CS 453/698 Assignment 3

TA: Ruizhe Wang (ruizhe.wang@uwaterloo.ca)
Office hours: Wednesdays 1pm to 2pm, online via BBB (access code 6gx9wq)

Due date: March 14th, 2025

Escaping `seccomp`-based Sandboxes

Environment preparation

We recommend using the ugster VM for this assignment. The `portal.sh` script, as usual, can be found in `http://ugster72d.student.cs.uwaterloo.ca:8000`.

- If you have already registered on ugster for A2, you can re-use your current VM. If you would like to reset the VM to a clean state for the new assignment, please reset your VM by `./portal.sh destroy` and then `./portal.sh launch`. However, be careful that this will completely wipe-out your data, so if you are still working on A2 or you'd like to save your work on A2, please do so before wiping out the machine.
- If you have not registered on ugster for A2, please register on the ugster platform with a new pair of keys. This can be done with `./portal.sh register`.
- In either case, please take care of your private key! if you lose your private key after successful registration, you will be locked out of the system completely. There is no way back. You can request a key reset on Piazza, and every reset means a **1 pt** deduction on the whole assignment.

After you register to the `ugster` platform and have your VM launched, please SSH into your VM and run the following command from the directory `/home/vagrant` to setup the VM for A3:

```
wget http://ugster72d.student.cs.uwaterloo.ca/a3/a3_setup.sh && \
   chmod +x a3_setup.sh && ./a3_setup.sh
```

This will create four binary executable files `sandbox[1-4]` and a text file `flag` in `/home/vagrant`.

Tasks

We have provided four binary executable files: `sandbox1`, `sandbox2`, `sandbox3`, and `sandbox4`. Each of these programs implements a `seccomp`-based sandbox and employs some distinct filtering rules. The programs are designed to execute external programs `sploit1`, `sploit2`, `sploit3`, and `sploit4` respectively. Additionally, we have provided a `flag` file that stores a flag you required to get.

Your task is to write four exploit programs in `C` language, namely `sploit1.c`, `sploit2.c`, `sploit3.c`, and `sploit4.c`, and compile them into four binary executable files `sploit1`, `sploit2`, `sploit3`, and `sploit4`, respectively, in such a way that when running the sandbox programs, the contents of the flag file are written into standard output. You are expected to submit the four source code files of your exploit programs with a `Makefile`, such that we can get your compiled binary executable files by simply typing `make` in the terminal.

Below is an example demonstrating how to test your code and run the sandbox programs:

```
// compile your exploit programs
// (should produces sploit[1-4])
$ make

// run the sandbox programs
// (your exploit programs should be in the same folder as the sandbox programs)
$ ./sandbox1
the flag should be written to standard output and displayed here

$ ./sandbox2
the flag should be written to standard output and displayed here

$ ./sandbox3
the flag should be written to standard output and displayed here

$ ./sandbox4
the flag should be written to standard output and displayed here
```

NOTE: When grading your submissions, we will replace the contents of the flag file. Therefore, **you should NOT hardcode the contents of the flag file in your program**.

Each `sploit` carries an equal weight of **10 pts**, making a total of 40 pts.

Tips to make your life a bit easier

- It might be a good idea to refresh your knowledge about system calls from CS 350 or whatever operating system courses you have taken.
- strace is a handy utility which will tell you the system calls invoked by a process.
- While reading the assembly can be a way to extract the seccomp-bpf rules enforced in each sandbox, it is not recommended. There are other ways of extracting the rules enforced.

Grading platform

We also open the grading platform for this assignment to you, which can be accessed through this link. You can find detailed instructions on the landing page. NOTE: you will need to use the campus VPN if you can't access it from outside.

In short (and see the landing page for details),

- to submit a package to the server, ZIP the package directory, send a HTTP `POST` request to `http://ugster72c.student.cs.uwaterloo.ca:9000/submit` with the ZIP content as body. You will get a hash string as the package identifier.
- to check the analysis result (after the server has processed the package), send a HTTP `GET` request to `http://ugster72c.student.cs.uwaterloo.ca:9000/status/<hash>` where the `<hash>` is the hash value you get from the package submission phase.

Some important things to note regarding the submission system:

- Submitting to the evaluation platform DOES NOT count towards assignment submission. To make the final submission, please follow the instructions in the Assignment instruction file and make the final submission on LEARN. We DO NOT accept package hashes as proof-of submission.
- This evaluation server is NOT well tested, meaning, there might be bugs. If you observe weird behaviors, please make a Piazza post and we will try to investigate as soon as possible.

Deliverables

All assignment submissions take place on LEARN Dropbox. Only the **most recent on-time submission** will be considered for evaluation.

You are required to hand in a single compressed `.zip` file that contains the following:

- **sploit1.c**, **sploit2.c**, **sploit3.c**, **sploit4.c**: The exploit programs.
- **Makefile**: A Makefile that compiles all your exploit programs.
- **write-up.pdf (optional)**: An optional write-up in PDF format (see below).

Your submitted `.zip` file should contain and only contain the above files. It must be acceptable to the submission server. Submitting invalid files, such as `SPLOIT1.c`, `write-up.md`, and `__MACOSX`, would result in a **deduction** on the whole assignment.

Write-up. We use an auto-grader to check the code submitted for this assignment. While efficient, the auto-grader can only provide a binary pass/fail result, which rules out the possibility of awarding partial marks for each task. As a result, we also solicit a write-up submissions.

The write-up can include information about any of the exploits you attempted as long as you believe the information is relevant for the grading. Typical things to be put in the write-up include:

- How the code you submitted is expected to work
- How you manage to get certain hardcoded information in the code
- Explanation on critical steps / algorithms in the code
- Any special situations the TAs need to be aware when running the code
- If you do not complete the full task, how far you have explored

On the other hand, if you are confident that all your code will work out of the box and can tolerate a zero score for any tasks on which the auto-grader fails to execute your code, you do not need to submit a write-up (or you can omit certain tasks in the write-up).