CS411: Individual Writeup - Group 13
Project 3: Encrypted Ram Disk Driver

Trevor Bramwell

November 5, 2012

Introduction

The main point of this assignment was to gain an understanding of Linux device drivers, and to understand how to work with the Linux Crypto API.

This was achieved by having our group write a Encrypted RAM Disk Driver. This driver works by presenting the user with a disk that is stored completely in RAM and is encrypted when written to and decrypted when read from.

Approach

My approach to the problem was to first review the assignment. Second, I for some external resources. This lead me to find the final project for a previous CS411 class which contained a wealth of more information. I informed my group of the discovery, and we followed the new instructions to read *Linux Device Drivers 3rd Edition* (LDD3).

Design

When designing the RAM disk driver, I basically followed all the information in LDD3. Knowing that sitting down and writing out all the source code from the book would be tedious, I opted to find a copy of the *sbull.c* file already written. This lead me to find a simple driver which had been updated for the 3.0 kernel.

Very little had to be done in terms of design, since most of the architecture was handled by the *sbull.c* driver from the book.

Testing

To test the new RAM disk driver I formated the new device */dev/sbd0* that was created when the sbd kernel module was loaded. Then I created a new partition on the device, and also created an ext2 file system. I then created a new file, wrote out some text, closed the file, and then 'cat'ed it to stdout.

Seeing as the printk statements in the driver, along with the data I wrote to the file was outputting correctly. I deduced that the implementation was correct and working accordingly.

Learning

From this assignment I learned a decent bit about how cryptography is handled in the Kernel. Mainly that there are several different methods of using a single algorithm, such as AES. The kernel allows you to use any of the block modes: CBC, ECB, CTS, CTR, XTS, PCBC, and LRW. On top of that you have your choice of algorithm from: AES, Blowfish, DES, Anubis, Twofish, ARC4, or CAST; just to name a few.

Originally when I tried using the AES algorithm, the module would not load. I found that this was because the AES module had to be loaded in as well, and the Kernel was not configured to even build it!

Overall I learned a great deal from this assignment about writing a Linux Kernel driver, encryption, and loading and compiling modules.