OS Course Project
Software sandbox using syscall prevention aproach

Yuyang Rong
School of Information Science and Technology
Shanghaitech University
Student ID: 69850764

Jianxiong Cai
School of Information Science and Technology
Shanghaitech University
Student ID: xxxxxxxxx

Abstract—**How to grade a students programming homework has become a tricky problem since the grading server have to protect itself from being hacked by student's homework, and to protect test case from being poached by students. To be general, a safe environment is needed for user to run untrusted program where user and his data can be safe. We propose that we track every syscall the untrusted program issues, thus we can trace it's movement and status and make sure it remain trustworthy.**

1. Motivation

1.1. Real life problem

From gradebot's point of view, student's homework should be considered untrustworthy. However, in order to do the grading job, the server have to run all students programs.

Currently there are many malware isolation applications, one of the most famous being lxd. We dived into lxd and realized that it's designed based on the following principles:

If you can't see it, you can't destroy it. You can't tempering with things you can't see.

Good as this principle is, it causes unintended inconvenience. For example, Rust compiler have to check github.com for extern crates used by students program. But to make sure the data is safe, network access has been baned by lxd. Such situation can be really problematic.

Thus we realized we need something better to help cope with the situation. We want network, file system, etc be available without damaging user's computer or server. That's how we started our project.

1.2. Principles

So we came to our solution that we will show the program the whole picture, it can access all file system or network, but we can't allow everything, so we have the following philosophy:

If you have to ask for my permission for every syscall you make, you can't damage things because I will not allow it.

1.3. Difficulties

Nice as tracing every syscall sounds, it's hard as soon as we realized the real scale the problem is.

1.3.1. Outnumbered. The number of syscall are enormous, far beyond our expectation. For example, in x86_64 architecture, there are in total 328 syscalls. At first we though we have to write 328 different policies to deal with these syscalls, but later we realized that there is a connection between these syscalls. Certain syscalls have to be made before others. For example, each and every program have to *open()* a file before it can read or write it. Or, *socket()* have to be established before a network connection can be possible.

So we decided to chock the untrusted program on those syscalls, to put pressure on where it hurts. For example, we allow *close()* , *read()* directly, but we carefully examine *open()* , whether the file is allowed, what it is opened for, etc.

In this way, we narrowed our work to much fewer syscalls like *open()* and *connect()*

1.3.2. Killing a children?. Not much program nowadays do not do *fork()* anymore, even students' homework use this syscall here or there. When a program *fork()* , tracing programming do *fork()* too, and immediately begin to trace the child just forked by the untrusted program.

The problem is, When a untrusted program *kill()* it's child, or one of it's children *kill()* another, the tracer can't tell the difference. Tracer have no way to know the *pid* about to be killed is another user program or one of one of tracee's children. Tracer program have to do inter communication to know what really happened.

Doing inter communication between tracers can be rather difficult can gains little, we then came up with a new solution that we change tracee's *uid*, thus every tracee's child will have the same *uid* and they can feel free to *kill()* each other, but once they try to *kill()* anyone else, operating system will stop them.

2. Implementation

3. Rust Experience

4. Future Work

5. Conclusion

The conclusion goes here. [1]

Acknowledgments

We would like to thank Prof. Chen for his brilliant course and his guidance for our project. He provided us with such a great opportunity to do system programming and rust programming ourselves.

References

[1] R. T. Ian Goldberg, David Wagner and E. Brewer, "A secure environment for untrusted helper applications," in *the Sixth USENIX UNIX Security Symposium*, San Jose, California, 1996.