CSI4109 Assignment #4
Due: May 22th 1:59pm

Sandboxing Third-Party Libraries

In this homework assignment you will learn how to *sandbox* your program. More specifically, you will learn how to mitigate the consequences of using libraries that you cannot trust.

Building a multi-user image classifier service. Imagine that you are developing an image classifier service, which, (i) takes an image filepath and an integer k ($k > 0$) specified by a user as input, and (ii) outputs top-k likely labels of the image. For example, given a filepath to an eagle image `eagle.jpg` and $k = 3$ as input, your service outputs likely top-3 labels on `stdout`, as follows.

```
87.09%: bald eagle
11.46%: kite
 0.53%: hen
```

You want this image classification service to serve multiple users. First, we assume that the image files belonging to a user is stored in a per-user subdirectory of the `data` directory, as follows.

```
$ find data
data
data/chris
data/chris/giraffe.jpg
data/chris/dog.jpg
data/kyle
data/kyle/horses.jpg
data/kyle/eagle.jpg
```

Then, you will write a program called `secure_classifier`, which emulates the image classification service. This program reads a file named `requests.txt` residing in the current working directory, which contains multiple lines of image classification requests. The format of this file goes like: `<username>:<relative filepath>:<k>`, and an example `requests.txt` file is given below.

```
$ cat requests.txt  # contains two requests
chris:dog.jpg:2
kyle:eagle.jpg:3
```

The `secure_classifier` program, when executed, (i) reads each line of `requests.txt`, (ii) serves the classification request specified in each line, and (iii) appends the top-k image classification result in a file named `results.txt` (in the same working directory). This process continues until it serves all requests specified in the `requests.txt` file. The following shows an `results.txt` file, generated as a result of executing `secure_classifier`.

```
$ ./secure_classifier  # this line executes your program creating results.txt
$ cat results.txt
[chris:dog.jpg:2]
12.50%: miniature schnauzer
12.40%: malamute
[kyle:eagle.jpg:3]
87.09%: bald eagle
11.46%: kite
 0.53%: hen
```

A seemingly perfect but non-trustworthy library. To facilitate the development of `secure_classifier`, you started to search for existing libraries, because it would be cumbersome and also error-prone to write code from scratch, which decodes a given image, executes an image classifier such as deep neural networks, etc. Fortunately, you found a perfect library called `libdarknet_predict.so`, which does the exact task (i.e., image decoding and classification) that you want. Unfortunately, however, you do not know where this library originated from, and there was no source code that you can review to ensure whether it is benign. In other words, *you found no reason to trust this library*.

This library can be invoked by calling a function called `image_classifier` as follows:

```
image_classifier(filepath, top_k);
```

The first argument specifies a filepath string to the image file (e.g., `eagle.jpg`), and the second argument specifies k in top-k. This library function call, when invoked, writes the top-k classification results to `stdout`.

The library file itself (i.e., `libdarknet_predict.so`) and the source code of an example program that invokes this library function have been distributed together with this specification, in all programming languages (e.g, C and Rust) supported.

Sandboxing your image classifier service. Even if you do not trust this library, you still want to use it to implement `secure_classifier`. Of course, you, as a security-savvy student who is taking CSI4109, decided to sandbox your program in order to contain damages that could potentially be caused by using the library. The **sandboxing policies** you want to enforce are:

- **Integrity:** Your program `secure_classifier` can only append the result of each image classification to the output file called `results.txt`; it must not be able to cause any other harm to the integrity of the rest of the system.
- **Confidentiality:** Your program `secure_classifier` must not leak the image specified in the request in any way. They must not be disclosed to, for example, other users directories, or remote adversaries through network.

There should be multiple ways to enforce this policy. You can use any set of isolation and sandboxing mechanisms (including those that you learned in class), as you would like, as long as your program thwarts attacks specified in the grading rubric at the end. Our recommendation, however, is **developing process-level sandboxes offered by Linux namespaces**. A good starting place would be to read the `namespaces` page of Linux Programmer's Manual, which can easily be pulled up on your terminal via `man 7 namespaces`.

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 compiling C/Rust programs, and for using the OpenSSL library) are also installed:

- C (`gcc`)
- Rust and Cargo (`rustc` and `cargo`) with a set of crates (and their re-exports) pre-installed for you in the grading environment. If you are using Rust, you must use the provided `Cargo.toml` file without any modification.
- Seccomp library (`libseccomp-dev`)

You'll probably need to set up a virtual machine to do your development. VirtualBox is a free and open-source VM system. 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 `secure_classifier`, 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 `secure_classifier`, we will place all the following files under the same directory ourselves for grading (meaning that you do not have to submit these):

- All `data/<user>/*.jpg` files
- `imagenet.shortnames.list`
- `darknet.cfg`
- `darknet.weights`
- `libdarknet_predict.so` (actually, we will use a *rogue* `libdarknet_predict.so` when grading your submission to test resiliency against various attacks.)

Your `README` file must be **plain text, without file extensions** and should contain your name, student ID, and a description of how your program works. Your submission can be zipped; your submission will be unzipped once before grading. However, the directory structure described above still apply to the unzipped files. In other words, structure your files and directories as below if you are submitting a zipped file.

```
submission.zip
    |-- Makefile
    |-- README
```

Grading Rubric

- All required files exist, and `make` successfully creates an executable file `secure_classifier`. (2 pts)
- Thwarts attempts to access the host's file system. (1 pts)
- Thwarts attempts to corrupt previous classification results in `results.txt`. (1 pts)
- Thwarts attempts to leak an input image through network after a dot-dot attack to the sandbox. (2 pts)
- Thwarts attempts to leak an input image of one user to any other user after a dot-dot attack to the sandbox. (2 pts)
- Thwarts attempts to corrupt previous classification results in `results.txt` after a dot-dot attack to the sandbox. (2 pts)
- You can assume that all necessary files exist, and their contents are in a valid format. You can solely focus on defending against the attacks described in earlier items.
- Note that, if you do not use `image_classifier` defined in the given `libdarknet_predict.so`, you will fail this homework.