COMP7024 Coursework2- Developing a file encryption system for the Minix Operating System

19129163 Mohammad Ali Khan

April 2023

1 Introduction

The objective of this project is to develop a program for the Minix 3.4 Operating System (OS) that will encrypt and decrypt files on the system.

This program will be written in the C programming language, and has a GitHub repository which can be found at https://github.com/mhdl1991/19129163-COMP7024-Coursework2

This program will be developed as a **daemon**.

2 A Brief note on Daemons

A **Daemon** (sometimes claimed to be an acronym for*Disk And Execution MONitor*) is a computer program that runs as a background process, not under direct control of a user, and supervises the system or provides functionality to other processes . Other terms for daemons include *service*, *ghost job*, or *started task*.

Examples of daemons include **init**, **crond**, **httpd**, and **syncd**, all of which perform useful tasks in the operating system.

3 Requirements

We will need the following:

- Knowledge of the Minix OS and filesystem
- Knowledge of signals and signal handling

- libraries for performing Encrpytion and file handling (some may already be installed as part of the Minix OS)
- enough memory for handling the encryption and decryption
- Some way to store credentials (keys and initialization vectors)
- Some way to know which files have been encrypted and which have not
- permissions and policy settings to allow the daemon to alter files.

4 Encryption

A common bit of advice regarding encryption is *"Never roll your own cryptosystem"*- from technical and security standpoints it is better to use an existing, tried and tested cryptosystem than to develop your own- From a **technical** standpoint, it's very difficult to build your own cryptosystem and test it, and to make it **secure**, which ties into the **security** standpoint for not rolling your own cryptosystem.

For this software, we have the option of using **OpenSSL**, which can be installed on Minix using the **pkgin** utility.

As per the OpenSSL documentation, it contains RSA, SHA, DES, SSL, TLS, and AES cipher suites/families. For this program we are using 256-bit AES encryption in CBC (Cipher Block Chaining) mode, using OpenSSL's **EVP library** (*OpenSSL EVP documentation* n.d.).

5 Design

the program will have two main functions, *file_encrypt* and *file_decrypt*. Both functions will take a pointer to a file, a key, and an IV (initialization vector).

the credentials (the key and IV) will be stored in an external file instead of being hard-coded into the daemon.

for handling and detecting changes to the file system, we have access to the **inotify** API (*inotify manpage* n.d.).

6 Development

The program was first built as a standalone bit of C before attempting to integrate it with the Minix operating system. This was done to make sure the file encryption and decryption functions worked properly and didn't result in bugs, memory leaks, unintended alterations to files, or other unintended consequences, and to reduce damage to the system.

6.1 A note on developing encryption/decryption functions

The encryption/decryption functions were more or less adapted from sample code in the OpenSSL documentation, with added code to take the contents of a file and read them into a **unsigned char** buffer, which the OpenSSL functions for encryption/decryption accept as arguments, and then write unsigned char buffers to files after encryption/decryption.

```
int file_encrypt(char *in_file, char *out_file, unsigned char *key, unsigned char *iv) {
    EVP_CIPHER_CTX *ctx;
    unsigned char *plaintext, *ciphertext;
    int plaintext_length,  ciphertext_length, len;

    FILE *f_in = fopen(in_file, "rb");
    if (f_in) {                                 // open the file and read it into  the plaintext_buffer
        fseek(f_in, 0, SEEK_END);
        plaintext_length = ftell(f_in);         // get the length/size of the plaintext file
        if (!plaintext_length) {return 1;}
        fseek(f_in, 0, SEEK_SET);               // return to the start of the plaintext file
        plaintext = malloc(plaintext_length);   // declare space for plaintext and ciphertext buffers
        ciphertext = malloc(plaintext_length);
        if (plaintext) { fread(plaintext, 1, plaintext_length, f_in); }        // read contents of file  into buffer
        fclose(f_in);
    } else { return 1; }                        // failure when opening file
    if (!plaintext) { return 1; }

    if(!(ctx = EVP_CIPHER_CTX_new())) { handle_errors(); }                 // Create and initialise the context
    // Initialise the encryption operation.
    if(1 != EVP_EncryptInit_ex(ctx, EVP_aes_256_cbc(), NULL, key, iv))  { handle_errors();}
    // Provide the message to be encrypted, and obtain the encrypted output.
    if(1 != EVP_EncryptUpdate(ctx, ciphertext, &len, plaintext, plaintext_length)) { handle_errors(); }
    ciphertext_length = len;
    // Finalise the encryption. Further ciphertext bytes may be written at this stage.
    if(1 != EVP_EncryptFinal_ex(ctx, ciphertext + len, &len)) {handle_errors();}
    ciphertext_length += len;
    // Write the ciphertext buffer to file.
    FILE *f_out = fopen(out_file, "wb");
    if (f_out) {
        fwrite((unsigned char *) ciphertext, ciphertext_length, 1, f_out);
        fclose(f_out);
    }
    // Cleanup
    if (ctx) { EVP_CIPHER_CTX_free(ctx); }
    // free up buffers
    if (ciphertext) { free(ciphertext); }
    if (plaintext) { free(plaintext); }
    return 0;
}
```

Figure 1: C function that encrypts a file using AES-256

```
#include<signal.h>
#include<sys/types.h>
#include<sys/stat.h>
#include<syslog.h>

static void skeleton_daemon()
{
        pid_t pid;
        pid = fork();                           // fork off parent process
        if (pid < 0) { exit(EXIT_FAILURE);}
        if (pid > 0) { exit(EXIT_SUCCESS);}     // terminate parent
        if (setsid()< 0) {exit(EXIT_FAILURE);}
        signal(SIGCHLD, SIG_IGN);               // catch/handle signals
        signal(SIGHUP, SIG_IGN);
        pid = fork();                           // fork off again
        if (pid < 0) {exit(EXIT_FAILURE);}
        if (pid > 0) {exit(EXIT_SUCCESS);}
        umask(0);                               // set new file permissions
        chdir("/");                             // go to root
        int x;
        for (x=sysconf(_SC_OPEN_MAX); x>=0;x--) // close file descriptors
        {close(x);}
        openlog("demon_log",LOG_PID,LOG_DAEMON);// open the log file
}
```

Figure 2: C function that provides a ”skeleton” for making a daemon

6.2 A note on Daemon development

the basic principle of how a daemon operates involves the following steps:

- Fork off the parent process (usually init)
- Change file mode mask (umask)
- Opening logs for writing *(optional)*
- Create a unique Session ID (SID)
- Change the current working directory
- Close standard file descriptors
- Enter actual daemon code

It will then run until system shutdown, and handle **signals** sent by the Operating system.

Minix also provides the **daemon** command that turns other processes into daemons, automatically performing the tasks needed to set them up as such (*MINIX manpage on daemon command* n.d.).

7 Testing

The program is tested in it’s standalone form first, to make sure that it is encrypting and decrypting files properly

It will be tested on a number of different file types and sizes, to see if encryption and decryption properly reverts the file back to it's former self.

8 Conclusion

Operating systems are remarkably complex, even Minix, an OS built as an educational tool to show how OS kernels work, has a huge degree of complexity that must be taken into account when attempting to add on or expand on it.

References

inotify manpage (n.d.). URL: `https://linux.die.net/man/7/inotify`.

MINIX manpage on daemon command (n.d.). URL: `https://www.unix.com/man-page/minix/1/daemon/`.

OpenSSL EVP documentation (n.d.). URL: `https://www.openssl.org/docs/man1.1.1/man7/evp.html`.