HY-457

Assignment 2

Due: 21/04/2017

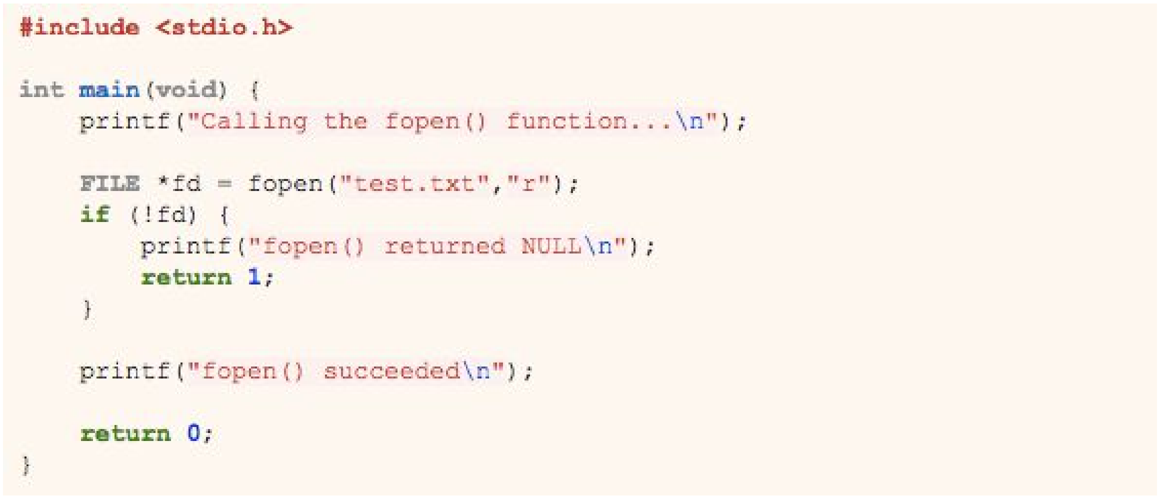
In this assignment you are going to develop an access control logging system in C. The system will keep track of all file accesses and modifications. More specifically, every file operation (such as open or modification) will generate an entry in a log file, stored for further investigation by a separate high privileged process.

For this assignment you will need to use LD_PRELOAD which gives the ability to instruct the linker to bind symbols provided by a certain shared library before other libraries. Using LD_PRELOAD you are going to override the C standard library's functions that handle file accesses and modifications (fopen, open, write, fwrite) with your own version in order to extend their functionality.

Using LD_PRELOAD

In the first task you will use LD_PRELOAD in order to add extra functionality to (fopen, open, write, fwrite).
In order to demonstrate the use of LD_PRELOAD we need a simple example.
Let’s start with a simple C program (prog.c)

prog.c



```c
#include <stdio.h>

int main(void) {
    printf("Calling the fopen() function...\n");

    FILE *fd = fopen("test.txt","r");
    if (!fd) {
        printf("fopen() returned NULL\n");
        return 1;
    }

    printf("fopen() succeeded\n");

    return 0;
}
```

The code above simply makes a call to the standard fopen function and then checks its return value. By compiling and executing it we get the following results.

```
$ ls
prog.c  test.txt

$ gcc prog.c -o prog

$ ls
prog  prog.c  test.txt

$ ./prog
Calling the fopen() function...
fopen() succeeded
```

Now let’s write a shared library called myfopen.c that overrides fopen in prog.c and calls the original fopen from the c standard library.

myfopen.c

```
#define _GNU_SOURCE

#include <stdio.h>
#include <dlfcn.h>

FILE *fopen(const char *path, const char *mode) {
    printf("In our own fopen, opening %s\n", path);

    FILE *(*original_fopen)(const char*, const char*);
    original_fopen = dlsym(RTLD_NEXT, "fopen");
    return (*original_fopen)(path, mode);
}
```

This shared library exports the fopen function that prints the path and then uses **dlsym** with the **RTLD_NEXT** pseudohandle to find the original fopen function. We must define the **_GNU_SOURCE** feature test macro in order to get the **RTLD_NEXT** definition from <dlfcn.h>. **RTLD_NEXT** finds the next occurrence of a function in the search order after the current library.

We can compile the shared library this way:

```
gcc -Wall -fPIC -shared -o myfopen.so myfopen.c -ldl
```

Now by using LD_PRELOAD we can run the executable with our own shared library.

```
$ LD_PRELOAD=./myfopen.so ./prog
Calling the fopen() function...
In our own fopen, opening test.txt
fopen() succeeded
```

Task A

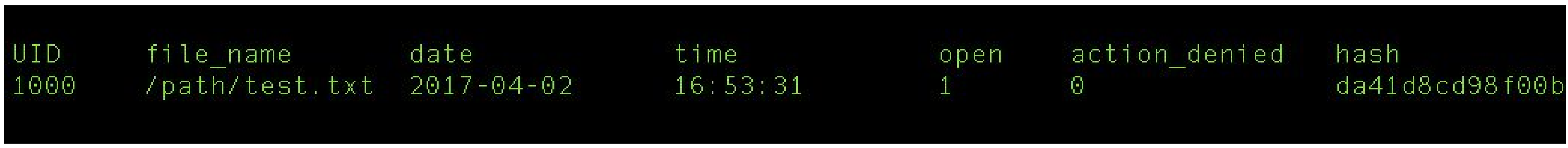
[Extending the Standard Library]

For the case of our exercise, information about the file accesses and modifications should be stored in a log file. In order to achieve this functionality, some functions of the C standard library should be intercepted using LD_PRELOAD:

1. Every time a user opens or tries to open a file, the log file will be updated with information regarding the file access. For this case **open** and **fopen** functions need to be intercepted and information about the user and the file access has to be collected.
2. Every time a user modifies or tries to modify a file, the log file will be updated with information regarding the file modification. For this case **write** and **fwrite** functions need to be intercepted and information about the user and the file access has to be collected.

The log file should provide read and write privilege only to a separate high privileged process, that you will create in **Task B**. The intercepted functions, previously mentioned, that need to add an entry in the log file, have to open it in append mode.

The intercepted **open, fopen, write** and **fwrite** functions should create a new entry in the log file. Each entry has to follow the following format:

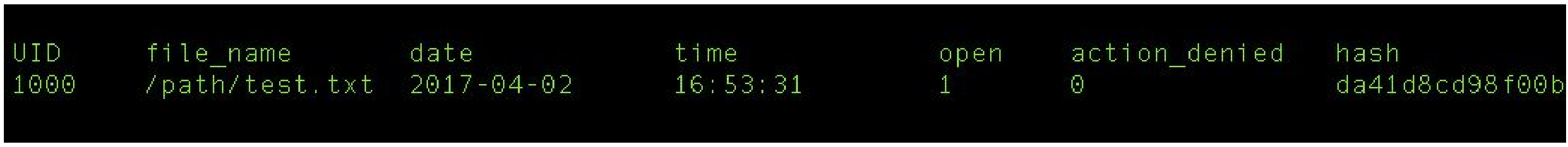


```
UID     file_name        date            time            open    action_denied   hash
1000    /path/test.txt   2017-04-02      16:53:31        1       0               da41d8cd98f00b
```

- **UID**
    - unique positive integer assigned to the user by the system
- **file name**
    - the path and name of the accessed file
- **date**
    - the date this action occurred
- **time**
    - the time this action occurred
- **open**
    - this field describes whether the corresponding file was opened or modified. It has to be 1 if the action performed to this file was an **open** or **fopen** and or 0 if it was a **write, fwrite**
- **action_denied**
    - this field reports the case at which the user was denied an action (**open, fopen, write, fwrite)** to a file, due to lack of the corresponding access privilege. It has to be 1 if the user tried to open or write a file and did not have permission, or 0 in every other case

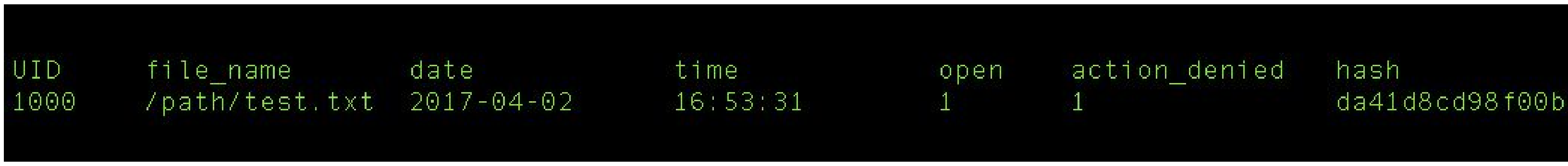
- **Hash**
    - this field contains the hash value of the file at the time this event occurred. The hash value should be created by the contents of the file. In order to generate the hash value of the file you must use any of the already known hash functions used in order to check the integrity of files (MD5, SHA1**,..**). Feel free to use existing implementations and tools already in your system for the hash function.

For example, following the above format, for a successful **open** or **fopen** a log entry looking like this should be added:



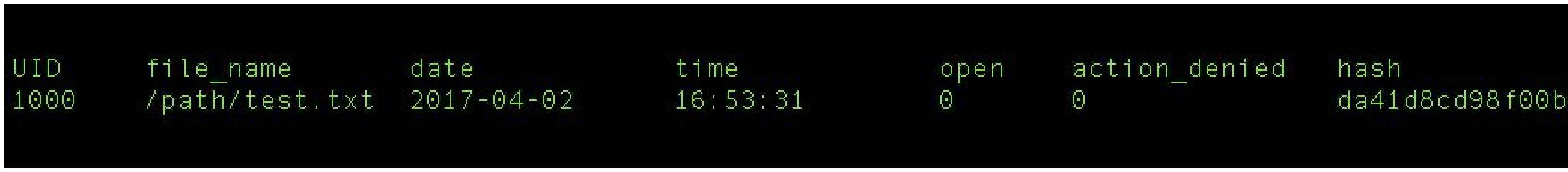
```
UID     file_name        date             time             open     action_denied    hash
1000    /path/test.txt   2017-04-02       16:53:31         1        0                da41d8cd98f00b
```

On the other hand, an unsuccessful **open** or **fopen** should create a log entry looking like this:



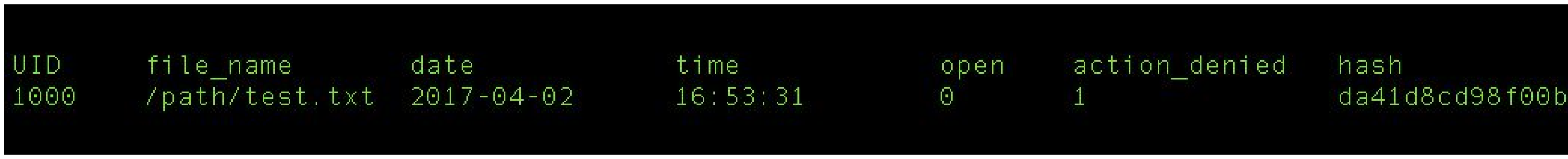
```
UID     file_name        date             time             open     action_denied    hash
1000    /path/test.txt   2017-04-02       16:53:31         1        1                da41d8cd98f00b
```

A successful **write** or **fwrite** should add an entry looking like the following:



```
UID     file_name        date             time             open     action_denied    hash
1000    /path/test.txt   2017-04-02       16:53:31         0        0                da41d8cd98f00b
```

And an unsuccessful **write** or **fwrite** should add an entry looking like this:



```
UID     file_name        date             time             open     action_denied    hash
1000    /path/test.txt   2017-04-02       16:53:31         0        1                da41d8cd98f00b
```

Task B

[Log Monitoring]

In this task you will implement a separate process, responsible for monitoring the logs, created by the extended functions of **Task A,** and able to extract certain information. This high privileged process will perform the following operations.

1. The process will parse the logs generated by the intercepted functions of **Task A** and detect the users trying to access files that do not have permission to **open** or **write**.
2. For a given file (path and filename) as input, the process must track all the users that have accessed that certain file. By comparing whether the hash of the file has changed it should be able to find the users that modified it.
3. The process must be able to find and report all the users that have **unsuccessfully** tried to access more than 10 different files in less than 24 hours.

Tool Specifications

For Task A:

When executing a program using LD_PRELOAD with your shared library, a file action (open, fopen, write, fwrite) should create a log entry in the log file **following the exact described format**. The logs must be stored somewhere where they can be accessible by all the users.

For Task B:

Your program should take two arguments, the path and filename of your logs created in **Task A** and a number ranging from 1 to 3 which corresponds to the 3 functionalities of **Task B**. For functionality number 2, another argument is required which will be the path and filename of the file you want to query about.

Example:

```
$ ./Task_B  /path/logs.txt  1
$ ./Task_B  /path/logs.txt  2  /path/example.txt
$ ./Task_B  /path/logs.txt  3
```

For all 3 queries the output should be a list of users. In functionality 2, a second letter will be needed next to the UID symbolizing A (access) or M (Modified).

Example:

```
1000 A
1000 M
```

Submit

A folder named assign_2_yourAM, containing all the source code of your implementation, a README file and a Makefile.

IMPORTANT

- You have to support the command line options described above.
- Use the list for questions and **not** private messages to the TAs.