University of Warsaw
Faculty of Mathematics, Informatics and Mechanics

Krzysztof Małysa
Student no. 394442

Multi-process sandbox for unprivileged users on Linux

Master's thesis
in COMPUTER SCIENCE

Supervisor:
dr Janina Mincer-Daszkiewicz
Institute of Informatics

Warsaw, December 2022

Abstract

TODO

Keywords

sandboxing, security, Linux, secure execution, arbitrary code execution

Thesis domain (Socrates-Erasmus subject area codes)

11.3 Informatics, Computer Science

Subject classification

Security and privacy – Systems security – Operating systems security

Tytuł pracy w języku polskim

Sandbox wielu procesów dla nieuprzywilejowanych użytkowników systemu Linux

Contents

Chapter 1

Introduction

1.1. Background

Secure execution environments are commonplace these days, from containers and virtual machines on servers to sandboxes on laptop and smartphones — most of which run on Linux. They are used to securely execute untrusted code, as well as trusted programs to prevent damage escalation in the event of unknown bugs. Their key features are isolation, limiting resources, and accounting for resource consumption.

The features of Linux allow the creation of simple yet effective and efficient secure environments. They work at application runtime, so in most cases existing software does not need to be adapted to use them. This makes them easily applicable, and explains why they adoption is growing.

1.2. Related work

Approaches to form a secure execution environments differ. One of them is virtualization or emulation e.g. QEMU [6] and KVM [5], VirtualBox [7], VMWare Workstation [15]. Although powerful and effective, they come with an enormous overhead i.e. booting up an entire operating system. Moreover, emulation noticeably slows down the runtime of an emulated application.

Containers provide much lower overhead: setup of an order of milliseconds and negligible runtime overhead. Docker [4], LXC [1] require root privileges to create a container. systemd-nspawn [14] requires root privileges to run.

Rootless are containers [12] that can be created and run by an unprivileged user. I discovered them lately in the process of doing this thesis. They provide almost all of the functionality of the normal containers but without the need to engage a privileged user. However, they often use setuid binaries and that is undesirable [13]. Also they are not optimized to run sequences of short-running programs.

1.3. Goal of this thesis

The Sim project [3] is an online platform for preparing people for and carrying out algorithmic contests. It has an online judge with a sandbox specially developed for this use case. Over the years the sandbox became a limitation. It only allows running a single-threaded statically linked executable of programs written in C, C++ or Pascal.

Nowadays, the lack of support for popular languages that require dynamic linking e.g. Python, or languages with a complex runtime e.g. Java (muli-thread runtime) is an issue that deters users who want to learn and practice these languages. Supporting such languages requires a different approach. The current sandbox uses `seccomp` filters [10] to filter syscalls and `ptrace` [9] to inspect the syscall arguments e.g. `open` [8]. Such checking of indirect syscalls arguments is prone to Time Of Check To Time Of Use race condition [16] if the memory is shared by at least two threads or processes, making the current approach ineffective for multi-threaded runtimes.

Compiled languages require compiling the source code in order to execute it. Currently, the Sim project sanitizes compilation only by allowing the compiler to access part of the file system using PRoot [11]. To prevent users from exploiting the compiler through vulnerabilities to perform unsafe syscalls a stronger method is required.

One might use secure compilers to increase safety of the compilation [2]. However, many languages are lacking secure compilers and even secure compilers are still exposed to bugs and vulnerabilities in the compiler itself. External sandbox alleviates these shortcomings by treating the compiler as a program fully controlled by the user. Sandboxing the compiler requires sandboxing a group of processes e.g. for C++. The similarity of a group of processes to a group of threads allows using the same sandbox for running multi-threaded runtime and for securing compilation.

The goal of this thesis is to create an efficient sandbox capable of safely isolating and running compilers, dynamically linked executables and multi-threaded runtimes.

1.4. Structure of the thesis

Chapter 2 contains brief summary of the Linux kernel features employed by the sandbox. Chapter 3 describes the sandbox and its design. In chapter 4 comparison with the rootless containers is described.

Chapter 2

Kernel features

Chapter 3

Sandbox design

Chapter 4

Comparison with rootless containers

Bibliography

[1] David Beserra et al. "Performance Analysis of LXC for HPC Environments." In: *CISIS*. IEEE Computer Society, 2015, pp. 358–363. ISBN: 978-1-4799-8870-9. URL: http://dblp.uni-trier.de/db/conf/cisis/cisis2015.html#BeserraMEBSF15.

[2] YangSun Lee, Junho Jeong, and Yunsik Son. "Design and implementation of the secure compiler and virtual machine for developing secure IoT services". In: *Future Generation Computer Systems* 76 (2017), pp. 350–357. ISSN: 0167-739X. DOI: https://doi.org/10.1016/j.future.2016.03.014. URL: https://www.sciencedirect.com/science/article/pii/S0167739X16300589.

[3] Krzysztof Małysa. *Sim project*. URL: https://github.com/varqox/sim (visited on 03/15/2023).

[4] Dirk Merkel. "Docker: Lightweight Linux Containers for Consistent Development and Deployment". In: *Linux J.* 2014.239 (Mar. 2014). ISSN: 1075-3583. URL: http://dl.acm.org/citation.cfm?id=2600239.2600241.

[5] *Official website of Kernel Virtual Machine*. URL: https://www.linux-kvm.org/ (visited on 11/23/2022).

[6] *Official website of QEMU — A generic and open source machine emulator and virtualizer*. URL: https://www.qemu.org/ (visited on 11/23/2022).

[7] Oracle. *Official website of VirtualBox*. URL: https://www.virtualbox.org/ (visited on 11/23/2022).

[8] man-pages project. *open, openat, creat - open and possibly create a file*. URL: https://man7.org/linux/man-pages/man2/open.2.html (visited on 03/15/2023).

[9] man-pages project. *ptrace - process trace*. URL: https://man7.org/linux/man-pages/man2/ptrace.2.html (visited on 03/15/2023).

[10] man-pages project. *seccomp - operate on Secure Computing state of the process*. URL: https://man7.org/linux/man-pages/man2/seccomp.2.html (visited on 03/15/2023).

[11] *PRoot — chroot, mount –bind, and binfmt_misc without privilege/setup*. URL: https://proot-me.github.io (visited on 03/29/2023).

[12] rootlesscontaine.rs. *Rootless Containers*. URL: https://rootlesscontaine.rs (visited on 11/28/2022).

[13] Giuseppe Scrivano. *Rootless containers with Podman and fuse-overlayfs*. June 4, 2019. URL: https://indico.cern.ch/event/757415/contributions/3421994/attachments/1855302/3047064/Podman_Rootless_Containers.pdf (visited on 11/28/2022).

[14] systemd. *systemd-nspawn — Spawn a command or OS in a light-weight container*. URL: https://www.freedesktop.org/software/systemd/man/systemd-nspawn.html (visited on 11/28/2022).

[15] VMWare. *Official website of VMWare Workstation.* URL: https://www.vmware.com/products/workstation/ (visited on 11/23/2022).

[16] Jinpeng Wei and Calton Pu. "TOCTTOU Vulnerabilities in UNIX-Style File Systems: An Anatomical Study". In: *4th USENIX Conference on File and Storage Technologies (FAST 05)*. San Francisco, CA: USENIX Association, Dec. 2005. URL: https://www.usenix.org/conference/fast-05/tocttou-vulnerabilities-unix-style-file-systems-anatomical-study.