CS 453/698 Assignment 3

TA: Ruizhe Wang (ruizhe.wang@uwaterloo.ca)
Office hours: Wednesdays 2pm to 3pm, online via BBB (see access code on Piazza)

Due date: Jul 11st, 2025

Escaping seccomp-based Sandboxes

Secure Computing Mode (seccomp) is a kernel feature that allows you to filter system calls to the kernel from a container. Should it attempt to call blocked system calls, the kernel will either log the event or terminate the process with SIGKILL or SIGSYS. It provides a fine-grained control and gives an attacker a limited number of syscalls from the container[1][2].

In this assignment, our goal is to escape seccomp-based sandboxes and obtain secret messages.

Environment preparation

We recommend using the ugster VM for this assignment. Instructions, as usual, can be found in `http://virtus.cs453.dpdns.org`.

- If you have already used the ugster VM 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 using `reset` and then `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.
- Please take care of your private key and frequently backup your files! if you lose your private key or mangle the `sshd` service within the VM, you will be completely locked out of the system. There is no way back. However, we do not expect that any steps of finishing A3 would break the SSH service.

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/mission/a3/a3_setup.sh && \
    chmod +x a3_setup.sh && ./a3_setup.sh
```

This will create five binary executable files `sandbox[1-5]` and a text file `flag` in `/home/vagrant`.

[1]https://en.wikipedia.org/wiki/Seccomp
[2]https://docs.redhat.com/en/documentation/red_hat_enterprise_linux_atomic_host/7/html/container_security_guide

Tasks

We have provided five binary executable files: `sandbox1`, `sandbox2`, `sandbox3`, `sandbox4`, and `sandbox5`. 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`, `sploit4`, and `sploit5` respectively. Additionally, we have provided a `flag` file that stores a flag you required to get.

Your task is to write five exploit programs in `C` language, namely `sploit1.c`, `sploit2.c`, `sploit3.c`, `sploit4.c`, and `sploit5.c`, and compile them into five binary executable files `sploit1`, `sploit2`, `sploit3`, `sploit4`, and `sploit5`, respectively, in such a way that when running the sandbox programs, a secret message will be written into standard output. The message should be identical to the content of the flag file for `sploit[1-4]` and its secret buffer for `sploit5`. You are expected to submit the five 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.

The `Makefile` should not contain logic other than generating `sploit[1-5]`. Generating and executing the exploit programs should also not requiring any additional packages other than what we have provided in `a3_setup.sh`. All commands, including both `make` and `sandbox*` while be executed in the path `/home/vagrant`.

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

```
// compile your exploit programs
// should produces sploit[1-5]
// should not generate any other files or spawn any other processes
// make should terminate before running any sandbox
$ 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

$ ./sandbox5
sandbox5's secret buffer 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**. We will only replace several characters.

Each `sploit` carries an equal weight of **10 pts**, making a total of 50 pts. Unless you choose to submit written descriptions as explained below, you will be awarded marks on an all-or-nothing basis for each of the exploits (i.e., **a sploit that fails to reveal the secret message will be given 0 marks**).

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, **there are other ways to extract the enforced rules.**
- You are free to use online tools to gather information and assist you in completing this assignment. However, please document the tools you have used (either in `sploit*.c` or the writeup).

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**, **sploit5.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. If you are using a MacBook, please refrain from using Finder-provided zip functionality. It is highly recommended to submit the same zip you submitted to the evaluation platform to Learn.

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).