Project 4: File System

Yesheng Ma

May 19, 2017

Abstract

In this project, we will mount a romfs image to our virtual machine and do some modifications to the romfs Linux kernel model to accomplish several tasks.

1 Introduction to romfs

Romfs is a space-efficient, small, read-only filesystem originally for Linux and used by some Linux based projects. In this project, our main goals are:

- Mount a romfs to your system.
- Change romfs code to hide a file/directory.
- Change the code to read an encrypted romfs file.
- Change the execution bit and modify display format of a file.

2 Basic romfs Build and Mount

The source code of romfs project is maintained at `romfs.sourceforge.net`. We can generate a romfs image using the `genromfs` tool. You can build `georomfs` from source or if you are using Debian-based Linux, you can directly install it from package manager by `apt install genromfs`.

We can create a romfs image by executing `genromfs -f fs.img -d dir` where `dir` is the directory to be transformed. To mount the romfs, execute `mount -o loop fs.img mnt`, where `mnt` is the directory to be mounted and maybe superuser privilege is needed.

Figure 1: Romfs mount

3 Hide a romfs File

To hide a file in a romfs, we need to do something in the file loop-up routine. When we see a file matching the name to be hided, we need to explicit set the return inode value to `NULL` so that other routines calling `romfs_lookup` will regard this file as if it does not exist.

```
// in romfs_readdir
if(hide_file_name!=NULL)
    ret =romfs_dev_strcmp(i->i_sb, offset+ROMFH_SIZE, hide_file_name,j);
else
    ret = 0;

// in romfs_lookup
if (hide_file_name != NULL && strcmp(name,hide_file_name == 0))
    inode = NULL;
```

Listing 1: Code for file hiding

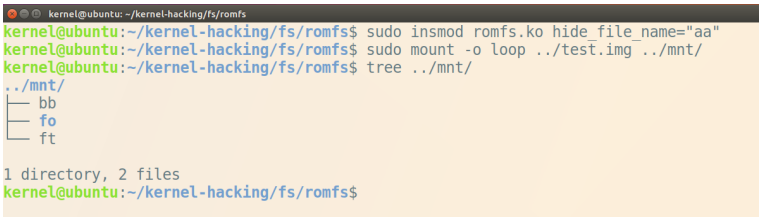


Figure 2: File hiding

4 Encrypt a romfs File

Since in previous projects, we are all ready quite familiar with how Linux handles file context display. We know that to display the content of a file, we are actually write to a buffer at a particular address offset. We first verify whether we want

to encrypt a file and then if the file needs encryption, we write * to the respective buffer.

```
// in romfs_readpage
int encrypt=0;
if (encry_file_name != NULL)
    if (strcmp(fsname,encry_file_name) == 0)
        encrypt=1;
...
if (encrypt == 1) {
    memset(buf,'*',fillsize-1);
    *((char *)buf + fillsize-1) = '\n';
}
```

Listing 2: Code for encryption

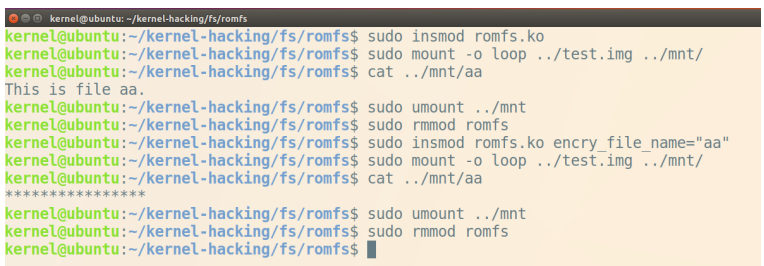


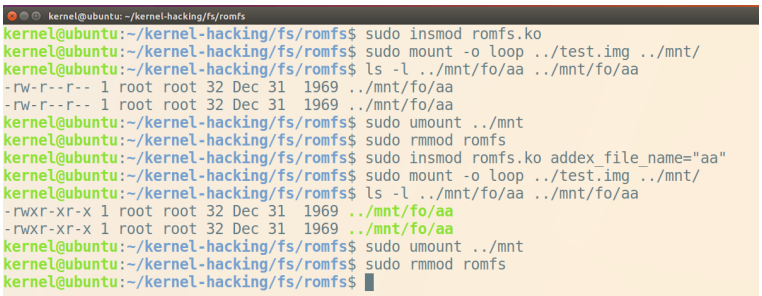
Figure 3: File encryption

5 Change Execution Bit

To change the execution bit of a file, when the inode is indexed, we need to change the `mode` in the inode struct. If the change execution bit option is enabled and the file name matches the target file name, we simply do a mode maks to enable the execution bit.

```
// in romfs_iget
if (addex_file_name != NULL && strcmp(fsname,addex_file_name) == 0)
    mode |= S_IXUGO;
```

Listing 3: Code for encryption



```
kernel@ubuntu: ~/kernel-hacking/fs/romfs
kernel@ubuntu:~/kernel-hacking/fs/romfs$ sudo insmod romfs.ko
kernel@ubuntu:~/kernel-hacking/fs/romfs$ sudo mount -o loop ../test.img ../mnt/
kernel@ubuntu:~/kernel-hacking/fs/romfs$ ls -l ../mnt/fo/aa ../mnt/fo/aa
-rw-r--r-- 1 root root 32 Dec 31  1969 ../mnt/fo/aa
-rw-r--r-- 1 root root 32 Dec 31  1969 ../mnt/fo/aa
kernel@ubuntu:~/kernel-hacking/fs/romfs$ sudo umount ../mnt
kernel@ubuntu:~/kernel-hacking/fs/romfs$ sudo rmmod romfs
kernel@ubuntu:~/kernel-hacking/fs/romfs$ sudo insmod romfs.ko addex_file_name="aa"
kernel@ubuntu:~/kernel-hacking/fs/romfs$ sudo mount -o loop ../test.img ../mnt/
kernel@ubuntu:~/kernel-hacking/fs/romfs$ ls -l ../mnt/fo/aa ../mnt/fo/aa
-rwxr-xr-x 1 root root 32 Dec 31  1969 ../mnt/fo/aa
-rwxr-xr-x 1 root root 32 Dec 31  1969 ../mnt/fo/aa
kernel@ubuntu:~/kernel-hacking/fs/romfs$ sudo umount ../mnt
kernel@ubuntu:~/kernel-hacking/fs/romfs$ sudo rmmod romfs
kernel@ubuntu:~/kernel-hacking/fs/romfs$
```

Figure 4: File execution bit

6 Conclusion

Virtual file system(VFS) is an important abstraction in the Linux kernel. In this project, we learned the romfs, one of the many file systems that support VFS API. This may benefit us when we learn other file systems later.

Acknowledgement

Thanks Prof. Chen for guidance on Linux kernel and TAs for their hard work.