Programming Assignment 5:
An Encrypted Filesystem

CSCI 3753 - Operating Systems
University of Colorado at Boulder
Fall 2014

Due Date: Tues, December 9th, 2014 11:55pm

Note: This is numbered as PA5 even though it is the fourth assignment in the course. We skipped PA4 due to time constraints.

1 Assignment Introduction

In this assignment, we take a closer look at filesystems. You will be writing a mirroring filesystem that provides a transparent encryption wrapper on top of an exiting file system. Specifically, you will be using FUSE[2] (Filesystems in USErspace) to implement your filesystem, extended attributes (xattr) to differentiate between encrypted and unencrypted files, and the OpenSSL[8] crypto[10] library to provide secure encryption.

The filesystem itself is a simple mirrored pass-through filesystem. For example, if our pa5-encfs is mounted on the directory */tmp/pa5fs* and is set to mirror the directory */home/user*, then any action performed on the file */tmp/pa5fs/foo.txt* will be translated into an action on the file */home/user/foo.txt*.

Instead of just directly passing actions to the underlying mirrored file, however, your pa5-encfs system will perform encryption and decryption as necessary. Thus, if you create a new file */tmp/pa5fs/bar.txt* in the aforementioned setup and write some data to it, the data will be encrypted before being written to the backing */home/user/bar.txt* file. Likewise, if you were to read the */tmp/pa5fs/bar.txt* file, your filesystem would read the backing encrypted */home/user/bar.txt* file and decrypt the contents before passing it to you. If you were to try to read the backing */home/user/bar.txt* file directly, you would just get encrypted binary gibberish. When your filesystem is unmounted, your data will be securely stored in the mirrored directory and indecipherable to anyone who may encounter it.

2 Your Task

The assignment is primarily a systems project. Thus, you will likely spend more time pulling together and learning to assemble a diverse set of existing APIs then you will spend writing actual code.

This work is best accomplished in a disciplined, iterative manner. Take it one step at a time, and make sure each step is functioning before moving on, and you will do well. Note that a working partial solution is worth more credit then a broken "full" solution.

2.1 Dependencies and Setup

This assignment has several dependencies that must be installed in order for the provided code to build correctly. Note that all of these should already be installed on your CS CU VM. If you are using a different system, you may have to install them manually. On Ubuntu, start by running `sudo apt-get update` to update your package list. Then run `sudo apt-get install <package(s)>` to install the following packages:

- `fuse-utils`
- `libfuse-dev`
- `openssl`
- `libssl-dev`
- `libssl-doc` (optional)
- `libssl1.0.0` or `libssl0.9.8`
- `attr`
- `attr-dev`

You will need a working Internet connection in order to insure these packages install correctly. Note that you can also specify multiple packages in a single `sudo apt-get install <package(s)>` call. Some packages may have their own dependencies, but `apt-get` will automatically take care of installing these for you.

2.2 FUSE

The FUSE API[3] is the core API used in this assignment. It provides a means for implementing filesystems in userspace. Normally filesystems are implemented as kernel modules, but this reduces your ability to utilize userspace libraries. FUSE provides a userspace interface for communicating with the FUSE kernel module that performs the necessary in-kernel work on behalf of your filesystem. Consult the Resources and References sections for additional details.

All filesystems in Unix-like operating systems provide a common interface to the user. We refer to this interface as the Virtual File System (VFS)[6]. The VFS is what allows us to write programs that can be completely agnostic to the underlying filesystems on a given system. Like all Unix filesystems, a FUSE filesystem also implements the VFS interface. The primary task involved in writing a FUSE filesystem is to provide implementations for all of the functions specified by the VFS.

To accomplish this task, your must implement a number of VFS functions, and then pass function pointers to your implementations to the FUSE system via a special struct. See the FUSE documentation and the included FUSE examples for more information [2, 3, 4, 12].

2.3 Encryption

Encryption provides a means for us to mathematically protect data, rendering it unreadable to everyone except those in possession of the necessary encryption key. There is one primary rule to using encryption in your own programs: *Never code your own encryption algorithms.* Writing good encryption libraries is a very difficult task. And only publicly reviewed, widely deployed, and heavily tested libraries written by encryption experts should ever be trusted. No encryption in preferable to bad encryption, as bad encryption just encourages a false sense of security.

Thus, we will be using the OpenSSL[8] crypto library[10] for our assignment. The OpenSSL libraries are the de-facto standard for open source encryption. In particular, we will be using the AES symmetric encryption algorithm with a 256-bit key in CBC mode.

The `aes-crypt-util` demo program provides an example of using AES encryption with a key derived from a user-provided pass phrase. The encryption methods used in this example are also suitable for your assignment.

The most common tripping point when using AES CBC encryption in that encryption and decryption is an all-or-nothing game. You can't encrypt or decrypt part of a file. You must encrypt or decrypt the entire file or none of it. This is due to the fact that CBC encryption schemes are stateful: the transform performed on each encrypted block is a function of the transformed output of the previous block. Thus, if we attempt to start encrypting or decrypting in the middle of a file, we will wind up with data that can not be used. You will need to consider this "entire file or none of it" limitation when designing your system.

Consult the Resources and References sections for additional details. See the OpenSSL documentation and the included `aes-crypt-util` example for more information [8, 9, 10, 11].

2.4 Extended Attributes

The POSIX extended attribute systems (xattr) provides a means to store additional file meta data beyond the standard time-stamp, ownership, and permission data. It provides an interface for associating arbitrary name-value pairs with specific files.

There are four operations (each with a corresponding system call) that allow you to manipulate extend attributes: list, get, set, and remove. See the `xattr-util` example file and the appropriate man pages for details on the use of each function.

Extended attributes must be supported by the underlying filesystem to function correctly. EXT2, EXT3, EXT4, XFS, ResierFS, and a number of additional filesystems provide xattr support. Often, however, this support is disabled by default and must be enabled via the `user_xattr` mount option in the `/etc/fstab` file. See the previous Dependencies and Setup section for additional details on configuring your system.

In addition, the Linux xattr implementation limits userspace use of extended attributes to a specific "user" namespace. The namespace of an extended attribute is defined by the first dot-separated field in its name. Thus, all extended attributes modified from user space must begin with a `user.` prefix. Failure to use this prefix will result in an error when attempting to add, modify, or remove an extended attribute. The included `xattr-util` program provides an example of the standard means for transparently affixing this prefix to

all xattr actions. See [1] for additional xattr namespace information.

Consult the Resources and References sections for additional details. See the xattr documentation, `attr` man page, and the included textttxattr-util example for more information.

2.5 Development Tips

As previously mentioned, you will be best served by taking an iterative approach to this assignment. Start by familiarizing yourself with the examples and available documentation. Then take your implementation one step at a time, insure a single feature is working before moving on to additional features. The following represents one possible feature development order for this assignment:

1. **Start With a Copy of `fusexmp.c`**: This will provide you with the basics of a mirror file system on which to build.
2. **Add Support for Specific Mirror Directory**: `fusexmp.c` only supports mirroring the root directory. Add an additional argument to `main` that allows the user to specify a specific directory to mirror (hint: [12] contains information of passing FUSE private data via `fuse_main` function and accessing it within your VFS implementations). Then modify each VFS function to redirect actions to the corresponding file in the specified mirror directory (hint: this involves a rather simple transformation of the `path` variable that many VFS functions are passed).
3. **Add Support for Encryption**: Add encryption and decryption support to your system. Remember that files must be encrypted and decrypted in a single aes-crypt pass. Also, note that you can use the `aes-crypt-util` function to manually encrypt or decrypt files to aid you in testing your system. Just make sure to use the same key pass phrase when encrypting or decrypting manually as you pass to your filesystem.
4. **Add Support for an XATTR Encrypted Flag** Update the flag when necessary and only perform crypto operations when it indicates that a file in encrypted. Again, you may find the provided `xattr-util` function useful when testing or debugging your system.

If you encounter errors, you may find FUSE's debugging capabilities handy. If you run a FUSE mount executable with the `-d` flag, it will mount the filesystem in debug mode. In this mode, all FUSE status is printed to the terminal. Use one terminal instance to monitor FUSE and another to interact with your filesystem and force specific behaviors and actions. Print output sent to `stderr` from within your implementation is also available via this debug mode.

3 Requirements and Specifications

In addition to the requirements and specification elicited elsewhere in this document, your file system must satisfy the following in order to receive full credit.

- **Executable** Your filesystem should build a standard FUSE executable file called `pa5-encfs` that mounts your file system. Your executable should accept the following call format: `./pa5-encfs <Key Phrase> <Mirror Directory> <Mount Point>`.

- **Read Behavior** When an encrypted file is read through your filesystem, it should be transparently decrypted and the plaintext data passed to the reading application. When an unencrypted file is read through your filesystem, the data should be passed directly to the reading application.
- **Write Behavior** When an existing encrypted file is written through your filesystem, the plaintext data passed from the writing application should be transparently encrypted before being written to the final destination in the mirror directory. When an existing unencrypted file is written through your filesystem, the plaintext data passed from the writing application should be written directly to the final destination in the mirror directory.
- **Create Behavior** When a new file is created through your filesystem, it should be encrypted (even empty files) and flagged as such.
- **Encryption Strength** Encryption should, at a minimum, meet AES 256-bit CBC security levels. The provided `aes-crypt.h` functions meet this requirement.
- **Extended Attributes** Your filesystem should store a flag indicating whether or not a file is encrypted using the the extended attributes for the corresponding mirror file. The flag should reside in the `user` namespace[1] and be named `pa5-encfs.encrypted`. It should have ascii text values of either "true" or "false". If no flag is detected on a specific file, then it should be assumed that the file is not encrypted (same behavior as detecting a flag with a value equal to "false").
- **Supported Functions** At a minimum your filesystem must support the functions included in the `fusexmp` example. Additional functions may be implemented if you deem them necessary for your purposes.

4 What's Included

We provide some code and examples to help get you started. Feel free to use it as a jumping off point (appropriately cited).

- **Makefile** A GNU Make makefile to build all the code listed here.
- **README** As the title so eloquently instructs: read it. Provides usage instructions and examples for files listed here.
- **fusehello.c** A basic "Hello World" FUSE example. See `README` for usage instructions.
- **fusexmp.c** A basic FUSE mirrored filesystem example that mirrors the root directory (/) and supports most standard operations. See `README` for usage instructions.
- **xattr-util.c** A basic extended attribute manipulation program. Provides an example of proper Linux xattr use. See `README` for usage instructions.
- **aes-crypt-util.c** A basic AES encryption program using the local aes-crypt library (see `aes-crypt.h`) and the OpenSSL EVP API[10]. See `README` for usage instructions.
- **aes-crypt.h** A basic AES file-pointer centric encryption library interface. Implemented in `aes-crypt.c`.
- **aes-crypt.c** A basic AES file-pointer centric encryption library implementation. Uses the OpenSSL EVP API[10].

5 What You Must Provide

When you submit your assignment, you must provide the following as a single archive file:

1. A copy of your pa4-endfs FUSE code
2. A copy of any supporting code used by your filesystem
3. A makefile that builds any necessary code
4. A README explaining how to build and run your code

6 Grading

40% of you grade will be based on implementing a filesystem that meets the following criteria. You will be expected to provide functional proof of the following criteria during your grading session.

- **+10 points:** Filesystem properly mirrors target directory specified at mount time.
- **+10 points:** Filesystem uses extended attributes to differentiate between encrypted and unencrypted files.
- **+10 points:** Filesystem can transparently read and write securely encrypted files using a pass phrase specified at mount time.
- **+10 points:** Filesystem can transparently read and update unencrypted files

In addition, the following items are worth extra credit. In no case will the maximum score on the assignment exceed 110/100.

- **+10 extra points:** Filesystem encrypts, hides, or otherwise obfuscates the directory structure to make it cryptographically difficult to determine the names or locations of encrypted files.
- **+10 extra points:** Filesystem encrypts, hides, or otherwise obfuscates file attributes to make it cryptographically difficult to determine the time-stamps, ownership, and permissions of encrypted files.
- **+5 extra points:** Filesystem supports multiple encryption keys for different files, using an additional extended attributes to mark each file with a unique identifier indicating the key used to encrypt it. Filesystem only attempts to decrypt files associated with the currently loaded key. You can think of this as a basic form of multi-user support.
- **+5 extra points:** Filesystem supports multiple encryption keys for a single file. You will probably need to use a layered encryption method, where a master key is used to encrypt each file, and then multiple copies of the master key are themselves encrypted with the necessary additional keys and stored for future retrieval. Such a system would be useful if you wished to provide multiple users with access to an encrypted file with forfeiting your master key. When combined with the previous item, this will implement fairly versatile multi-user support.

If your code does not build or run without errors, you will not receive any credit on the objective portion (40%) of your assignment.

If your code generates warnings when building under gcc on the VM using `-Wall` and `-Wextra` you will be penalized 1 point per warning. In addition, to receive full credit your submission must:

- Meet all requirements elicited in this document
- Code must adhere to good coding practices.
- Code must be submitted to Moodle prior to due date.

The other 60% of your grade will be determined via your grading interview where you will be expected to explain your work and answer questions regarding it and any concepts related to this assignment.

7 Obtaining Code

The starting code for this assignment is available on the Moodle.

8 Resources

Refer to your textbook and class notes on the Moodle for an overview of filesystems.

If you require a good C language reference, consult K&R[7]. If you need an updated C99 reference see Harbison & Steele[5].

The Internet[13] is also a good resource for finding information related to solving this assignment.

You may wish to consult the man pages for the following items, as they will be useful and/or required to complete this assignment. Note that the first argument to the "man" command is the chapter, insuring that you access the appropriate version of each man page. See `man 1 man` for more information. Not all of these man pages are installed be default. Install the previously discussed dependencies or consult an online man page repository if you can not locate a specific man page on your system.

- `man 1 make`
- `man 1 fusermount`
- `man 5 attr`
- `man 2 setxattr`
- `man 2 getxattr`
- `man 2 listxattr`
- `man 2 removexattr`
- `man 3 EVP`
- `man 3 EVP_CipherUpdate`
- `man 3 crypto`
- Many of the system calls used in the FUSE examples also have man pages

In addition, you may find a number of the references in the bibliography helpful.

References

[1] freedesktop.org. *Guidelines for extended attributes.* http://www.freedesktop.org/wiki/CommonExtendedAttributes.

[2] FUSE. *Filesystems in Userspace.* http://fuse.sourceforge.net/.

[3] FUSE. *Fuse Doxygen API Reference.* http://fuse.sourceforge.net/doxygen/index.html.

[4] FUSE. *Fuse Wiki.* http://sourceforge.net/apps/mediawiki/fuse/index.php?title=Main_Page.

[5] Harbison, Samuel and Steele, Guy. *C: A Reference Manual.* Fifth Edition: 2002. Prentice Hall: New Jersey.

[6] Johnson, Michael. *A tour of the Linux VFS.* 1996. http://tldp.org/LDP/khg/HyperNews/get/fs/vfstour.html.

[7] Kernighan, Brian and Dennis, Ritchie. *The C Programming Language.* Second Edition: 1988. Prentice Hall: New Jersey.

[8] OpenSSL. *Cryptography and SSL/TLS Toolkit.* http://www.openssl.org/.

[9] OpenSSL. *OpenSSL Documents.* http://www.openssl.org/docs/.

[10] OpenSSL. *OpenSSL EVP Documentation.* http://www.openssl.org/docs/crypto/EVP_EncryptInit.html.

[11] Pillai, Saju. *Openssl AES encryption example.* Decemper 9th, 2008. http://saju.net.in/blog/?p=36.

[12] Pfeiffer, Joseph. *Writing a FUSE Filesystem: a Tutorial.* January 10th, 2011. http://www.cs.nmsu.edu/~pfeiffer/fuse-tutorial/.

[13] Stevens, Ted. *Speech on Net Neutrality Bill.* 2006. http://youtu.be/f99PcP0aFNE.