دانشگاه صنعتی شریف

دانشکده مهندسی کامپیوتر

فاز دوم پروژه درس سیستم‌های عامل

گردآورندگان

نیما جمالی ۹۶۱۰۵۶۶۱

علیرضا دقیق ۹۶۱۰۵۷۲۳

سینا کاظمی ۹۶۱۰۶۰۱۱

فهرست مطالب

مقدمه

در این فاز از پروژه، آرایه‌ای از کاربران و سطح امنیتی آنها و آرایه‌ای از فایل‌ها و سطح امنیتی آنها را ورودی می‌گیریم. سپس با استفاده از تابع write، این آرایه‌ها را در یک پرونده ارتباطی ذخیره می‌کنیم تا در Kernel Module به آنها دسترسی داشته باشیم. پس از آن فراخوان سیستمی ()open را در این ماژول بازنویسی می‌کنیم و در انتها دسترسی به فایل‌های موجود در لیست را در یک فایل به نام listAccess.txt ذخیره می‌کنیم. شایان ذکر است که ubuntu های نسخه‌ی بالاتر از ۱۸ هنگام تغییر جدول syscall دچار مشکل می‌شدند و لذا از نسخه‌ی ۱۵ ubuntu برای طراحی و اجرای این فاز استفاده کردیم.

ماژول راه‌انداز

در طراحی راه‌انداز علاوه بر توابع استفاده شده در فازهای قبلی یعنی init، exit و llseek از توابع دیگری هم بهره بردیم. با استفاده از تابع device_write آرایه‌های سطح کاربر را ذخیره کردیم و همچنین جدول syscall را بازنویسی کردیم. بدین منظور از تابع newOpen استفاده کردیم که به جای open قبلی در جدول syscall قرار می‌گرفت. همچنین در این تابع آرایه‌ای از دسترسی به فایل‌های مشخص شده را نگه می‌داشتیم و با استفاده از تابع device_read، این آرایه را به سطح کاربر انتقال می‌دهیم. در ادامه به توضیح بیشتر درباره‌ی این توابع می‌پردازیم.

توابع استفاده شده در فازهای قبل

توابع task_init، task_exit و device_llseek مانند فازهای قبل تعریف شده‌اند. صرفا در تابع task_exit جدول syscall را به حالت اولیه‌اش باز می‌گردانیم که این مورد در تابع device_write بیشتر توضیح داده خواهد شد.

تابع device_write

در این تابع ابتدا با استفاده از تابع copy_from_user آرایه سطح کاربر را گرفته و به آرایه‌ای از کاربران و فایل‌ها تبدیل می‌کنیم. سپس جدول syscall را به گونه‌ای تغییر می‌دهیم که با تابع open جدید کار کند.

بازسازی آرایه سطح کاربر

در این قسمت، ابتدا رشته‌ی سطح کاربر را توسط تابع copy_from_user در آرایه‌ی char_arr.array ذخیره می‌کنیم. از دو struct به نام‌های User و File هم استفاده می‌کنیم که هر کدام دارای سطح دسترسی هستند. User یک متغیر از جنس عدد صحیح دارد که همان uid کاربر است و File نیز یک رشته به نام path دارد که آدرس فایل را مشخص می‌کند. حال یک آرایه ۱۰۰ تایی از هر کدام از این struct ها به صورت سراسری تعریف می‌کنیم. سپس با استفاده از فرمتی که برای جداسازی ارقام و رشته‌ها در برنامه‌ی سطح کاربر استفاده کردیم (به بخش برنامه‌ی سطح کاربر مراجعه شود)، آرایه User ها و File ها را در سطح کرنل بازسازی می‌کنیم. همچنین تعداد کاربران و فایل‌ها را در دو متغیر سراسری numOfUsers و numOfFiles نگه می‌داریم تا در بخش‌های دیگر از آنها استفاده کنیم.

بازسازی فراخوان سیستمی open()

در این قسمت از کد، ابتدا جدول syscall را می‌گیریم و تابع open اصلی سیستم را در تابع prevOpen ذخیره می‌کنیم. سپس نیاز داریم تا در جدول syscall تابع open جدید که در کد تعریف کرده‌ایم را به جای open اصلی بنویسیم. اما جدول syscall تنها قابلیت خواندن دارد. یک رجیستر کنترلی به نام cr0 در معماری سیستم وجود دارد که بیت شانزدهم آن بیت محافظت (Protect Bit) است که با CR0_PROT در کد مشخص گردیده است. برای این که در جدول syscall بتوانیم بنویسیم، باید این بیت را به صفر تبدیل کنیم که این عمل با دستور write_cr0(cr0 & ~CR0_PROT) انجام می‌گیرد. سپس به جای open قبلی، تابع newOpen را صدا می‌زنیم. باید دقت داشت که در انتها و هنگامی که کار به پایان رسید، در تابع task_exit لازم است که جدول syscall را به حالت اولیه در آوریم و بیت محافظت را هم دوباره برابر با ۱ کنیم.

تابع newOpen

در این بخش، تابع open را به صورتی که مدنظرمان است، بازنویسی می‌کنیم. برای این کار ابتدا لازم داریم تا با استفاده از دستور get_current_user()->uid.val شناسه‌ی کاربر جاری را دریافت می‌کنیم و در یک متغیر از جنس uid_t قرار می‌دهیم.

سپس کاربر را در میان کاربرهایی که در لیست اولیه موجود بوده‌اند، جست‌وجو می‌کنیم. در صورتی که آن کاربر موجود بود، سطح دسترسی آن‌را از داخل struct مربوطه می‌خوانیم. در غیر این صورت سطح

دسترسی آن را عادی در نظر می‌گیریم. مشابه همین کار را برای فایل انجام می‌دهیم با این تفاوت که اگر فایل در لیست فایل‌های ورودی یافت نشد، (به طور مثال از بین فایل های سیستمی بود که در سطح کاربر وارد نشده بود) سطح دسترسی آن‌را ۰ در نظر می‌گیریم تا با اتکا به همین flag برای جلوگیری از مختل شدن عملکرد سیستم، فایل را با فرمت مورد نظر سیستم باز کنیم. در واقع اگر فایلی در لیست فایل‌های ورودی نباشد، آن را با open قدیمی باز می‌کنیم. حال برای پیاده‌سازی بخش امتیازی، باید در صورتی که یک فایل موجود در لیست ورودی توسط یک کاربر باز شد، اطلاعات مربوط به این دسترسی را در یک فایل ذخیره کنیم. برای این کار، آدرس تمام فایل‌هایی که با یکی از فایل‌های لیست ورودی یکی است، به همراه شناسه‌ی کاربری که قصد بازکردن آن را دارد و همچنین سطح دسترسی کاربر و فایل به همراه mode فایل را در یک آرایه به اسم saveAccesses ذخیره می‌کنیم. از آنجا که شناسه کاربر و mode فایل می‌توانند اعدادی چند رقمی باشند، در قسمتی از کد این اعداد را به رشته تبدیل کرده‌ایم. همچنین برای time از ساختاری به نام timespec استفاده می‌کنیم و آن را به تابع getnstimeofday پاس می‌دهیم. سپس با استفاده از متغیر tv_sec این ساختار و انجام محاسبات ریاضی ساده، زمان را به فرمت UTC به دست می‌آوریم.

بعد از انجام این کارها باید فایل را براساس سطح دسترسی کاربر و فایل، باز کنیم. در مراحل قبلی توانستیم سطح دسترسی فایل داده شده و سطح دسترسی کاربر را تشخیص دهیم.. حال براساس مقدار این سطح دسترسی ها آن‌ها را باز می‌کنیم. اگر سطوح دسترسی برابر باشند، آن‌گاه کاربر می‌تواند هم روی فایل بنویسد و هم از آن بخواند و تابع prevOpen(file , 0x0002 , mode) را بر می‌گردانیم که prevOpen همان تابع open قدیمی است که از sys_call_table برداشته‌ایم. اگر سطح دسترسی کاربر بزرگ‌تر باشد، کاربر می‌تواند فقط از فایل بخواند و در نتیجه prevOpen(file , 0x0000 , mode) را خروجی می‌دهیم. در غیر این صورت یعنی سطح دسترسی کاربر کمتر باشد یعنی کاربر فقط می‌تواند روی فایل بنویسد. در نتیجه prevOpen(file, 0x0001, mode) را باز می‌گردانیم.

تابع device_read

در این تابع آرایه‌ای از کاراکترها که در واقع شامل فایل‌هایی است که در لیست فایل‌های ورودی کاربر موجود بوده‌اند و در این مدت باز شده‌اند، توسط تابع copy_to_user به آرایه‌ی سطح کاربر فرستاده می‌شوند تا از آن طریق در یک فایل مجزا نوشته شوند.

طراحی برنامه‌ی سطح کاربر

در این قسمت اطلاعات ورودی شامل شناسه و دسترسی کاربران و آدرس و دسترسی فایل‌ها در سطح کاربر گرفته می‌شوند و با استفاده از تابع write به ماژول راه‌انداز انتقال پیدا می‌کنند تا هنگامی که دستور سیستمی open در سطح کرنل صدا زده می‌شود، بررسی کند که سطح دسترسی هر کاربر به فایل درخواست شده چگونه است.

کد این واسط در فایل api.c آمده است. در این فاز چون کاربر و فایل داریم، اطلاعات مربوط به آنها را در دو struct به نام‌های User و File نگهداری می‌کنیم. User شامل دو متغیر با نام‌های id و userPrivacy و File شامل دو متغیر path و filePrivacy است. در بخش ابتدایی این کد ابتدا تعداد کاربران و فایل‌ها را ورودی می‌گیریم و به همان اندازه آرایه‌ای از کاربران و فایل‌ها تشکیل می‌دهیم. سپس در خطوط بعدی ورودی، اطلاعات کاربران و فایل‌ها را ورودی می‌گیریم.

سپس نیاز داریم تا این اطلاعات را به برنامه‌ی سطح کاربر انتقال دهیم. از این رو ابتدا پرونده‌ی ارتباطی (first_phase) را باز می‌کنیم و سپس آرایه‌ای از کاراکترها به نام temp تشکیل می‌دهیم. ابتدا شناسه‌های کاربری و سطح دسترسی هر کاربر را با ’%‘ جدا می‌کنیم و در این آرایه می‌نویسیم. پس از اتمام بخش کاربران، با کاراکتر ’?‘ مشخص می‌کنیم که وارد بخش فایل‌ها شده‌ایم و همان کار بخش کاربر را مجددا بر روی فایل‌ها پیاده می‌کنیم. حال این آرایه را بر روی پرونده‌ی ارتباطی می‌نویسیم. همچنین چون برای بخش امتیازی نیاز داریم تا پوینتر فایل را جلو ببریم، این پوینتر را در یک فایل به نام reader.txt ذخیره می‌کنیم. مقدار این پوینتر برابر با سایز اشغال شده‌ی آرایه‌ی temp است که در متغیر i ذخیره گردیده است.

در تصویر زیر آرایه temp بر اساس ورودی چاپ گردیده است.

```
alireza@alireza-VirtualBox:~/Desktop/f$ sudo insmod kernelmodule.ko
[sudo] password for alireza:
alireza@alireza-VirtualBox:~/Desktop/f$ sudo chmod 666 /dev/firs_phase
chmod: cannot access ‘/dev/firs_phase’: No such file or directory
alireza@alireza-VirtualBox:~/Desktop/f$ sudo chmod 666 /dev/first_phase
alireza@alireza-VirtualBox:~/Desktop/f$ gcc api.c
alireza@alireza-VirtualBox:~/Desktop/f$ ./a.out
enter number of users: 2
enter number of files: 2
enter id and state of 1th user: 1000 4
enter id and state of 2th user: 0 3
enter path and state of 1th file: /usr/lib/locale/locale-archive 3
enter path and state of 2th file: /home/alireza/Desktop/f/kernelmodule.c 4
1000%4%0%3%?/usr/lib/locale/locale-archive%3%/home/alireza/Desktop/f/kernelmodule.c%4%
```

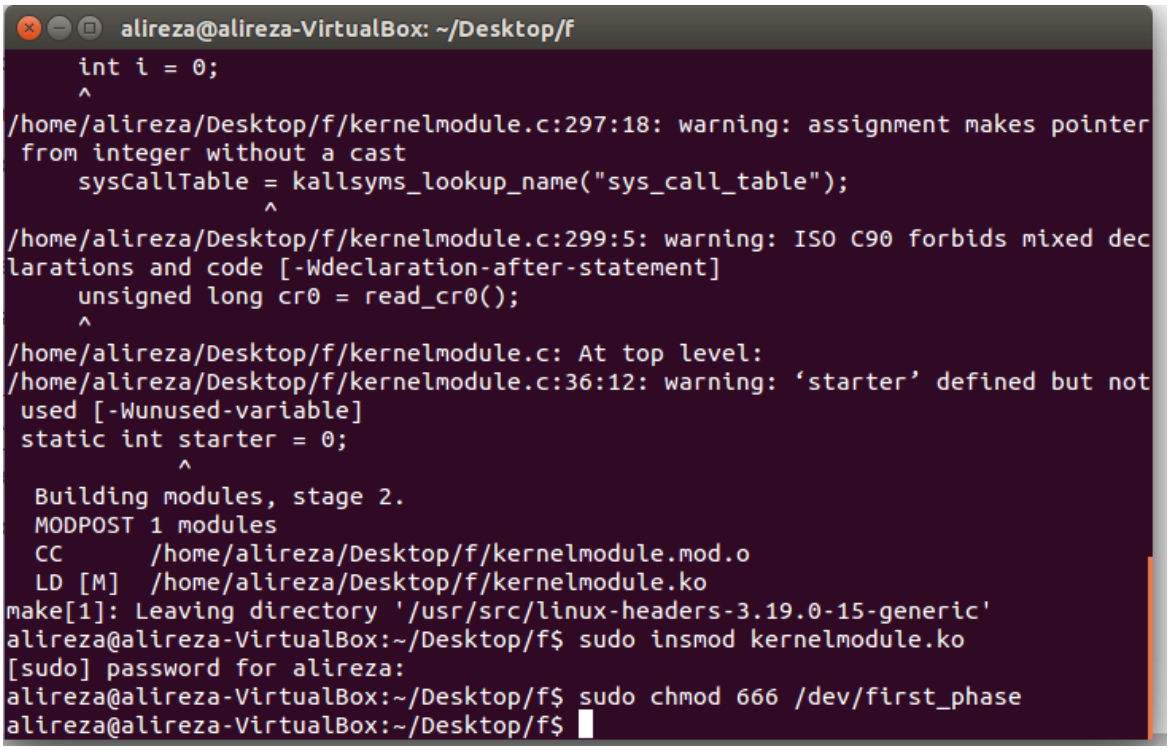
ذخیره دسترسی به فایل‌های مشخص شده در یک فایل

در این قسمت اطلاعات مربوط به فایل‌های موجود در لیست ورودی که open شده‌اند، در یک فایل به نام listAccess.txt ذخیره می‌کنیم. به این منظور در برنامه‌ی read.c، ابتدا فایل reader.txt را باز می‌کنیم و پوینتر پرونده‌ی ارتباطی را که قبلا در آن نوشتیم، می‌خوانیم و با دستور lseek در پرونده ارتباطی به همان اندازه جلو می رویم، پس از آن یک بافر ۵۰۰۰۰ تایی از کاراکتر می‌گیریم و با استفاده از تابع read، اطلاعات مطلوب را از پرونده‌ی ارتباطی می‌خوانیم و با دستور fprintf آن را در فایل listAccess می‌نویسیم.

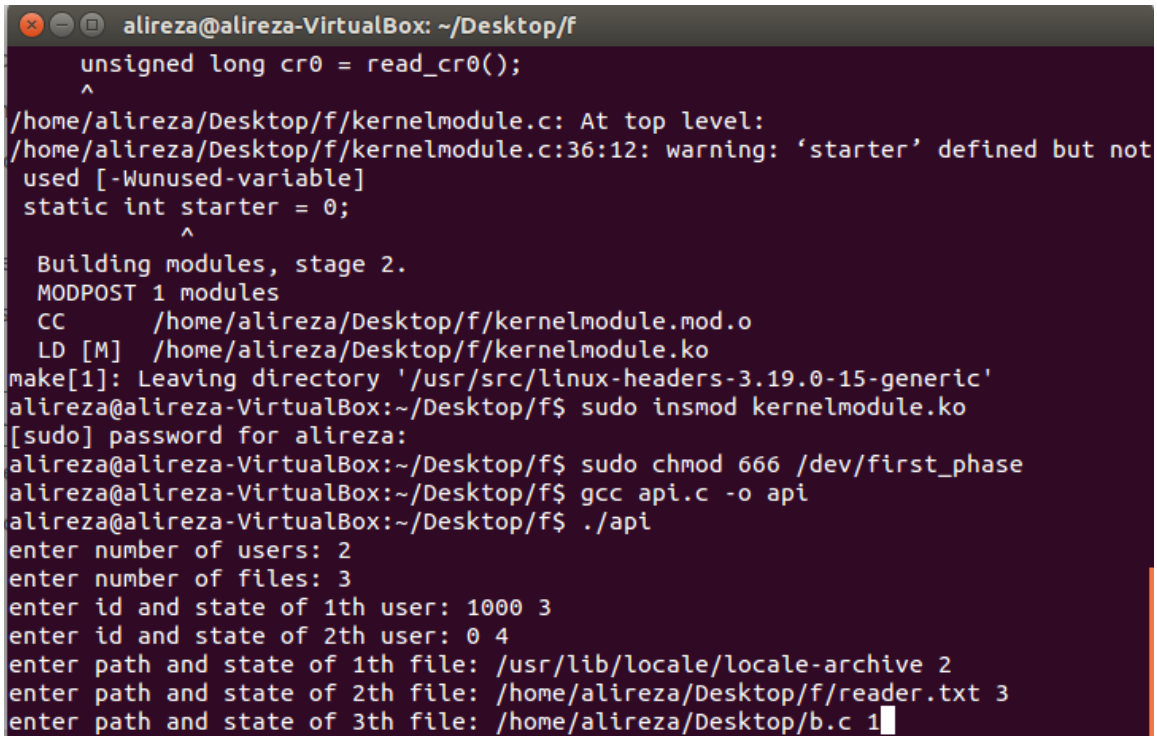
اجرای برنامه

در این بخش تصاویری از یک بار اجرای برنامه نمایش داده می‌شوند.

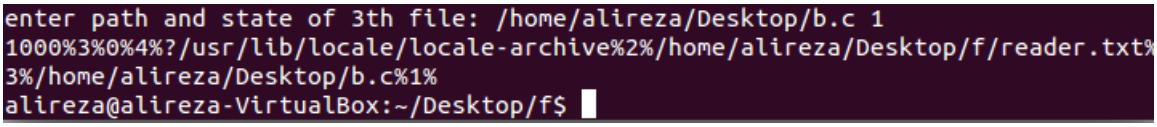
ابتدا راه‌انداز را make می‌کنیم و تابع init آن را با استفاده از دستور sudo insmod صدا می‌زنیم.



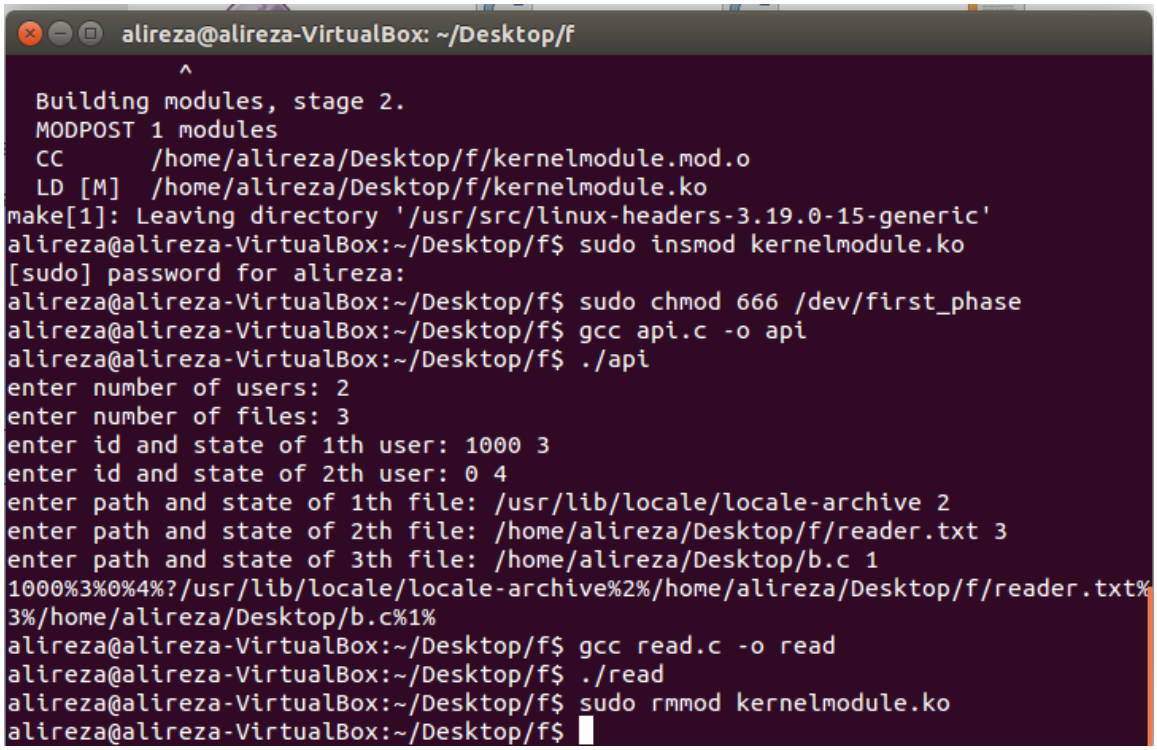
سپس فایل api.c را کامپایل می کنیم و اجرا می کنیم. در این نمونه ورودی، دو کاربر و ۳ فایل گرفته شده‌اند. شناسه‌ی اول مربوط به خود کاربر است (با سطح دسترسی ۳ یعنی فوق محرمانه) و شناسه‌ی دوم مربوط به root (با دسترسی ۴ یعنی سری) است. همچنین ۳ فایل ورودی گرفته شده‌اندکه سطح دسترسی هر کدام در تصویر مشخص است. فایل اول یک فایل سیستمی است که در حین اجرا باز می‌شود. فایل دوم هم در انتهای برنامه‌ی api.c اجرا می‌شود و فایل سوم در کل فرایند باز نمی‌شود.



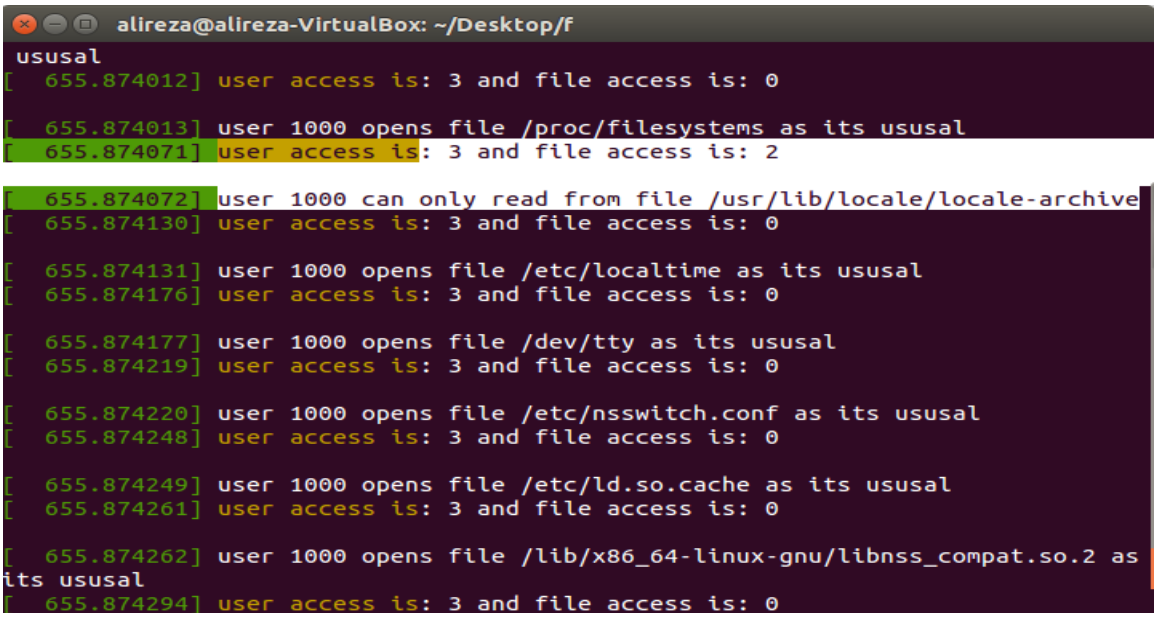
در خروجی آرایه temp که از سطح کاربر گرفته شده و قرار است به کرنل داده شود، چاپ می‌شود.



سپس نوبت به کامپایل و اجرای فایل read.c می‌رسد. در نهایت هم با استفاده از دستور sudo rmmod از راه‌انداز خارج می‌شویم.



حال دستور sudo dmesg را می‌زنیم تا پیغام‌هایی که در سطح کرنل چاپ شده است مشاهده کنیم. باید دقت کرد که در این میان تعدادی برنامه سیستمی نیز باز می‌شوند که در لیست ورودی نبوده‌اند. این فایل‌ها با سطح دسترسی صفر مشخص شده‌اند و به حالت عادی باز می‌شوند.



به عنوان نمونه‌ای دیگر، کاربر با شناسه‌ی ۱۰۰۰ فایل دوم لیست ورودی را باز کرده و با توجه به این که سطح امنیتی کاربر و فایل برابر بوده است، با دسترسی R/W باز شده است.

```
alireza@alireza-VirtualBox: ~/Desktop/f
[  619.458657] user access is: 3 and file access is: 3

[  619.458660] user 1000 can read/write on file /home/alireza/Desktop/f/reader.t
xt
[  619.459464] user access is: 1 and file access is: 0

[  619.459466] user 104 opens file /dev/xconsole as its ususal
[  619.459474] user access is: 1 and file access is: 0

[  619.459475] user 104 opens file /dev/xconsole as its ususal
[  619.459477] user access is: 1 and file access is: 0

[  619.459478] user 104 opens file /dev/xconsole as its ususal
[  619.459480] user access is: 1 and file access is: 0

[  619.459481] user 104 opens file /dev/xconsole as its ususal
[  619.459482] user access is: 1 and file access is: 0

[  619.459483] user 104 opens file /dev/xconsole as its ususal
[  619.459485] user access is: 1 and file access is: 0

[  619.459486] user 104 opens file /dev/xconsole as its ususal
[  619.459488] user access is: 1 and file access is: 0
```

در نهایت اطلاعات مربوط به فایل‌های موجود در لیست ورودی که در این مدت باز شده‌اند، در فایل listAccess.txt قابل مشاهده خواهند بود.

```
kernelmodule.c × listAccess.txt ×
1000 3 /home/alireza/Desktop/f/reader.txt 3  20:26:08
1000 3 /usr/lib/locale/locale-archive 3 0 20:26:17
1000 3 /usr/lib/locale/locale-archive 3 0 20:26:17
1000 3 /usr/lib/locale/locale-archive 3 0 20:26:17
1000 3 /usr/lib/locale/locale-archive 3 0 20:26:17
1000 3 /usr/lib/locale/locale-archive 3 0 20:26:17
```

منابع

https://www.gilgalab.com/blog/2013/01/11/Hooking-Linux-3-syscalls.html
http://codewiki.wikidot.com/c:system-calls:open
https://unix.stackexchange.com/questions/119970/user-id-in-kernel-module
https://stackoverflow.com/questions/13552536/get-current-time-in-seconds-in-kernel-module