# Homework 3:

## *The Crypt Keeper's Key*

**CS 444 : *O*perating Systems II, Oregon State University**

**Group 31**

***Spring Term, 5/27, 2018***

**Prepared by: Zach Tusing, Austin Sanders, and Kevin Talik**

# 1 PURPOSE OF THIS ASSIGNMENT

This assignment had two key purposes. The first was to understand exactly how ram disk drivers work and how to encrypt the data. The second was to learn how crypto tools can be used to protect information on the computer. Files that are encrypted are described as a ”Trapdoor” algorithm. It is easy to write a cypher, but difficult for a computer to decode without a key. Security-wise, it is important to learn about the vulnerabilities of files transferred to a running operating system. If the kernel cannot verify the integrity of the host, the kernel can accept files that can be inserted into the work queue of a processor. Additionally, if the module is encrypted, the owner of the OS can not see what the block device is running. A more complex version of what we are implementing is a credit card reader. This device reads in information and then encrypts the data before reading it out to a centralized server.

# 2 DESIGN

For the design of this code we decided to take a very minimalistic approach. This means only touching on the very necessities that are required for a RAM Disc device driver. This means that we are only going to use a few key functions to do what we need to do. These functions provide the following functionalities: Read/Write, Encryption, Queuing, Disk Geometry, Initialization, and Exit.

The only parameter of the kernel module is the key used for encryption. Since this key is public information, we will Asymmetric Key Generation (The RSA Algorithm) defined in https://github.com/sanderau/OS2_group31/blob/master/linux-yocto-3.19.2/crypto/asymmetric_keys/

## 2.1 Read/Write

With the read and write operations on our Linux machine, we decided to handle the reading and writing with a simple memory allocation into ram and then handling each individual request to write by encrypting the data sent to ram with memcopy. This allowed us a simple and easy way to handle individual requests while also making sure all memory was encrypted at the time. We handled read requests by just checking the read/write bit and seeing which way memory was being copied. This time decryption will be handled after reading out since we will only be handling the buffer. This provided us with a simple execution of a read/write function and our encryption functionality built in.

## 2.2 Encryption

### *2.2.1 One Time Pad Encryption*

One Time Pad Encryption [**?**] is a method of protecting messages by making a new, random key for every message. One Time Pad encryption is one of the strongest types of encryption, as there are many permutations of what a cypher could contain. The strength of OTP is dependent on the integrity of the key to decrypt the cypher that is shared. The problem with OTP is that when you use symmetric keys over asymmetric keys, someone in the middle can decypher messages [**?**].

The RSA algorithm can provide a public key, and a private key between a host and client, protecting the integrity of a device. Even if an external source finds a public key, the private key can encrypt messages from the host machine, and send to other devices that have a shared RSA key.

### 2.3 Encryption Implementation

For the actual implementation of the encryption we used an API provided for us inside of the linux kernel. We used the library fcrypto.h. We also allowed for the user to enter the key if they chose to using command line arguments. However it was not critical and a key was inside the module if the user decided not to enter a key. If they did decide to enter a key the user also needs to include the length of the key. Once those things are out of the way the module just uses several function calls to set the key for the encryption and then just encryption decryption calls to encrypt and decrpyt the data written.

### 2.4 Request Queue

We handled request queuing by implementing a simple request queue. This was done with the function

```
struct request *elv_next_request(struct request_queue *queue)
```

. This function triggers a series of checks to see if the request in the queue actually identifies as a valid request for the driver. If it does validate then it will send it to the read/write function which handles data encryption and Input/Output.

### 2.5 Disk Geometry

We have implemented a standard device driver I/O control function, since this is required for registering devices. This function describes and handles device information for the user and defines the devices geometry(this is what is used to describe how many sections and blocks the disk is using)

### 2.6 INIT

Our Init function is probably the most complex. This is due to the different things that we need to initialize when setting up the driver. Our driver initializes the ram using Kmalloc. It then intializes a block queue using

```
struct request_queue * blk_init_queue (    request_fn_proc * rfn, spinlock_t * lock);
```

This request queue is established and will run requests through our driver. We register the block device with the kernel allowing it to be accessed with a major number. Major numbers are used to call device drivers within the active device queue. We then establish our struct and initialize the data for our disk and then return 0. If there was an error it jumps to the bottom to execute kfrees to allow for easy and non-messy cleanup.

### 2.7 EXIT

This exit function is simple and consists of only five calls. The first call is

```
void del_gendisk(struct gendisk *disk)
```

which is a call to remove the disk from the active disk queue. This call cleans up all data associated with said disk and stops all future requests. The second call is to

```
void put_disk(struct gendisk *disk)
```

This call actually makes it so the disk is no longer accessible. This can be useless in the newer implementations of Linux but it sometimes can clean up things that are missed. The next call is to

```
unregister_blkdev(Int, char*)
```

This call will unregister the major number assigned to the block device. This unregisters the major:minor pairing. The fourth call cleans up the queue which will remove all nodes from the queue. The final call frees the memory allocated for the disk driver.

## 3 TESTING CORRECTNESS

Once the module is inserted into the kernel to test the functionality you just need to use the operating system normally. Once you have run a few programs and done a few things you can go check the syslog in the operating system. Since the module is peppered with printk statements you can just find your modules printk statements inside the log to see what is performing its job and what is failing. Once that informtation is retrieved it is just your typical debugging from there.

## 4 COMMAND LOG

The first thing we did to enable an SCP connection between the host machine and the virtual was change the commands entered when booting the qemu VM. I for the most part used the same command that in the first homework, but I changed one element. Instead of the ”-net none” command I changed it to ”-redir tcp:2222::22”. This would allow me to SCP into qemu on the host machine at the host machine port 2222. The files would then get transfered into qemu on port 22. The command would look like this in total:

- qemu-system-i386 -gdb tcp::5531 -S -nographic -kernel bzImage-qemux86.bin -drive file=core-image-lsb-sdk-qemux86.ext4,if=virtio -enable-kvm **-redir tcp:2222::22** -usb -localtime –no-reboot –append ”root=/dev/vda rw console=ttyS0 debug”

You can now successfully transfer files into the VM using SCP. The command to transfer files into the VM will generally look like this:

- scp -P 2222 $\langle FileToTransfer \rangle$ root@localhost:/home/root/$\langle desiredDirectory \rangle$

Of course if we bound the host machine port to a different port we would use a different port number here, but for the sake of this paper I just used the exact command that I used. With these two commands I could now transfer all the needed files to insert our new module into Yocto’s kernel. To actually make the *.ko file that we will insert into the kernel we just use the makefile that we have included to construct the module and then put the *.ko file into the VM using scp. Once that file is inside the VM we just use the command

- insmod *.ko

The star in the command just represents what we named the module. Once the module is inside the kernel with the insert module command you should be able to use our written module like it is block device driver. To take the module out of the kernel you just use the rmmod and the module name to remove it from the kernel.

After we’ve inserted our device driver into the kernel we run the command

```
mkfs –t ext2 ~/
```

This will create an ext2 file system on our new device. This should be encrypted file system that has the read/write and decrypt/encrypt files respectively.

## 5 WORK LOG

Zach and Austin started the implementation during the week of 5/14. We fished up our block device during the weekend of 5/26.

We wrote the paper on 5/26 and 5/27 where Austin and Zach talked about their implementation and . Kevin wrote about encryption.

## 6 VCS LOG

commit cf5d86c1bfb17bcfe544bfe886ed0273bb4a175c

Author: tusingz ¡zttusing@protonmail.ch¿

Date: Sun May 27 23:17:33 2018 -0700

finalize

commit 4f1cbdb26a7dd4bac6a2c3cee148e2de2114a2e4

Author: Austin Sanders ¡sanderau@oregonstate.edu¿

Date: Sun May 27 23:13:17 2018 -0700

forgot to reinclude a library. No idea how it went missing

commit d3480bee5efb7a684187cb1ed4362640bea8385a

Merge: 34faab7 6b8c4c8

Author: Austin Sanders ¡sanderau@oregonstate.edu¿

Date: Sun May 27 23:00:30 2018 -0700

Merge branch 'master' of https://github.com/sanderau/OS2_group31

commit 34faab71c673043851b405f8cdd336f099ca2729

Author: Austin Sanders ¡sanderau@oregonstate.edu¿

Date: Sun May 27 23:00:14 2018 -0700

The complete module for a simple block driver with basic encryption

commit 6b8c4c8880e4b3241b68d9a8738490e1843a7265

Author: Kevin Talik ¡Kevin Talik¿

Date: Sun May 27 22:10:33 2018 -0700

finalized the encryption section and added a working bibliography

commit 2b7e0b513d8fc7f8f32d5c77ea9d5aa40aaee8a1

Author: Kevin Talik ¡Kevin Talik¿

Date: Sun May 27 17:05:05 2018 -0700

Added a section about the importance of using Asymmetric keys over symmetric keys

commit d56be63445993299dc306551000d16124cf2de5b
Author: Kevin Talik ¡Kevin Talik¿
Date: Sun May 27 15:04:51 2018 -0700
Adding sections on RSA key generation, and the benefits of Asymmetric vs Symmetric Key generation

commit 1c18452df7a4b531c684548446d6e5741ba9959d
Author: Kevin Talik ¡Kevin Talik¿
Date: Sun May 27 14:12:51 2018 -0700
Rephrased the purpose of the crypto assignment

commit 87ace2179b812fb442812e084cc9368a620a22ac
Author: tusingz ¡zttusing@protonmail.ch¿
Date: Sun May 27 03:05:08 2018 -0700
small change

commit cded7168a6cb8826159468e7dd280d8c331d1b46
Author: tusingz ¡zttusing@protonmail.ch¿
Date: Sun May 27 03:04:24 2018 -0700
Adding more code changes and mostly done paper.

commit e1285d6e7d85a13d20d8ff5edd6dda00f52cc418
Author: tusingz ¡zttusing@protonmail.ch¿
Date: Sat May 26 22:45:55 2018 -0700
MORE COMMENTS

commit f18fc2ce053ed9685d39922131af197b6c37cfa8
Author: tusingz ¡zttusing@protonmail.ch¿
Date: Sat May 26 22:43:45 2018 -0700
ading r/w function

commit 35058b22ff0094ae20eafcf72bfb73e97804fef5
Author: tusingz ¡zttusing@protonmail.ch¿
Date: Sat May 26 22:32:00 2018 -0700
I think I got them all now

commit b64e5108895b30c983a7afad886083043f4f71a6
Author: tusingz ¡zttusing@protonmail.ch¿
Date: Sat May 26 22:29:26 2018 -0700

Fixing comments I think

commit 2653312c6b27ea8b9afff82935afbe0764a0f085
Author: tusingz ¡zttusing@protonmail.ch¿
Date: Sat May 26 22:27:53 2018 -0700
small change

commit bf68698f550ceb85223c2a73275d08c0ed85d09f
Author: tusingz ¡zttusing@protonmail.ch¿
Date: Sat May 26 22:23:38 2018 -0700
Adding this file so austin can see it. Still working on modifing it so it accomplishes what we need. Not ready to compile. Still applying the block device and checking yocto systems to make sure it fits between these requirements

commit 5c3c66eff924086e083d5924c8ae5c271c439b40
Author: Austin Sanders ¡sanderau@oregonstate.edu¿
Date: Sat May 26 21:16:18 2018 -0700
User can now add their own key via command line if they choose to

commit 34f02b12892b456c94c2a156fc4ced7bdac55807
Merge: 9797708 48715b4
Author: Austin Sanders ¡sanderau@oregonstate.edu¿
Date: Sat May 26 20:33:54 2018 -0700
Merge branch 'master' of https://github.com/sanderau/OS2_group31

commit 9797708caba0902eb1012299ee3d12a0a6af1d7b
Author: Austin Sanders ¡sanderau@oregonstate.edu¿
Date: Sat May 26 20:33:17 2018 -0700
Alright so the module skeleton now compiles. I can not get it to compile while including the linuxBRD file, but we will figure that out when we need functionality from it

commit 48715b40e9f82d833a0bf269c9188f394230c2db
Author: Austin Sanders ¡sanderau@oregonstate.edu¿
Date: Sat May 26 20:07:40 2018 -0700
Delete makefile

commit b1c25681e34f52be1e85e679373e540d683043a6
Author: Austin Sanders ¡sanderau@oregonstate.edu¿
Date: Sat May 26 20:07:04 2018 -0700

So I guess it matters how you name things when writing modules. Changing the makefile to have capital M because who ever made the compiler made it very clear that capital M makefile is the only way to

commit d9167e96ab0c3eb6592c058e0de945995a5896e0
Author: tusingz ¡zttusing@protonmail.ch¿
Date: Sat May 26 10:10:01 2018 -0700
Adding outline to paper. This only includes a few changes to C file since the product I have still isn't compiling.

commit a23c2a26963257df8a104ac79e22e3529f991a92
Author: tusingz ¡zttusing@protonmail.ch¿
Date: Fri May 25 12:11:49 2018 -0700
Adding the linux implementation for testing with the yocto kernel. This is part of the original kernel that I have no rights to.

commit 2296e45a4c878b6596424f3e4e68f5e795707818
Author: tusingz ¡zttusing@protonmail.ch¿
Date: Wed May 23 21:46:20 2018 -0700
The

commit e8509bdd80562a925f04cda4acc319718dd78153
Author: Austin Sanders ¡sanderau@oregonstate.edu¿
Date: Wed May 23 21:04:09 2018 -0700
adding the makefile

commit 215a2fe38a9f58dd92a9e0bf32e92b4b00a745ef
Author: Austin Sanders ¡sanderau@oregonstate.edu¿
Date: Wed May 23 20:58:58 2018 -0700
Changing source to c

commit 155f082e7245b314a4de651ae0f524fd00a6a896
Author: Zachary Tusing ¡zttusing@protonmail.ch¿
Date: Wed May 23 20:52:02 2018 -0700
Create disk driver file
commit a46d85a6f5907185bc4604b321ec6f0a43374e12
Author: Kevin Talik ¡talikk7@github.com¿
Date: Wed May 23 18:10:36 2018 -0700
Added a source for SBD (Simple Block Driver), by corbet

commit 123db2c2e54a9a861c60c3b7cbbdb435d35156e3
Author: Kevin Talik ¡talikk7@github.com¿

Date: Wed May 23 17:43:06 2018 -0700
Framed the introduction, and the outline

commit 919d65066118e23eaebcd9773979a4daaaa25df2
Author: Kevin Talik ¡Kevin Talik¿
Date: Thu May 17 08:52:19 2018 -0700
Made some edits on the intro

commit 99420307f41ed82d2eb0a3fe7d435680aa34f781
Author: Kevin Talik ¡talikk7@github.com¿
Date: Tue May 15 09:15:08 2018 -0700
Added an introduction... Probably wont use SSH in the end..

commit 46880d0ab665af1b01c35fe1d679e3b0f1fedc9a
Author: Austin Sanders ¡sanderau@oregonstate.edu¿
Date: Tue May 15 08:43:53 2018 -0700

commit 155f082e7245b314a4de651ae0f524fd00a6a896
Author: Zachary Tusing ¡zttusing@protonmail.ch¿
Date: Wed May 23 20:52:02 2018 -0700
Create disk driver file

commit a46d85a6f5907185bc4604b321ec6f0a43374e12
Author: Kevin Talik ¡talikk7@github.com¿
Date: Wed May 23 18:10:36 2018 -0700
Added a source for SBD (Simple Block Driver), by corbet

commit 123db2c2e54a9a861c60c3b7cbbdb435d35156e3
Author: Kevin Talik ¡talikk7@github.com¿
Date: Wed May 23 17:43:06 2018 -0700
Framed the introduction, and the outline

commit 919d65066118e23eaebcd9773979a4daaaa25df2
Author: Kevin Talik ¡Kevin Talik¿
Date: Thu May 17 08:52:19 2018 -0700
Made some edits on the intro

commit 99420307f41ed82d2eb0a3fe7d435680aa34f781
Author: Kevin Talik ¡talikk7@github.com¿
Date: Tue May 15 09:15:08 2018 -0700

Added an introduction... Probably wont use SSH in the end..

commit 46880d0ab665af1b01c35fe1d679e3b0f1fedc9a

Author: Austin Sanders ¡sanderau@oregonstate.edu¿

Date: Tue May 15 08:43:53 2018 -0700

Adding the blank write up files