Toward Efficient Fuzzing for Container Escape Vulnerability Detection

Seiga Ueno
The University of Tokyo
Tokyo, Japan

Takahiro Shinagawa
The University of Tokyo
Tokyo, Japan

EXTENDED ABSTRACT

In recent years, container technology has gained popularity in modern cloud and enterprise computing environments. Containers are more resource-efficient and lightweight than virtual machines, making them ideal for serverless computing, microservices, and other cloud-native applications. However, containers are process-level isolated environments that share the single OS kernel and container runtime, leading to container escape vulnerabilities when there are flaws in their isolation mechanisms [4]. These vulnerabilities allow attackers to gain unauthorized access to the host system or other container resources, leading to data theft, system compromise, and lateral movement within the network. Therefore, identifying and mitigating container escape vulnerabilities is crucial for maintaining containerized environment security.

Fuzzing is a software testing technique that involves automatically generating and inputting random data into a program to uncover security vulnerabilities and bugs. Compared to manual testing and static analysis, fuzzing can explore numerous execution paths and detect vulnerabilities that are often missed by other methods. However, leveraging the fuzzing technique to find container escape vulnerabilities faces the following challenges. (1) Extensive exploration space: The OS kernel's codebase is extensive and complex, encompassing a wide range of functionalities and modules. However, container-related code constitutes only a small portion of the OS kernel, making random input targeting inefficient for discovering container-specific vulnerabilities. (2) Detecting non-crashing vulnerabilities: Fuzzing typically finds bugs that trigger system crashes, but container escape does not necessarily cause crashes, making their detection challenging.

Syzkaller is a popular Linux kernel fuzzer that uses system call sequences to identify vulnerabilities [2]. However, Syzkaller is not specifically designed to target container-specific vulnerabilities, making it less effective for finding container escape issues. Torpedo [3] is a specialized fuzzer designed to find container vulnerabilities by extending Syzkaller's functionality. However, Torpedo primarily focuses on resource-limit bypassing vulnerabilities, which allow containers to exceed their allocated CPU usage limits, and does not address container escape vulnerabilities in general. Paced [1] is a real-time system for detecting container escape attacks by analyzing cross-namespace events with a provenance-based approach. However, Paced is based on provenance and has less kernel code coverage than Fuzzing. CPEED [5] performs integrity measurements in a hardware-isolated security domain to detect container escape. However, CPEED requires additional hardware and runtime overhead.

We propose a novel fuzzing framework to efficiently discover container escape vulnerabilities. To address the challenge of the vast kernel space, we introduce directed fuzzing towards fuzzing-related kernel code fragments. For instance, container-enabling technologies include namespaces and cgroups, with namespaces encompassing several sub components. We identify container-specific code in the source based on their component names and prioritize seeds that focus on executing this code intensively, thereby allowing concentrated testing of container-related code. To address the challenge of non-crash vulnerabilities, we introduce a dedicated container escape detector. For example, to detect incorrect conditions where an application in a container has access to unauthorized host resources, such as files, we check the capability of the application in the container, such as file descriptors, in the host kernel at an appropriate timing. This allows for the detection of conditions where the kernel does not crash but the in-container application is actually silently escaping, which is undetectable with conventional sanitizers.

As a first step toward detecting container escapes in general, we aim to detect mount namespace escapes. First, we identify code sections that handle kernel data structures related to file descriptors in order to perform directed fuzzing of kernel code that handles mounted namespaces. Specifically, we identify functions that handle `struct file` structures and perform directed fuzzing on them so that they can be intensively tested. In addition, to detect vulnerabilities that allow mount namespace escapes, we obtain a list of i-nodes of files accessible by the container and periodically scan file descriptors that are open by the container. This allows the detection of file descriptors that have opened files to which the container should not have access rights.

We are in the process of implementing this fuzzing framework by extending Syzkaller. We used Docker on Linux as the target for fuzzing, with fuzzing system calls issued from inside the container. AFL is used for the Fuzzer. Currently, our first goal is to detect CVE-2024-21626. Then we will consider directed fuzzing methods and container escape detection methods for other namespaces such as PID, Network, user, IPC, and UTS.

References

[1] Mashal Abbas et al. 2022. PACED: Provenance-based Automated Container Escape Detection. In *Proc. 2022 IEEE International Conference on Cloud Engineering (IC2E)*. 261–272. https://doi.org/10.1109/IC2E55432.2022.00035

[2] Google. 2016. syzkaller - kernel fuzzer. https://github.com/google/syzkaller.

[3] Kenton McDonough et al. 2022. Torpedo: A Fuzzing Framework for Discovering Adversarial Container Workloads. In *Proc. 2022 52nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. 402–414. https://doi.org/10.1109/DSN53405.2022.00048

[4] MITRE Corporation. 2024. CVE-2024-21626: Runc Container Breakout Through Process.Cwd Trickery And Leaked Fds.

[5] Mengting Zhou et al. 2023. Container Privilege Escalation and Escape Detection Method Based on Security-First Architecture. In *Proc. 2023 IEEE International Conference on High Performance Computing Communications, Data Science Systems, Smart City Dependability in Sensor, Cloud Big Data Systems Application (HPCC/DSS/SmartCity/DependSys)*. 490–498. https://doi.org/10.1109/HPCC-DSS-SmartCity-DependSys60770.2023.00073