CS 411 Operating Systems II Project 3

A RAM Disk device driver Using the Linux Kernel's Crypto API

Li Li

1 What do you think the main point of this assignment is?

This assignment was great, I enjoyed it a lot. More than 30 hours's work was continually keeping me studying. The main points in the project would be:

- *modules* That is a feeling that you adding some stuff to the linux kernel. It gives a feeling how the kernel works from kernel to differnent modules and how it cames from Linus's personal works to the worldwide Linux Community. These includes initing modules, seting parameter, making config file and finally testing it in the kernel.
- *build a block device* Register, make ramdisk struct and operations.
- *crypto api* Another hard issue to take care. How it works, how to do and how to represent it works.

2 How did you personally approach the problem? Design decisions, algorithm, etc.

The whole solution is base on sbull.c provided in Chapter 16 of Linux Device Drivers. After discussing with Josh, we found that our ramdisk is in the RAM so it doesn't like to have mediachange and revalidate. Also since we present in squidly and only one user would use it a time, so we do not need users in the ramdisk struct as well as timer_list. So I came up with a "lazy" solution that only have two operations: .owner and .ioctl. Then it came up with a lot of problems.

During the debugging time, we figured out that we do need .getgeo. Because ramdisk must be implemented as disk structure in order to set geo infos for the disk-like device. In this way, we also need struct gendisk *gd to initial the disk.

And then it is how we implemented encryption and decryption. My plan is just chose a simple ecb mode and aes-128 as the algorithm. And I did not want to use iv, because it is mostly used in cbc. And even I did not use it, it still did the crypton job. So it is another lazy solution.

3 How did you ensure your solution was correct? Testing details, for instance.

The ramdisk part was easy to test, because if you can correctly login and mkfs to it. It works. However, the crypto part was quite tricky. In the beginning we try to echo some files to it. And read the raw date from the ramdisk, but we came up with null outputs or tons of unreadable stuff. After 6 hours attemptation, we gave up. Luckily, a simple operations saved us: change the key! After we changed the key, the kernel readout still the same stuff as we input. We knew we are in the right direction, so we kept trying and googling. And finally found out that the problem was casued by the cache, so we dumped the cache. And it finally gave out some ramdom ascii chars

that are the right data we are looking for since we give a wrong key. In all, it can represent that the data in ramdisk is encrypted.

4 What did you learn?

As I mentined in the first section, what I saw as importance is also what i learned from the project. Furthormore, I guess next time I would not stick in a wrongside for to long a time, it may have a much simpler way to do it.