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, judging system,

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 vulnerabilities. Their key features are isolation, limiting resource usage, 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 their adoption is growing.

In this thesis, the most important application of sandboxing are online judge systems. Online judge systems have beneficial role in programming education and competitive programming. They allow testing user-provided solution to a specific problem. The solution is run on a predefined test cases in order to check if it is valid. In such platforms isolating the compilation and running of the solution is essential to provide security and robustness of the platform itself.

Historically, isolation techniques evolved together with the online judge platforms. The most primitive (yet insecure) was usage of `chroot(2)` [23] to restrict access to part of the filesystem. To increase isolation virtual machines were used [2]. Later, containerization became a new way to provide isolation [15, 28].

Online education platforms greatly facilitate teaching and learning programming. They provide quick feedback on the correctness of the code the user submits. They are used in schools and universities and provide great learning opportunities for all.

Moreover, a versatile sandbox has applications outside online judging platforms. For example, it can be used to sanitize compiling a PDF form LaTeX sources or for safe execution of untrusted server-side scripts in web applications.

1.2. Goal of the thesis

The goal of the thesis is to design, implement and integrate a new sandbox for the Sim project [11]. The Sim project is an online platform for preparing people for and carrying out algorithmic contests. The project started in 2012 and is developed by me since the beginning. It is used at the XIII High School in Szczecin and programming camps to teach young people programming and algorithms. 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. The new sandbox will allow supporting more programming languages and improve security of the solution compilation stage.

1.2.1. Requirements

The new sandbox needs to be optimized for running short-running programs as well as have minimal runtime overhead. Most of the test cases the solution is run on are small and solution completes them in less than 10ms. The goal is to allow hundreds of such sort-running runs per second, hence optimizing for short-running programs is important. However, minimizing overhead of the sandbox during the run is also important i.e. if the program runs X ms normally, the objective is that the program inside the new sandbox will also run approximately X ms.

The new sandbox needs to be versatile. It will be used to secure the compilation of the solutions as well as running of the solutions. Compilation is a complicated process that involves parsing, translating, optimizing and linking the final program. For languages like C, C++ and Rust it involves running several executables in coordination e.g. compiler and linker i.e. more than one process at a moment — the sandbox needs to support that.

Sandboxing needs to have a low overhead. Apart from small test where solution runs quickly (a matter of milliseconds), almost always the solution is run on big test cases, where it may need several seconds for it to solve the problem. Increasing this time as little as possible while the solution is running inside the sandbox is one of the primary objectives.

It often requires running several executables e.g. compiler and linker, so allowing a single process inside sandbox is not enough. Sandboxing solution is simpler, because it is a single process. But since it is often short-running, the overhead needs to be minimal.

The sandbox needs to allow limiting resources. Real time, CPU time, memory – these need to be limited not only for the robustness of the platform, but specific problems require different limits. The goal of some problems is to solve it with very restricted memory e.g. find a missing integer in a random permutation of integers $1, \ldots, n$ without one element, but in constant memory.

The sandbox needs to account resource usage. For every test, the user is presented with consumed memory and CPU time by their solution. The sandbox needs to provide this information.

The last requirement is the sandbox will not require any privileges. There is a tool called Sip [12] for preparing the problem packages for the Sim platform. One of the purposes of the tool is to run the solutions inside the same secure environment as on the Sim platform. The user should not need any privileges to run this tool, so the sandbox should not require them either.

1.2.2. Existing solutions

Approaches to form a secure execution environments differ. One of them is virtualization or emulation e.g. QEMU [21] and KVM [20], VirtualBox [22], VMWare Workstation [31]. 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, rendering such solutions inapplicable.

Containers provide much lower overhead: setup of an order of milliseconds and negligible runtime overhead. But, Docker [16], LXC [1] require root privileges to create a container.

systemd-nspawn [29] requires root privileges to run.

Rootless are containers [26] that can be created and run by an unprivileged user are the almost perfect solution to the problem. 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 [27]. Also they are not optimized to run sequences of short-running programs. In this thesis we will create a sandbox that uses the same techniques as rootless containers but will be optimized for running sequences of short-running programs.

1.3. Structure of the Thesis

Chapter 2 contains overview of sandboxing techniques and existing implementations and comparative analysis of them. In chapter 3 all aspects of the development process from architecture and design to testing, debugging and performance evaluation are described. Details of design and architecture are described in chapter 4. Implementation is described in 5. Chapter 6 contains performance evaluation of the final implementation and impact of some optimizations. Later, in chapter 7 use cases and applications are discussed. Chapter 8 details integration with online judge and challenges involved. In chapter 9 future work and opportunities are discussed. Finally, chapter 10 contains the conclusion.

Chapter 2

Literature overview

2.1. Overview of Sandboxing Techniques

During the first programming competitions, the human judges manually read and verified the source code of the contestants' solutions [30]. Over time this became infeasible and gave birth to automatic judge systems.

To prevent people from interfering with the normal workflow of the competition e.g. Denial of Service Attack by exhausting memory resources, the automatic judge systems need a secure way to compile and execute a contest's solution. This is where sandboxes come into place.

First sandboxes required modification of the OS kernel [24, 4, 5, 7, 10]. While they had little run-time overhead, some of them were limited to single-threaded applications [18].

Later, as support for process tracing matured, `ptrace`-based sandboxes arose [14, 9, 8]. The problem with those solutions is the overhead that varies from around 75% [17] to 160% [15] for syscall-intensive programs. This overhead however, does not affect programming contest fairness much [13]. Supporting multi-threaded and multi-process programs while using `ptrace` is tricky, but possible [8], because of Time of Check/Time of Use (TOCTOU) problem. `ptrace`-based sandbox needs to inspect syscall arguments. To do so it has to read them, but the multi-threaded or multi-process program can change the indirect argument after the reading but before the kernel uses the argument. Thus creating a dangerous race condition.

Finally, after the kernel support for containerization materialized, namespace and cgroup based sandboxes came into place [15, 19, 25, 6, 3, 28]. Contrary to ptrace-based sandboxes, namespace-based sandboxes have negligible runtime overhead [15]. Moreover, they don't require modifications of the Linux kernel and work on major Linux distributions out of the box.

Chapter 3

Methodology

Chapter 4

Design

Chapter 5

Implementation

Chapter 6

Performance Evaluation

Chapter 7

Use Cases and Applications

Chapter 8

Integration with Online Judge

Chapter 9

Future work and Opportunities

Chapter 10

Conclusion

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] Sander van der Burg and Eelco Dolstra. "Automating System Tests Using Declarative Virtual Machines". In: *2010 IEEE 21st International Symposium on Software Reliability Engineering*. 2010, pp. 181–190. DOI: 10.1109/ISSRE.2010.34.

[3] Flatpak. *Flatpak - the future of application distribution*. URL: https://flatpak.org/ (visited on 2023-10-17).

[4] T Garfinkel. "Janus: A practical tool for application sandboxing". In: *http://www. cs. berkeley. edu/daw/janus* (2004).

[5] Tal Garfinkel, Ben Pfaff, Mendel Rosenblum, et al. "Ostia: A Delegating Architecture for Secure System Call Interposition." In: *NDSS*. 2004.

[6] Google. *A light-weight process isolation tool, making use of Linux namespaces and seccomp-bpf syscall filters*. URL: https://github.com/google/nsjail (visited on 2023-10-17).

[7] Suman Jana, Donald E Porter, and Vitaly Shmatikov. "TxBox: Building secure, efficient sandboxes with system transactions". In: *2011 IEEE Symposium on Security and Privacy*. IEEE. 2011, pp. 329–344.

[8] Taesoo Kim and Nickolai Zeldovich. "Practical and effective sandboxing for non-root users". In: *2013 USENIX Annual Technical Conference (USENIX ATC 13)*. 2013, pp. 139–144.

[9] Rob Kolstad. "Infrastructure for contest task development". In: *Olympiads in Informatics* 3 (2009), pp. 38–59.

[10] Yanlin Li et al. "{MiniBox}: A {Two-Way} Sandbox for x86 Native Code". In: *2014 USENIX annual technical conference (USENIX ATC 14)*. 2014, pp. 409–420.

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

[12] Krzysztof Małysa. *Sip – a tool for preparing problem packages for the Sim platform*. URL: https://github.com/varqox/sip (visited on 2023-03-15).

[13] Martin Mareš. "Fairness of Time Constraints." In: *Olympiads in Informatics* 5 (2011), pp. 92–102.

[14] Martin Mareš. "Perspectives on grading systems". In: *Olympiads in Informatics* (2007), pp. 124–130.

[15] Martin Mareš and Bernard Blackham. "A New Contest Sandbox." In: *Olympiads in Informatics* 6 (2012), pp. 100–109. URL: https://ioi.te.lv/oi/pdf/INFOL094.pdf.

[16] Dirk Merkel. "Docker: Lightweight Linux Containers for Consistent Development and Deployment". In: *Linux J.* 2014.239 (2014-03). ISSN: 1075-3583. URL: http://dl.acm.org/citation.cfm?id=2600239.2600241.

[17] Bruce Merry. "Performance analysis of sandboxes for reactive tasks". In: *Olympiads in Informatics* 4 (2010), pp. 87–94.

[18] Bruce Merry. "Using a Linux security module for contest security". In: *Olympiads in Informatics* 3 (2009), pp. 67–73.

[19] netblue30/firejail. *Linux namespaces and seccomp-bpf sandbox.* URL: https://github.com/netblue30/firejail (visited on 2023-10-17).

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

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

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

[23] Vassilis Prevelakis and Diomidis Spinellis. "Sandboxing Applications." In: *Usenix annual technical conference, freenix track.* Citeseer. 2001, pp. 119–126.

[24] Niels Provos. "Improving Host Security with System Call Policies." In: *USENIX Security Symposium.* 2003, pp. 257–272.

[25] Inge Alexander Raknes, Bjørn Fjukstad, and Lars Ailo Bongo. "nsroot: Minimalist process isolation tool implemented with linux namespaces". In: *arXiv preprint arXiv:1609.03750* (2016).

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

[27] Giuseppe Scrivano. *Rootless containers with Podman and fuse-overlayfs.* 2019-06-04. URL: https://indico.cern.ch/event/757415/contributions/3421994/attachments/1855302/3047064/Podman_Rootless_Containers.pdf (visited on 2022-11-28).

[28] František Špaček, Radomír Sohlich, and Tomáš Dulík. "Docker as Platform for Assignments Evaluation". In: *Procedia Engineering* 100 (2015). 25th DAAAM International Symposium on Intelligent Manufacturing and Automation, 2014, pp. 1665–1671. ISSN: 1877-7058. DOI: https://doi.org/10.1016/j.proeng.2015.01.541. URL: https://www.sciencedirect.com/science/article/pii/S1877705815005688.

[29] 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 2022-11-28).

[30] Tocho Tochev and Tsvetan Bogdanov. "Validating the Security and Stability of the Grader for a Programming Contest System." In: *Olympiads in Informatics* 4 (2010), pp. 113–119.

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