STUDENTS, PLEASE COPY THIS PAGE AND USE AS THE COVER PAGE FOR YOUR SUBMISSION

Module No:	**COMP7024**	Module title:	**Operating Systems Security and Development**
Assessment title :			**Coursework 2**
Due date and time**:**			**28/04/2023 17:00**
Estimated total time to be spent on assignment:			35 hours per student

LEARNING OUTCOMES

On successful completion of this module, students will be able to achieve the module following learning outcomes (LOs): *LO numbers and text copied and pasted from the module descriptor*
LO 2: Create system-level software that modifies and extends existing operating systems. Conduct experiments designed to evaluate the performance, security and reliability of their modifications and additions.
LO 4: Demonstrate a thorough understanding of multi-threaded/process systems through the design and implementation of communicating, multi-threaded systems software.

Engineering Council AHEP4 LOs assessed (from S1 2022-23) *LOs copied and pasted from the AHEP4 matrix*	

STUDENT NAMES (ONLY IF GROUP ASSIGNMENT, OTHERWISE ANONYMOUS)

Student No:	**Student Name:**	**Group Name and Number:**
1.		

Statement of Compliance *(please tick to sign)*

X I declare that the work submitted is my own and that the work I submit is fully in accordance with the University regulations regarding assessments (*www.brookes.ac.uk/uniregulations/current*)

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

19129163 Mohammad Ali Khan

April 2023

1 Description

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 when they are stored to disk and when the user reads them respectively, to prevent them from being read by outsiders.

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**.

1.1 A Brief note on Daemons

A **Daemon** (sometimes claimed to be an acronym for*Disk And Execution MONitor*) is a 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 Unix daemons include **init**, **crond**, **httpd**, and **syncd**, all of which perform useful tasks in the operating system. As a *nix environment, Minix also has daemons, and also the **daemon** function which can be used to run scripts or processes as daemons.

2 Requirements

To build this daemon, we will need the following:

- Knowledge of the Minix OS and filesystem
- Knowledge of signals and signal handling for
- 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.
- a suitably strong and tested encryption algorithm/cipher.

Further details regarding the requirements for a cryptosystem will be discussed in the section concerning Encryption.

3 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 (especially one that is compliant with international standards) 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.), which meets several requirements for cipher strength and security.

AES stands for **Advanced Encryption Standard** and is a variant of the Rijndael algorithm. It is a symmetric block cipher for the encryption of electronic data developed by the United States National Institute of Standards and Technology in 2001 (Morris J. Dworkin n.d.).

it works by taking a plaintext and a key, performing an operation, and repeatedly performing a set of **substitutions** (analogous to a classical substitution cipher) and **permutations** (analogous to a classical transposition cipher) on them.

It is a very secure and widely used cipher, there are no known *computationally feasible* attacks for it- there is a known biclique attack for AES-128 (Andrey Bogdanov 2013), and a related-key attack for AES-256 (Alex Biryukov and Nikolić 2009) (Biryukov and Khovratovich 2009), but both are exceptionally complex.

4 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 can be hashed and stored in an external file instead of being hard-coded into the daemon, and the user can be asked to enter it at system startup.

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

inotify is a file change notification system in the Linux kernel, and it has a number of flags for certain events:

- IN_ACCESS – File was accessed
- IN_CLOSE_WRITE – File opened for writing was closed
- IN_CLOSE_NOWRITE – File not opened for writing was closed
- IN_CREATE – File/directory created in watched directory
- IN_MODIFY – File was modified
- IN_OPEN – File was opened

By detecting an IN_CLOSE _WRITE or _NOWRITE event, we can make the daemon read the data of the file closed, and encrypt it.

Upon detecting an IN_OPEN flag, the program can be instructed to decrypt the file being opened.

In addition to the program itself, there may be additional user-configured files for storing the credentials for the daemon, and perhaps also a **blacklist** for excluding files that the user potentially doesn't want encrypted (this may be done for performance reasons- while testing showed encryption/decryption takes a tiny amount of time, this may vary based on the user machine).

5 Development

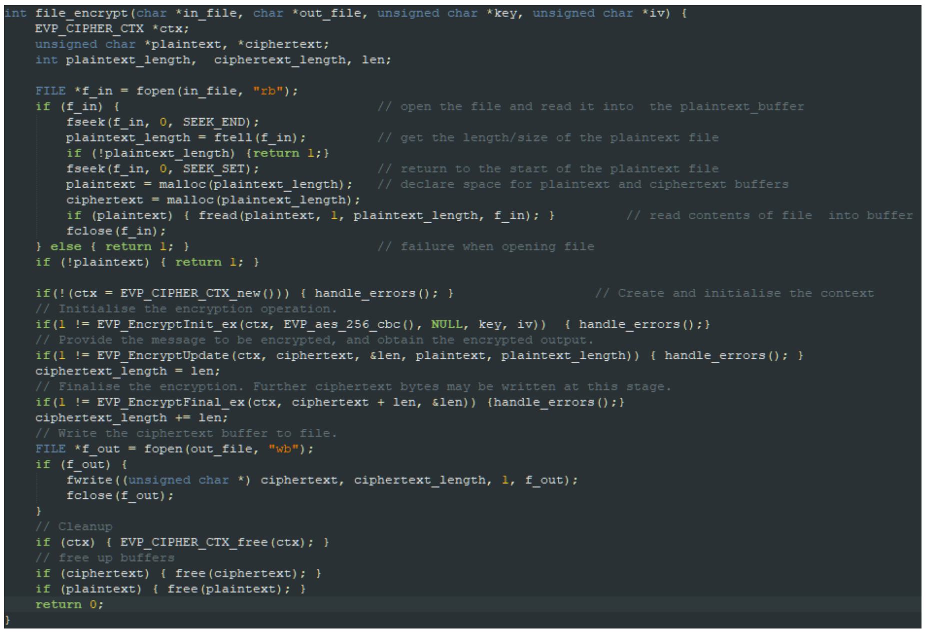
The program is first built as a standalone C program before attempting to integrate it with the Minix operating system.

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.

This standalone program simply took a file, encrypted it using a hardcoded key and IV (the final program will read the credentials from an external source).

5.1 A note on developing encryption/decryption functions

The encryption/decryption functions were 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.



```c
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

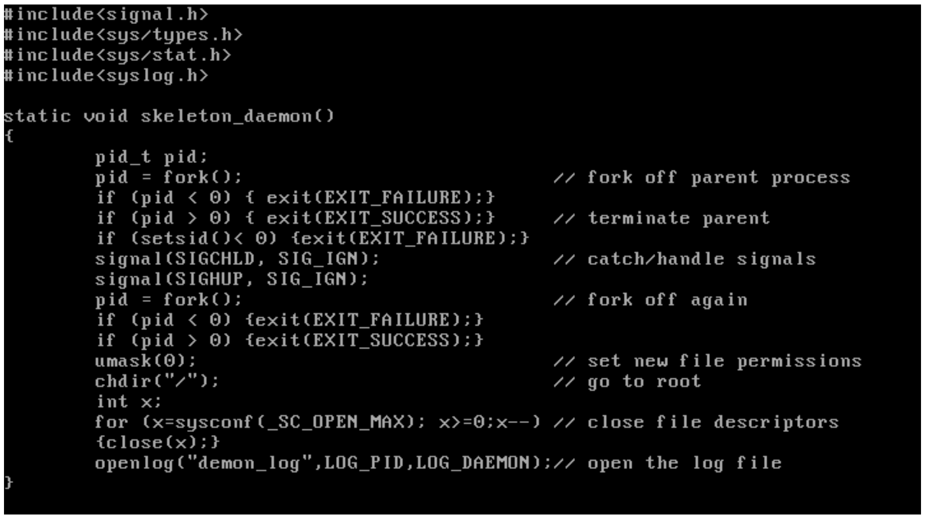
5.2 A note on Daemon development

the basic principle of how a daemon operates involves the following steps to set it up:

- 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.).



```c
#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
}
```

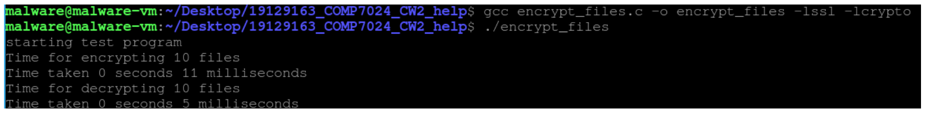
Figure 2: C function that provides a ”skeleton” for making a daemon

6 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.

We can use libraries like **time.h** to figure out the performance of the algorithm, how long it takes to encrypt and decrypt individual files vs. a large volume of files.



```
malware@malware-vm:~/Desktop/19129163_COMP7024_CW2_help$ gcc encrypt_files.c -o encrypt_files -lssl -lcrypto
malware@malware-vm:~/Desktop/19129163_COMP7024_CW2_help$ ./encrypt_files
starting test program
Time for encrypting 10 files
Time taken 0 seconds 11 milliseconds
Time for decrypting 10 files
Time taken 0 seconds 5 milliseconds
```

Figure 3: results of testing to see how long it takes to encrypt and decrypt multiple files

For better testing, these performance tests could be repeated on different hardware and virtual machine configurations to see how much faster or slower it can become.

Additionally tests can also be performed on the daemon's ability to detect and react to changes in the filesystem, whether there are any delays in between a file actually opening and the daemon logging that a file has been opened.

7 Integration Testing

Once testing of the individual sections and functions is complete, we can start putting them together and seeing how they work together, and how they work within the target system

Care must be taken to make backups of the system and the files on it in order to make sure recovery is possible if the program makes an error that damages core system files unusable and renders the system inoperable as a result.

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

Alex Biryukov, Dmitry Khovratovich and Ivica Nikolić (2009). *Distinguisher and Related-Key Attack on the Full AES-256 (Extended Version)*. URL: `https://eprint.iacr.org/2009/241`.

Biryukov, Alex and Dmitry Khovratovich (2009). *Related-key Cryptanalysis of the Full AES-192 and AES-256*. URL: `https://eprint.iacr.org/2009/317`.

Andrey Bogdanov Dmitry Khovratovich, Christian Rechberger (2013). *Biclique Cryptanalysis of the Full AES*. URL: `https://web.archive.org/web/20160306104007/http://research.microsoft.com/en-us/projects/cryptanalysis/aesbc.pdf`.

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/`.

Morris J. Dworkin Elaine B. Barker, James R. Nechvatal et al (n.d.). *Advanced Encryption Standard (AES)*. URL: `https://www.nist.gov/publications/advanced-encryption-standard-aes`.

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