CSI4109 Assignment #2
Due: Mar 29th 12:59pm

Mandatory Access Control (MAC)

In this homework assignment, you will implement the *read-down, write-up* mandatory access control policy discussed in class. Specifically, you will write a command-line tool called `mac`, which, together with `setuid` / `setgid` functionalities in Linux, is used as a mechanism to implement this policy.

- This command line tool `mac` will be owned by the user `root` and the group `root`, and be given the permissions: `6755`, where the `setuid`, `setgid`, and executable bit are set as follows.

```
-rwsr-sr-x 1 root root 18K Mar 14 21:04 mac
```

- **Objects:** There are four documents (a document for each security classification level) that are protected by the mandatory access control policy: `top_secret.data`, `secret.data`, `confidential.data`, and `unclassified.data`. These four files are classified as `TOP_SECRET`, `SECRET`, `CONFIDENTIAL`, and `UNCLASSIFIED`, respectively. These four files will be (i) pre-created, (ii) owned by the user `root` and the group `root`, and (iii) given the permissions: `0640` as follows.

```
-rw-r----- 1 root root 20 Mar 14 16:01 secret.data
-rw-r----- 1 root root 24 Mar 14 21:15 top_secret.data
-rw-r----- 1 root root 26 Mar 14 21:15 unclassified.data
-rw-r----- 1 root root 26 Mar 14 21:15 confidential.data
```

  Note that the owners and permissions of these files must stay the same at all times.

- **Subjects:** All *non-root* users in the system are considered as subjects. Security clearance levels of subjects are specified in a file called `mac.policy`, which is owned by the user `root` and the group `root`, and given the permissions: `0640` as follows.

```
-rw-r----- 1 root     root      23 Mar 14 21:17 mac.policy
```

  `mac.policy` will be pre-created and written in the following format:

```
<user_1>:<security clearance level>
<user_2>:<security clearance level>
```

Remember that subjects whose access is controlled by our policy are *non-root* users; that is, the command line tool `mac` will be executed by a non-*root* user whose primary group is the same as the name of that user. (e.g., the user `david` and the group `david`) This effectively prevents subjects from reading or manipulating the data files directly; this requires accesses to the data files be always mediated by the program `mac`.

- **Dropping the root privilege:** During the execution of `mac`, drop the `root` privilege when you do not need it anymore, by changing the effective `uid` and `gid` of the process. (Hint: Use the following functions: `seteuid`, `getuid`, `setegid`, and `getgid`.)

- **Logging:** After dropping your root privilege, **append** the command line arguments followed by a newline character to the per-user log file named `<username>.log`. Do not log, however, what has been writen.

```
$ ./mac read secret.data
$ cat david.log
read secret.data
$ ./mac write secret.data TEST_INPUT
$ cat david.log
read secret.data
write secret.data
```

If the log file doesn't exist for the given user, it must be created before appending. Once created, this file must **at all times** be owned by the executing non-`root` user and group, and be given the permissions: `0640`. You can assume that the user has the permission to create a file in its current working directory. (Hint: Use the following functions: `getpwuid` and `umask`.)

Interface. All variable inputs to the program (e.g., `<document file>`) will be `[a-zA-Z0-9_\-\.]` (i.e., alphanumeric, underscore, dash, and period). All matching is case-sensitive.

You must implement the following command-line interface:

- `./mac read <document file>`

reads and prints the content of the document file (e.g., `top_secret.data`) followed by a newline character on standard output, if and only if (i) the security clearance level for the executing user is specified in `mac.policy`, and (ii) our *read-down* confidentiality policy allows it. Otherwise, print `ACCESS DENIED`, followed by a newline character. You can assume that `<document file>` is always valid (one of the four files specified above).

- `./mac write <document file> <data>`

appends `<data>` followed by a newline character to the file `<document file>`, if and only if the executing user has a security clearance level, and the policy allows it. Otherwise, print `ACCESS DENIED`, followed by a newilne character. You can assume that a given `<document file>` (one of the four given above) is always valid.

Example. With `david:SECRET` written in the `mac.policy` file:

```
$ whoami
david
$ sudo cat mac.policy
david:SECRET
```

Running the program as follows:

```
./mac read secret.data
```

- will produce the following output on standard output:

  ```
  THIS IS SECRET DATA
  ```

- `./david.log` looks like:

  ```
  read secret.data
  ```

And running the program again as follows:

```
./mac read confidential.data
```

- will produce the following output on standard output:

  ```
  THIS IS CONFIDENTIAL DATA
  ```

- `./david.log` now looks like:

  ```
  read secret.data
  read confidential.data
  ```

And running the program again as follows:

```
./mac read top_secret.data
```

- will produce the following output on standard output:

```
ACCESS DENIED
```

- `./david.log` now looks like:

```
read secret.data
read confidential.data
read top_secret.data
```

Implementation.

- **Your program must work on Ubuntu 22.04 64-bit with the default packages installed.** In addition to the default packages, the following packages for languages are also installed:
  - C (`gcc`)
  - Rust and Cargo (`rustc` and `cargo`) with selection of crates pre-installed for you in the grading environment. If you are using Rust, provided `Cargo.toml` should not be modified.

  Again, you're probably better off if you set up a virtual machine to work on the course assignments early on. You can use VirtualBox, a free and open-source VM monitor software. Or, if you are using MS Windows, you may want to use WSL (WSL version 2 is recommended.) (Ubuntu 22.04 on Microsoft Store).

Submission Instructions. Submit on LearnUs (`ys.learnus.org`) your source code, along with a `Makefile` and `README`. When the command `make` is run, the `Makefile` must create your executable, called `mac`, on the same directory as your `Makefile` and `README`. These files must **not** be included in any subdirectory. Note that we may invoke `make` multiple times, and it needs to work every single time. After creating `mac`, we will change its owner and owning group, as well as permissions, as specified above. We will also create the data files as well as the policy file, and change their owner and owning group and permissions as specified above. Your `README` file must be **plain text** and should contain your name, student ID, and a description of how your program works. Your submission can be zipped, if required; your submission will be unzipped once before grading. However, the directory structure described above still apply.

Grading Rubric

- All required files exist, and `make` successfully creates an executable file `mac`. (2 pts)
- Produces a log file in a correct format. (1 pt)
- The log file's owner/group owner/permissions are correct at all times. (2 pts)
- Read accesses are correctly controlled. (3 pts)
- Write accesses are correctly controlled. (2 pts)

Note: It is your responsibility to comply with the specification as well as our grading environment. You can request regrading for minor problems (e.g., issues caused by using different environments), but they will result in deductions of 1 or more points.

Late policy: 1 pt deduction for every 3 hours that is late. This means that late submission up to 3 hours get one point deduction, 6 hours two point deduction, and so on.