Writeup 3

Alex Hoffer
Nehemiah Edwards

CS 444
Spring 2017

Abstract

This third write-up details Alex and Nehemiah's work on Project 3. This project consists of adapting a RAM Disk device driver to include cryptographic protocols using the Linux Kernel's poorly documented crypto API.

Contents

1 Design plan for RAM Disk device driver

We plan to use a RAM Disk device driver (either sbull or sbd) for the Linux Kernel which allocates a chunk of memory and presents it as a block device. This will be developed in the drivers/block directory. Using Linux Kernel's crypto API, we will add the encryption to our block device that will allow the device driver to encrypt and decrypt data as it reads and writes it.

2 What we think the main point of this assignment is

We think that the main point of this assignment is to work with a poorly documented API, which in this case it's the Linux Kernel's crypto API. This thought is supported by the fact that Kevin had stated in class that this was the point of having us do the assignment. In addition, this assignment will make us familiar with how block devices are handled by the Linux kernel. Finally, an important facet of this project is to help us understand how AES encryption and decryption can transform data.

3 How we personally approached the problems

To begin, we began doing some research on how we could implement a RAM Disk device driver for the Linux Kernel. We took look at both sbd and sbull implementations and ultimately decided to go with sbd. After this we looked into the Linux Kernel's crypto API to add encryption to the block device as data is read and written. Once we were finished with this, we looked into how it needed to be integrated into the Linux kernel. We found the only real places within the sbd file (cited within our patch) that we needed to modify were in the *sbd_transfer* and initialization and garbage collection methods. We figured that AES would be a solid choice for a cryptographic protocol because it is ubiquitous. It's important to note that neither of us have much experience in cryptography, however, we found that "Free Electrons" was a valuable source of information and that most of the methods we needed were really straightforward. For example, methods were provided by the API that could initialize the crypto "cipher" struct, set its key, encrypt a chunk of data, decrypt a chunk of data, and perform some garbage collection operations. While the API was most certainly poorly documented, we found that the methods that transformed data within the cipher struct were aptly named and we used them kind of like interfaces: we tried to ignore what they were doing behind the scenes. We believe this Know-Nothing philosophy was effective in the case of this assignment, considering our device driver does work for the most part. Testing it was perplexing, and we never quite figured out how to do it before the patch was due. The hardest part of the project, as usual, was having the virtual machine execute it correctly. It took Alex a really long time to understand how to properly partition a disk, create an ext2 filesystem, and mount the filesystem. It was humbling in how confusing it was, because he had never thought about how disks were to be used. He ran into a weird problem where the disk partition wasn't holding outside of the fdisk GUI. Oddly enough, he solved it by not pressing anything in the fdisk interface and just pressing [Enter].

4 How we ensured solution was correct

To verify that are solution was correct we used mkfs.ext2 to create a ext2 file system on a disk partition. Then, we mounted this filesystem to a directory holding the necessary files on the virtual machine that we scp'd in. Using this we were able to check that our block device was performing the encryption correctly. Alex really wanted to figure out how to use the devmem stuff that McGrath spoke about in class, but couldn't figure out how to correctly pass it arguments. This was

unfortunate, because using devmem would've been awesome because it allows you to pass a physical address in memory and some information to the address. Imagine passing our driver partition to it along with some data! We would've had a much more solid testing metholodogy doing this, but alas, time was running out and we believe that mounting a filesystem corresponding to the disk partition with plenty of kernel print statements gets the job done just fine.

5 What we learned

We learned a bit about how block devices work and how we could add a form of encryption to it. This experience has helped us figure out how we should approach working with other poorly document APIs in the future. We also learned how to integrate a block device to be used by the Linux Kernel. Perhaps one of the most crucial things we learned through the course of this assignment was that you can take any aspect of a Linux kernel and build what you want into it. We also learned how to create modules, build modules using Kconfig and Makefile, and pass modules parameters. Finally, we learned what partitions, filesystems, and mounting was. All three are valuable things to know in the context of systems programming because they allow you to properly apply what you've built to the kernel.

6 Version control log

History of versions

Version	Date	Author(s)	Changes
1	5.10.2017	Nehemiah	Added Project3 folder and start of Latex file
2	5.14.2017	Alex	Add base file sbd
3	5.14.2017	Alex	Find and add a simple block device c file from the internet
4	5.14.2017	Alex	Begin meddling with module parameters
5	5.15.2017	Alex	Change init and exit methods in sbd
6	5.16.2017	Alex	Change sbd_transfer method in sbd
7	5.17.2017	Alex	Add final assignment3.patch in tarball folder
8	5.18.2017	Nehemiah	Additions to report on A3
9	5.18.2017	Alex	Revise and add to writeup.

7 Work log

7.1 Week 1

Alex and Nehemiah began by doing some research on how to implement a RAM Disk device driver for the Linux Kernel. We took at look at sbd and sbull implementations and decided we would go with sbd. Alex decided to go with sbd because it was built against a newer version of the Red Hat distribution of the Linux kernel. This meant it was easier to import into the Linux kernel we're using in this class. Then, Alex poked around the Crypto API and found out how to manipulate the cipher struct. Then, he had to figure out what the parameters in the sbd_transfer file were and how to use them in order to a) take data from a buffer, encrypt it, and write it to a device, b) grab data from a device and decrypt it using a buffer, and c) how to print the unencrypted and encrypted forms of data present in both operations. Finally, he had to figure out how to intialize and destroy the cipher struct instance he was using to implement AES encryption.

7.2 Week 2

Alex added encryption to sbd. We both then tried to work out how to integrate our modified block device into the Linux Kernel on the os-class server. Once we got this figured out we made sure it was correct by using the mkfs.ext2 tool. Alex spent a short amount of time coding and a considerable amount of time trying to get the device driver to run on the virtual machine.