Assignment 5

- You should have already installed and become familiar with the **Ubuntu Mate 20.04 Operating System(OS).** You can do this either:
  (1) with dual boot i.e. if you already have an OS such as MS Windows, MAC OS, or another linux distribution, just install Ubuntu Mate 20.04 as a secondary OS, or,
  (2) by installing first a Virtual machine (e.g. the Oracle VM VirtualBox is freely available) in your existing OS, and then within the VirtualBox installing the Ubuntu Mate 20.04, or,
  (3) by installing Ubuntu Mate 20.04 as your primary OS.
- If you prefer install the **Ubuntu 20.04 OS** instead of the Ubuntu Mate.
- You should have already installed the gcc compiler:

```
sudo apt update
sudo apt install build-essential
```

- Develop you C code using one of your favorite editors such as gedit, pluma or vi.

Implement a basic ransomware

For this assignment, you will develop a basic ransomware. It is a type of malicious software that is used to block access to the files of a user by encrypting them. Its main goal is to extort money from users in order to decrypt their files back. The main functionality of the ransomware is to encrypt a number of files and delete the original unencrypted files. Of course, a proper ransomware would be able to detect the access control logging system and bypass it. However, for the purposes of this assignment, you are required to provide only the basic functionality that we ask for.

Step 1: Implement the ransomware

More specifically, you should develop a bash script (named "`ransomware.sh`") that demonstrates the creation, encryption & deletion of a number of files inside a directory in a certain amount of time. More specifically, you should (1) create/select the files to be encrypted, (2) produce the new, encrypted files, and finally (3) delete the original files. **We strongly suggest you to create new text files that will then be encrypted. DO NOT RISK encrypting and deleting your existing files that you don't want to lose/corrupt.**

Your ransomware should also be able to create a big volume of files, so *given a directory as an argument*, it should create X[1] files in that directory. *X is also given as an argument*.

You can use the **OpenSSL** binaries for the encryption of the files. Two *aes-ecb* examples follow, make sure you use the correct **encryption function** as parameter and the correct **key**. In the following example the encryption password is "1234".

[1] X is an integer number that specifies the number of files your ransomware must create.

Encryption: `$ openssl enc -aes-256-ecb -in test.txt -out test.txt.encrypt -k 1234`
File deletion: `$ rm test.txt`
Decryption: `$ openssl aes-256-ecb -in test.txt.encrypt -out test.txt -d -k 1234`

For more information on the OpenSSL command line tool visit:

- https://linux.die.net/man/1/openssl
- https://wiki.openssl.org/index.php/Command_Line_Utilities

Step 2: Use the Access Control Logging tool from previous assignment[2]

Your ransomware will make use of the *shared library* you implemented in the previous assignment "Access Control Logging", named "`logger.so`". This means that every access type (create, open or write) will be logged with an entry in the log file named "`file_logging.log`". Therefore, the entries must have the same format as in the previous assignment:

1. **UID**: The unique user ID assigned by the system to a user (hint: see `getuid()` function).
2. **File name**: The path and name of the accessed file.
3. **Date**: The date that the action occurred.
4. **Timestamp**: The time that the action occurred.
5. **Access type**: For *file creation*, the access type is "0". For *file open*, the access type is "1". For *file write*, the access type is "2".
6. **Is-action-denied flag**: This field reports if the action was denied to the user with no access privileges. It is "1" if the action was denied to the user, or "0" otherwise.
7. **File fingerprint**: The digital fingerprint of the file the time the event occurred. This digital fingerprint is the hash value of the file contents (hint: You can use the md5 hash functions: https://www.openssl.org/docs/man1.1.0/man3/MD5_Init.html).

Events that must be logged:

1. **File creation:** Every time a user creates a file, the log file must be updated with information about the creation of the file. Make sure to modify the `fopen()` function in a way that the *creation* of a file can be distinguished from the *opening* of an existing file.
2. **File opening**: Every time a user tries to open a file, the log file must be updated with the corresponding file access attempt information. For this case, `fopen()` functions need to be intercepted and information about the user and the file access has to be collected.
3. **File modification (write)**: Every time a user tries to modify a file, the log file should be updated with the corresponding file modification attempt information. This means that `fwrite()` functions need to be intercepted and information about the user and the file access has to be collected. Every `fopen()/fwrite()` function should create a new entry in a log file.

[2] This step only requires to use the `logger.so` shared library from assignment 3.

Step 3: Detect the ransomware by enriching the Access Control Log Monitoring tool from Assignment 3

Enrich your Access Control Monitoring tool from the previous assignment, named "`acmonitor.c`", to have the following extra functionality. The Access Control Monitoring tool will detect the existence of your ransomware. More specifically:

1. In many cases, a ransomware tries to hide malicious files in directories populated by huge amounts of files. In this scenario, a ransom will create a big volume of files. You need to find if X files (at minimum) were created in the last 20 minutes.
2. A ransomware will also try to encrypt files and then discard the unencrypted version of those files. You need to find and report all the events in the log where a ransomware opened an unencrypted file and created an encrypted one. Remember that encrypted files have the suffix "`.encrypt`".

Tool Specification

The enriched Access Control Log Monitoring tool (Step 3) will receive the required arguments from the command line upon execution. Specifically, you should add the (1) `-v <number of files>` and the (2) `-e` options, and keep the `-m, -i, -h` options as-is from the previous assignment.

Options

`-m`	Prints malicious users
`-i <filename>`	Prints table of users that modified the file given and the number of modifications
`-v <number of files>`	Prints the total number of files created in the last 20 minutes
`-e`	Prints all the files that were encrypted by the ransomware
`-h`	Help message

Notes

1. If no appropriate option is given, the Access Control Monitoring tool has to print the corresponding error message.
2. You need to create a `Makefile` to compile your library and programs (you must submit it with your source code), or, you can use the one provided in the previous related assignment.
3. We do not provide you with a source code corpus; you have to enrich the source code of the previous related assignment.
4. You must submit the following files: `README`, `Makefile`, `logger.c`, `logger.so`, `acmonitor.c`, `ransomware.sh`
5. You need to submit all the source code of your tool, a **"Makefile"**, and a **"README.txt"** file that explains briefly your implementation and what you didn't implement and why. Place all these files in a folder named <yourlastnameAM>_assign5, and then compress it as a .zip file that you will upload to eclass. For example: christodoulou2018123456_assign5.zip.
6. The README.txt file is important to submit.
7. Very important: execute the command gcc –-version and write whatever the output is into your README.txt file, e.g. "gcc (Ubuntu 9.3.0-10ubuntu2~20.04)"
8. Your logfile should have a log entry for the creation of files e.g. the "`.encrypt`" files. But this cannot be implemented with `fopen`. Hint: search for `fopen64` (`o_fopen.c`, `bss_file.c` line 12). Try to implement the `fopen64` (it calls the `fopen`). You might also need to pass the flag **`-D_FILE_OFFSET_BITS=64`** into the `makefile` that creates the `logger.so`.
9. The entries in the logfile should be created from `fopen` and `fwrite`. Do not implement the log entries using bash function. What you can do is: from bash, call a new `test_aclog.c` file with `./test_aclog`, which in turn calls the `fopen` (therefore the action will be logged), in order to test the Steps 2 and 3. In your submitted files include also the `test_aclog.c`.
10. In the script for ransomware, as argument you pass the number of files; this refers to the number of encypted files that will be created into a directory. In the Access Control Log Monitoring, you are also giving as argument the number of files; in this case you should check whether at least `<number of files>` have been produced within the last 20 minutes, and then print the number of files that have been produced within this time period, either with suffix "`.encypt`" or without. The idea behind this is: if in a system a large volume of files be created within a short time this will be considered suspicious and might correspond to a malicious case. In your implementation make sure that you print the total number of files created within the last 20 minutes; if this number is smaller than the `<number of files>`, then this does not indicate a suspicious behavior.
11. As arguments should be the name of a directory, and the number of files that will be created with your script. We repeat: do not use your existing files because you might corrupt them.
12. Your ransomware should have 2 operations: one is encrypting X files, and the other is creating a large volume of files. You can combine them i.e. creating X encrypted files within 20 minutes, but also separate these operations so as each of them will run individually. To implement selecting one of these 2 operations you can do it either via arguments, or, by calling the

ransomware and selecting the corresponding operation from a menu. Choose whatever implementation you wish, and describe this briefly in your README.txt.

13. Your ransomware could also support decryption operation. For example, add an argument in your `ransomware.sh` that will be indicating the operation to run, i.e. creation of files and then encryption, or, decryption of files. Describe this briefly in you README.txt.
14. For the encryption/decryption use whatever algorithm you want. Describe this in your README.txt.