AWS Firecracker MicroVMs to Host Rootless Podman Containers

Introduction

Contributions

Lim Xin Yi contributed to the design and implementation of the project, under the supervision of Nicholas Sim.

Overview

This report outlines the implementation and benefits of nesting each Secure Internet Surfing (SIS) container within a lightweight virtual machine (VM) using Firecracker microVMs (uVMs). The SIS system currently relies on rootless Podman containers for user session isolation, and this project aims to improve security by leveraging the hardware-level isolation provided by Firecracker uVMs. By doing so, we fortify the segmentation boundary between user sessions and between each session and the host, ultimately strengthening the security of the SIS system.

Background / Motivation

The existing SIS System utilises rootless Podman containers, relying on namespaces to establish a layer of separation between the user and the host server. However, containerization only offers operating system (OS) level isolation. Additionally, all containers are controlled by a common container engine, which directly interacts with the host OS.

On the other hand, traditional virtualization provides hardware-level isolation, with each VM having its own set of virtualized resources, including vCPUs and vIO. This approach executes the VM's instructions in its own kernel, independent of the host's kernel. However, traditional VMs are typically bulky and have longer startup times.

Proposal

To overcome the limitations of both containerization and traditional VMs, our solution is to adopt Firecracker microVMs (uVMs). Firecracker enables the deployment of such lightweight uVMs, excluding unnecessary devices, making them ideal for our use case. Additionally, we can configure the kernel used by each uVM to include only essential functionality, reducing boot time, memory footprint, and attack surface area.

Commented [SN1]: Add a "Contributions" subsection to the intro

Commented [SN2]: This section appears to only be background. For motivation also, could you write down the big idea here? e.g. isolation, ephemeral workloads -- "what is the problem this work tries to solve"?

Commented [SN3R2]: It is ok if this is very short, but it must tell the reader:
- What this is about
- Why they might want to read it

Commented [SN4]: Move this to a new "Background and related work" section

Commented [SN5]: Move this to the "Design" section

Besides the enhanced security, Firecracker also offers a snapshotting feature, valuable for backup and recovery purposes and ensuring the system's resilience against unexpected incidents.

Architecture / Design / Setup

Commented [SN6]: Just "Design" will do

Commented [SN7R6]: another todo: explain how this design attempts to meet the isolation requirements documented in the intro. This will be your own "Evaluation" of the work

Chroot Jail

In production, we run Firecracker within an execution jail designed to isolate the Firecracker process and enhance its security posture. Fundamentally, it is a chroot jail that contains its own copy of the Firecracker binary, tuntap device, kvm, and urandom device. Additionally, the kernel image and file system images to be mounted onto the uVM must also reside within the jail.

Seccomp TODO: add what main syscall are excluded

To restrict the uVM's access to system calls, Seccomp filters are implemented.

Cgroups TODO: add specific limits like swap mem, cpu duration usage etc

We use cgroup to limit a Firecracker process' resources, namely:

- Disk
- Memory
- vCPU

Implementation

Configuring Firecracker

While Firecracker VMM is built to be processor agnostic and hardware virtualization support for 64-bit Intel CPUs are generally available, we encountered an issue with detecting the frequency of our Intel Xeon Gold 5318Y processor. To resolve this, we could either provide the frequency or edit the Firecracker source code to ignore the frequency. Due to time constraints, we employed the more straightforward method of hardcoding the host CPU's frequency into the Firecracker source code.

Kernel

To host rootless Podman containers in the uVMs, the base kernel configuration recommended by Firecracker is insufficient and we require several other modules which mainly include:

- Netfilter modules to support nf-tables in the guest VM for Internet connectivity.
- Fuse-overlayfs modules to support filesystem in userspace for rootless containers. *Note: this could be redundant for kernels >5.14 which will not be deployed as of now due to the lack of proper snapshotting support.*
- Hardware random number generator modules to support greater entropy to enhance snapshot security which is elaborated in the later section.

We can then use the kernel building tool provided by Firecracker to build the guest kernel based on the updated config file.

Rootfs

The root file system (rootfs) of each uVM is mounted with read-only option while a secondary disk mounted as tmpfs with read-write options. Unlike having just a single read-write rootfs, this avoids the need for multiple copies of the rootfs image. Each uVM only requires a hard link that points to a common rootfs EXT4 image. Albeit having a copy of the secondary disk for each uVM, there is no RAM wastage since tmpfs consumes only the required amount of RAM at any point in time instead of reserving the full amount of RAM.

To build a custom rootfs image, we first prepare a file of size *<1.2GB>* and create an empty EXT4 file system on it. To populate it with the init system, /usr filesystem etc. and tools including Slirp4netns, Podman we do the following:

1. Bind-mount it onto an Alpine Docker container
2. Install OpenRC init system (for Alpine) and other tools like Podman, Slirp4netns etc.
3. Add a normal user without CAP_SYS_ADMIN, and modify file permissions of the FUSE and KVM devices to assign read-write permissions to the user.
4. Add setuid and setgid capabilities to /usr/bin/newuidmap and /usr/bin/newgidmap respectively. This allows Podman to write user and group ID mappings in a non-root user namespace
5. Edit container engine and storage configuration files to redirect the storage such as container image layer files to the user's home directory ie. the mount point of the RW disk.
6. Copy the newly configured system onto the mounted rootfs image.

Commented [SN8]: For future reference: if you prefer to just write the scripts (with comments!), you can do that and write in the report "see this script for the details"

We can launch the Firecracker process with the jailer feature and pass in the kernel image and rootfs image either via a config file or the API socket. To enable networking between the uVM and the host, we create a TAP device on the host that is within the same subnet as the uVM's which is specified in the boot kernel arguments.

Features

Snapshotting

Hosting each user session in an uVM enables snapshotting, which is done by sending an API request to the socket. This allows the original guest workload to be resumed in a different Firecracker process**.** TO make this possible, the snapshots save the full state of the guest memory and emulated hardware state, stored across multiple files as below:

- Guest memory file,
- uVM state file,
- Disk files

This is a significant upgrade from containers where there is a lack of support for checkpointing rootless Podman containers which SIS is currently deploying.

It is important to ensure that these snapshots are loaded properly and do not lead to potential security risks. The greater guest OS entropy pool mentioned above helps to generate higher quality random numbers and tokens to ensure uVM uniqueness. However, there still lacks a strong mechanism that guarantees that the unique identifiers of each uVM remains unique across snapshot restores. Hence, a snapshot should only be loaded once such that resumption of execution from a state happens only once.

Logger

A Firecracker logger is configured for each uVM on startup to record the uVM's base activity. It will log the line of Firecracker source code that was executed and the outcome. For instance, when a PUT request is sent to the API server to create a full snapshot of the uVM, the line of source code triggered to parse the request, the response, and the API call duration would all be logged. In the event where we would like to examine malicious activity detected in a uVM, we could complement its logs with the snapshot and reverse-engineer them. Ideally, the snapshot should give us an idea of the events that happened in the user space while the logs detail the events that happened in kernel space.

Performance TODO: add in the metrics after measuring

uVM boot time

uVM RAM usage

Snapshot (storage, save and load latency)

Alternative Approaches

Commented [SN9]: Move to "related work", after the intro section

Kata Containers

Kata Containers integrates containerization and virtualization into a single platform. It uses a hypervisor to launch a lightweight VM that runs systemd as its kernel init system which launches an agent in the VM to create a Docker container environment. It offers flexibility in terms of the hypervisor used, with support for QEMU, Cloud-hypervisor, Firecracker etc. Hence, it could be an alternative to the Firecracker-Podman design outlined above when more guest functionality such as ACPI Power Management is desired since QEMU supports a larger range of hardware emulation as compared to Firecracker. However, it is also important to account for the increased startup latency and resource consumption for more complex uVMs, which can limit the number of uVMs deployed at any point in time.

Nabla Containers

Nabla containers adopt a strategy of attack surface reduction to the host by minimising the containers' access to system calls. It uses library OS techniques to substitute all but 7 system calls, isolating the container from the host container.

Future work

Network Filtering

All network traffic in the uVM is routed through eth0 on the guest to its corresponding TAP device on the host, before getting forwarded to enp1s0 on the host. We employ Nftables to set up a firewall between the uVM and the host. A simple use case would be to block packets with a certain destination address that is deemed to be unsafe from being forwarded from the uVM to the host.

Commented [SN10]: For future work subsections, put the motivation up front, and in the first sentence if possible

Conclusion

The deployment of Firecracker microVMs to host rootless Podman containers in the Secure Internet Surfing (SIS) system offers enhanced security and resilience. With a strong focus on hardware-level isolation and efficient snapshotting, Firecracker uVMs provide a robust solution for secure internet browsing, complementing the existing containerized architecture. As we continue to refine and optimise our setup, we anticipate further improvements in security, performance, and overall system reliability.

References

https://nabla-containers.github.io/

www.katacontainers.io

https://podman.io/docs

https://github.com/kata-containers

https://www.cs.princeton.edu/~yuetan/assets/snapfaas-poster.pdf

https://github.com/firecracker-microvm/firecracker

https://wiki.nftables.org/wiki-nftables/index.php/Quick_reference-nftables_in_10_minutes

https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html/configuring_and_managing_networking/getting-started-with-nftables_configuring-and-managing-networking

https://www.cs.yale.edu/homes/aspnes/pinewiki/Virtualization.html

https://man7.org/linux/man-pages/man7/capabilities.7.html