Members: Mackenzie Huynh (Captain)
Anthony Lin,
Khachatur Mirijanyan,
Priya Calaimany
Team: Evanescence
Date: June 2, 2019

WRITEUP

What We Learned about FUSE:

Q: What is the relationship between FUSE and VFS?
A: FUSE is a layer that allows the implementation of a file system in user space. Normally, file systems are exclusive to kernel drivers.

Q: How does FUSE accomplish this?
A: FUSE attaches itself as a file system kernel driver (using the kldload command) under VFS. It then redirects all the calls to the userspace implementation. The results from userspace return to the VFS through the FUSE kernel driver.

Q: How do we use FUSE to modify VFS functionality?
A: FUSE *does not* modify VFS functionality. VFS must rely on other file system drivers to accomplish its tasks. The FUSE kernel module is an example of a file system driver that does this.

Q: How does VFS know when to call the FUSE kernel driver?
A: The FUSE kernel driver is mounted at a certain point, called the mount point (which is just a folder/directory). When any user request comes from that directory or comes from below that directory, VFS knows to call the FUSE kernel driver, which will service the requests under its jurisdiction.

Q: For example, how does modifying xmp_open modify VFS open?
A: When VFS receives the open call from a user application, it checks which file system driver can service that request and redirects it to that driver. If the open call happens to be on a file in the jurisdiction of the FUSE file system, it is redirected to the FUSE kernel driver, which in turn calls the FUSE open (xmp_open) implementation on userspace.

Q: What are the limitations of using FUSE?
A: FUSE suffers in performance due to the number of transitions, from userspace to kernel space (and vice versa), necessary to fulfill a request. When working with FUSE, fuse doesn't necessarily have a standard output not sure but, doing printf from the fuse operations to print

some information didn't work for us. Thus, debugging was done over logging as information was logged to the debug.log file to debug our code.

What We Learned about Encryption:

Q: What is the purpose of encrypting a file?
A: The purpose of encrypting a file is to prevent unauthorized entities from accessing confidential content.

Q: What is per-file encryption?
A: Per-file encryption means that each file is encrypted with a single key. The key does not necessarily have to be unique between files, meaning two files can be encrypted with the same key. Different files, however, can have different keys.

Q: What are the issues with using the AES implementation provided on Piazza?
A: The CBC mode provided did not update the init vector. This had to be done manually outside the call.

Q: How does the ECB implementation differ from the CBC implementation of AES?
A: The CBC mode introduces an init vector, which transforms the data once again, eliminating patterns that can be guessed by a computer in the ECB implementation.

What We Learned about File Systems:

Q: What does the purpose of an inode?
A: An inode discloses the physical location of a file on the storage media (disk, USB, etc.).

Q: What are the limitations of our encryption system?
A: Limitation 1: If there are multiple threads attempting to access different files that are encrypted, then it is possible that the file access will not be successful (i.e, data will get scrambled).

Members: Mackenzie Huynh (Captain)
Anthony Lin,
Khachatur Mirijanyan,
Priya Calaimany
Team: Evanescence
Date: June 2, 2019

WRITEUP

Limitation 2: If an encrypted file is copied, then it cannot be decrypted. This is because the copied file will use the same inode address as the original file. Instead, the original file must be explicitly read and written to a new file to successfully copy that file.

What We Learned about System Calls:

Q: How are custom system calls integrated?
A: There are two files that are associated with the whole control and access to systems calls and they are in the syscalls.master and vfs_syscalls.c file. In syscalls.master, it its where we define our "prototype" of the system call function. We provide it a unique number as all system functions/calls are each associated with a unique number, and the prototype function/call that it represents. The implementation of the actual system call is done in vfs_syscalls.c. In that file, we define our function and the parameters (if any) for the system call.

Q: How are system calls accessed?
A: For system calls, in order to access them, you use the syscall(int, param1, …). The first parameter which is an integer, is the unique number for the system call which is defined in syscalls.master. Then, the parameters after it are passed into the actual system call function.