;
```

شکل ۳: نمیه‌ی دوم تابع __init در فایل sysaddr.c

در شکل ۲ ابتدا تابع kallsyms_lookup_name را دریافت کرده که با کمک آن، sys_call_table که تمام توابع سیستمی در آن قرار دارند را دریافت کنیم. سپس تابع kallsyms_lookup که با کمک آن می‌توان نام و آدرس و سایر موارد مربوط به توابع سیستمی را دریافت کرد را با کمک تابع کمکی بالا دریافت می‌کنیم.

در شکل ۳ روی تمام توابع یک حلقه اجرا کرده تا نام و آدرس و سایر موارد مربوط به آنها را چاپ کنیم. در آن ابتدا متغیرهای خالی تعریف کرده، سپس به تابع kallsyms_lookup آنها را می‌دهیم تا پر کند. سپس آنها را چاپ می‌کنیم.

درنهایت، تابع __exit را تعریف کرده که اتمام کار ماژول را گزارش می‌کند، و درنهایت __init و __exit را به عنوان ماژول تعریف می‌کنیم.

۲.۱ نحوه‌ی اجرا و نیازمندی‌ها

برای اجرای این کد، موارد زیر با این دستور باید نصب شوند:

```
sudo apt install make build-essential linux-headers-`uname -r`
```

همچنین برای اجرای کد، لازم است طبق [۱] یک Makefile درست کنیم. این Makefile درست مانند منبع است، فقط یک بخش تست هم به آن اضافه کردیم که تست کردن این کد راحت تر شود. در بخش تست آن، ابتدا dmesg خالی می‌شود. سپس ماژول اضافه می‌شود و یک ثانیه صبر می‌کند تا ماژول به طور کامل اجرا شود، سپس محتویات dmesg نشان داده می‌شود و در نهایت، ماژول حذف می‌شود. در شکل ۴ می‌توان محتویات Makefile را دید. همچنین در آدرس source code/part1 نیز وجود دارد.

```
obj-m += sysaddr.o

KDIR ?= /lib/modules/$(shell uname -r)/build
MODULE = sysaddr.ko

all:
    $(MAKE) -C $(KDIR) M=$(PWD) modules

clean:
    $(MAKE) -C $(KDIR) M=$(PWD) clean

test: all
    @echo "=== Clearing dmesg ==="
    sudo dmesg -C
    @echo "=== Inserting module ==="
    sudo insmod $(MODULE)
    @sleep 1
    @echo "=== Latest dmesg output ==="
    sudo dmesg
    @echo "=== Removing module ==="
    sudo rmmod sysaddr
    @echo "=== Test complete ==="
```

شکل ۴: محتویات Makefile

برای اجرای آن، ابتدا باید دستور make را زد، و سپس با make test آن را تست کرد.

۳.۱ خروجی

در شکل‌های ۵ و ۶ می‌توان خروجی آن را دید. چون کل خروجی در شکل ۶ قابل مشاهده نیست، خروجی کامل تست را در آدرس source code/part1/output.txt می‌توان مشاهده کرد.

```
~/Desktop/az8/part1/test make
make -C /lib/modules/6.14.0-27-generic/build M=/home/arefzarezade/Desktop/az8/part1/test modules
make[1]: Entering directory '/usr/src/linux-headers-6.14.0-27-generic'
make[2]: Entering directory '/home/arefzarezade/Desktop/az8/part1/test'
warning: the compiler differs from the one used to build the kernel
  The kernel was built by: x86_64-linux-gnu-gcc-13 (Ubuntu 13.3.0-6ubuntu2~24.04) 13.3.0
  You are using:           gcc-13 (Ubuntu 13.3.0-6ubuntu2~24.04) 13.3.0
  CC [M]  sysaddr.o
  MODPOST Module.symvers
  LD [M]  sysaddr.ko
  BTF [M] sysaddr.ko
Skipping BTF generation for sysaddr.ko due to unavailability of vmlinux
make[2]: Leaving directory '/home/arefzarezade/Desktop/az8/part1/test'
make[1]: Leaving directory '/usr/src/linux-headers-6.14.0-27-generic'
```

شکل ۵: کامپایل کد ماژول هسته

```
~/Desktop/az8/part1/test  make test
make -C /lib/modules/6.14.0-27-generic/build M=/home/arefzarezade/Desktop/az8/part1/test modules
make[1]: Entering directory '/usr/src/linux-headers-6.14.0-27-generic'
make[2]: Entering directory '/home/arefzarezade/Desktop/az8/part1/test'
warning: the compiler differs from the one used to build the kernel
  The kernel was built by: x86_64-linux-gnu-gcc-13 (Ubuntu 13.3.0-6ubuntu2~24.04) 13.3.0
  You are using:           gcc-13 (Ubuntu 13.3.0-6ubuntu2~24.04) 13.3.0
make[2]: Leaving directory '/home/arefzarezade/Desktop/az8/part1/test'
make[1]: Leaving directory '/usr/src/linux-headers-6.14.0-27-generic'
=== Clearing dmesg ===
sudo dmesg -C
=== Inserting module ===
sudo insmod sysaddr.ko
=== Latest dmesg output ===
sudo dmesg
[  741.321230] sysaddr: init
[  741.349426] sysaddr: kallsyms_lookup_name at 00000000c2cf6647
[  741.363903] sysaddr: kallsyms_lookup at 000000008a5ddc55
[  741.363987] sysaddr: found symbol 'sys_call_table' at 00000000c4ac44b6
[  741.363991] sysaddr: walking syscall table at 00000000c4ac44b6 (printing up to 512 entries)
[  741.364006] sysaddr: nr=  0 entry=00000000db2c3a57  name=__x64_sys_read +0x0  module=kernel
[  741.364016] sysaddr: nr=  1 entry=000000009797b987  name=__x64_sys_write +0x0  module=kernel
[  741.364027] sysaddr: nr=  2 entry=00000000645734bb  name=__x64_sys_open +0x0  module=kernel
[  741.364033] sysaddr: nr=  3 entry=0000000090d7940f  name=__x64_sys_close +0x0  module=kernel
[  741.364040] sysaddr: nr=  4 entry=00000000cbef34ee  name=__x64_sys_newstat +0x0  module=kernel
[  741.364046] sysaddr: nr=  5 entry=00000000d4708b81  name=__x64_sys_newfstat +0x0  module=kernel
[  741.364052] sysaddr: nr=  6 entry=000000002b70e368  name=__x64_sys_newlstat +0x0  module=kernel
[  741.364058] sysaddr: nr=  7 entry=0000000013f82460  name=__x64_sys_poll +0x0  module=kernel
[  741.364064] sysaddr: nr=  8 entry=00000000fb4ec694  name=__x64_sys_lseek +0x0  module=kernel
```

شکل ۶: اجرای کد ماژول هسته

۲ آزمایش ۲

۱.۲ توضیح کد

برای اینکه با LD_PRELOADING بتوانیم عملکرد دستور ls را تغییر دهیم، ابتدا باید بررسی کنیم که این دستور از چه توابعی استفاده می‌کند، سپس یک کد به زبان C بنویسیم که تابعی هم‌نام با تابعی که دستور ls با کمک آن محتویات یک دایرکتوری را پیدا می‌کند ساخته و عملکرد آن را به صورت مورد نظر خودمان تغییر دهیم. درنهایت، آن را به فرمت مناسب برای کتابخانه‌های مشترک در آورده و کاری کنیم که دستور ls آن را به عنوان کتابخانه‌ی مشترک، قبل از بقیه‌ی کتابخانه‌های مشترک لود کند. اینگونه، linker تابع ساخته شده‌ی ما را به جای تابع اصلی به دستور ls می‌دهد.

طبق [۳] می‌دانیم که دستور ls از تابع readdir برای خواندن محتوای یک دایرکتوری استفاده می‌کند. در شکل‌های ۷ و ۸ می‌توانیم تابع readdir جدید که ساخته‌ایم را مشاهده کنیم. همچنین در آدرس

source code/part2/fake_ls.c

فایل کامل این کد قرار دارد. در ادامه‌ی گزارش، بخش‌های اصلی این کد را توضیح می‌دهیم.

```c
struct dirent *readdir(DIR *dirp) {
    static orig_readdir_f_type orig_readdir = NULL;
    if (!orig_readdir) {
        orig_readdir = (orig_readdir_f_type)dlsym(RTLD_NEXT, "readdir");
    }

    static int injecting_fake = -1;
    static int fake_index = 0;

    if (injecting_fake == -1) {
        struct dirent *entry;
        int has_real = 0;
        while ((entry = orig_readdir(dirp)) != NULL) {
            if (strcmp(entry->d_name, ".") && strcmp(entry->d_name, ".."))
                has_real = 1;
        }
        rewinddir(dirp);
        injecting_fake = !has_real;
        fake_index = 0;
    }
```

شکل ۷: نیمه‌ی اول تابع readdir جدید

```c
if (!injecting_fake) {
    return orig_readdir(dirp);
}

if (fake_entries[fake_index]) {
    static struct dirent fake;
    memset(&fake, 0, sizeof(fake));
    strncpy(fake.d_name, fake_entries[fake_index], sizeof(fake.d_name)-1);
    fake.d_type = DT_REG;
    fake_index++;
    return &fake;
}

injecting_fake = 0;
return NULL;
```

شکل ۸: نیمه‌ی دوم تابع readdir جدید

می‌دانیم تابع readdir اصلی، با هربار اجرا شدن، یکی دیگر از فایل‌های موجود در دایرکتوری را خروجی می‌دهد، تا وقتی که فایل دیگری نباشد، که در آن زمان NULL را خروجی می‌دهد [۴]. پس تابع ما نیز طبق خواسته‌ی صورت آزمایش در کوئرا، ابتدا بررسی می‌کند که دایرکتوری خواسته شده خالی است یا نه. برای این کار، ابتدا تابع readdir اصلی را با کمک تابع dlsym از کتابخانه‌ی libdl را پیدا می‌کنیم. آرگومان RTLD_NEXT باعث می‌شود که دفعه‌ی بعدی که این تابع در کتابخانه‌های مشترک دیده شود خروجی داده شود [۵]. سپس، متغیر static به نام injecting_fake با مقدار اولیه‌ی 1- ساخته و اگر مقدار آن هنوز 1- باشد، به آن مقدار مناسب می‌دهیم. دلیل پیاده سازی به این صورت این است که دستور ls این تابع را یک بار لود کرده و چند بار اجرا می‌کند. پس مقدار مناسب به این متغیر، فقط در دفعه‌ی اول اجرا داده می‌شود.

برای بررسی اینکه دایرکتوری خالی است یا خیر، با تابع readdir بررسی می‌کنیم که خروجی‌ای به جز . و .. در آن دایرکتوری است یا نه. اگر باشد، یعنی دایرکتوری خالی نیست.

سپس طبق شکل ۸ اگر دایرکتوری خالی نباشد، خروجی تابع readdir اصلی را می‌دهیم. در غیر این صورت، تعدادی فایل جعلی (که در یک آرایه از رشته ذخیره کرده‌ایم) را خروجی می‌دهد. این دقیقا مطابق خواسته‌ی کوئرا است که در صورت خالی بودن دایرکتوری، محتوای جعلی نشان داده شوند و در صورت خالی نبودن محتوای اصلی.

در نهایت نیز NULL را خروجی می‌دهیم که به دستور ls نشان دهیم تمام فایل‌های دایرکتوری نشان داده شده اند.

۲.۲ نحوه‌ی اجرا و دیدن عملکرد کد

برای اینکه این کد را به فرمت مناسب برای کتابخانه‌های مشترک کامپایل کنیم، به gcc باید دو فلگ -shared و -fPIC را اضافه کنیم [۶]. اولی به gcc می‌گوید که آن را به صورت کتابخانه‌ی مشترک کامپایل کند، نه فایل اجرایی. دومی هم به gcc می‌گوید که آن را به صورت position independent کامپایل کند، به این صورت که آدرس‌های مورد استفاده در کد، وابسته به موقعیت برنامه در حافظه هنگام اجرا نباشد. این به این دلیل مهم است که کتابخانه‌های مشترک در هر زمان و در هرجایی از حافظه ممکن است لود شوند.

همچنین چون از کتابخانه‌ی libdl استفاده شده است، باید تگ -ldl را هنگام کامپایل اضافه کنیم. یک نمونه‌ی مناسب کامپایل این کد به صورت زیر است:

```
gcc -Wall -fPIC -shared -o fake_ls.so fake_ls.c -ldl
```

سپس باید کاری کنیم که این کتابخانه‌ی مشترک لود شود. راحت ترین راه برای این کار این است که آدرس خروجی کامپایل را در فایل /etc/ld.so.preload اضافه کنیم. آن را در شکل ۹ می‌توانید مشاهده کنید.

۳.۲ خروجی کد

در شکل ۹ می‌توانید عملکرد درست این کد را مشاهده کنید. ابتدا دستور ls روی همان دایرکتوری اجرا می‌شود. از آنجایی که آن دایرکتوری خالی نیست، محتوای واقعی آن نمایش داده می‌شود. سپس یک دایرکتوری خالی ساخته می‌شود و دستور ls روی آن اجرا می‌شود. همانطور که می‌توان مشاهده کرد، محتوای جعلی نمایش داده می‌شوند.

```
~/Desktop/az8/part2 pwd
/home/arefzarezade/Desktop/az8/part2

~/Desktop/az8/part2 cat /etc/ld.so.preload
/home/arefzarezade/Desktop/az8/part2/fake_ls.so

~/Desktop/az8/part2 ls
fake_ls.c  fake_ls.so

~/Desktop/az8/part2 mkdir empty

~/Desktop/az8/part2 ls empty
hidden_treasure  projects  secret_notes.txt  todo_list.md
```

شکل ۹: بررسی درستی عملکرد خواسته‌ی آزمایش دوم

مراجع

[۱] Robert W. Oliver II. *Writing a Simple Linux Kernel Module*. Accessed: .2025-08-17 2017. URL: https://blog.sourcerer.io/writing-a-simple-linux-kernel-module-d9dc3762c234.

[۲] xcellerator. *kallsyms_lookup_name is not exported anymore in kernels > 5.7*. Accessed: 2025-08-17. 2021. URL: https://github.com/xcellerator/linux_kernel_hacking/issues/3.

[۳] GNU Coreutils contributors. *ls.c — Source code of ls command*. Accessed: .2025-08-17 2025. URL: https://cgit.git.savannah.gnu.org/cgit/coreutils.git/tree/src/ls.c.

[۴] The Open Group. *readdir — Directory Reading Function*. Accessed: .2025-08-17 2024. URL: https://pubs.opengroup.org/onlinepubs/7908799/xsh/readdir.html.

[۵] The Open Group. *dlsym — Dynamic Linker Function*. Accessed: .2025-08-17 2024. URL: https://pubs.opengroup.org/onlinepubs/009604299/functions/dlsym.html.

[۶] GeeksforGeeks contributors. *Working with Shared Libraries — Set 2*. Accessed: 2025-08-17. 2025. URL: https://www.geeksforgeeks.org/operating-systems/working-with-shared-libraries-set-2/.