Forensic analysis of container checkpoints

[criu.org]

Mentors: Radostin Stoyanov, Adrian Reber

Mentee Personal Information:

- Name: Kouame Behouba Manasse
- Email: behouba@gmail.com
- GitHub: github.com/behouba
- Matrix: @behouba:matrix.org
- Location: Abidjan, Côte d'Ivoire
- Timezone: GMT+0

Table of contents

I. Introduction
II. Technical details
    A. Container checkpoint
III. Implementation plan
    A. Requirements
    B. Extend go-crit
    C. Extend checkpointctl
IV. Timeline Proposition
V. About me
    A. Why me?
    B. Open-source contributions
VI. Time commitments

I. Introduction

The increasing popularity of containerization technology has brought significant benefits to software development, deployment, and management. However, the use of containers also presents new challenges in the field of digital forensics. Containerized applications often run in isolation from the host operating system, making it difficult to access and analyze their contents in the event of security breaches, cyber-attacks, or other criminal activities. In response to this challenge, tools like CRIU (Checkpoint/Restore In Userspace) have emerged with the capability of checkpointing and restoring running containers making it possible to investigate container state at a given time without the original container being stopped nor being aware that it has been checkpointed.

CRIU provides a CLI tool called CRIT, primarily implemented in Python, to analyze CRIU images. However, go-crit was developed during GSoC 2022 to enable Golang tools to analyze CRIU images. Although go-crit is excellent for encoding and decoding CRIU images, it lacks user-friendliness when analyzing container checkpoints. The goal of this proposal is to extend go-crit with container checkpoint forensic analysis features to improve the tool's usability in other tools such as checkpointctl. As a result, checkpointctl will be enhanced to provide user-friendly features to quickly and easily get needed information about container runtime state.

II. Technical details

A. Container checkpoint

When CRIU is used to checkpoint a container, it produces a tar archive file that can be used to analyze the runtime state of the container. When extracted, the content of a container checkpoint directory will typically look as follows:

- / (root directory)
  - /artifacts
  - **/checkpoint** this directory contains the actual checkpoint as created by CRIU
    - cgroup.img -this file contains information about cgroups (control groups) that were associated with the container at the time it was checkpointed.
    - core-[PID].img- these files contain detailed information about the execution environment and the state of the processes.

- fdinfo-*.img - these files contain information about open file descriptors in the container.
- fs-*.img - these files contain information about chroot and chdir of processes.
- mm-*.img - these files contain information about processes' memory address space, including VMAs, segments, and other memory-related data.
- mountpoints-*.img - these files contain information about the mount points in the container.
- netns-*.img - this file contains information about the container's network namespace,
- pagemap-[PID].img - this file contains information about the process's memory pages.
- pagemap-shmem-*.img - this file contains information about shared memory segments in the container.
- pages-*.img - these files contain raw binary memory data of memory pages.
- pstree.img - this file contains information about the processes running in the container, including their parent-child relationships.
- seccomp.img - this file contains information about the container's seccomp filters.
- timens-*.img - this file contains information about the container's time namespace.
- tmpfs-dev-*.tar.gz.img - this file contains information about tmpfs filesystems mounted in the container.
- utsns-*.img - these files contain information about the container's UTS namespace, including the hostname and domain name.
- mmfd.img - this file contains information about the file descriptor used to manage the container's memory mappings.
- inventory.img- this file contains information about top-level descriptions of images.
- descriptors.json: a JSON file
- files.img - this file contains the list of all open files of different type present inside the container.
- ids-*.img- these files contain information about various types of IDs used in the container, such as VM, files, fs, IPC, control group, etc.
- ipcns-var-11.img - these files contain information about the container's IPC namespace.
- pipes-data.img - these files contain information about the content of the pipes of the container.

- dump.log - this file contains the debug output created during the checkpoint.
- devshm-checkpoint.tar: this file contains the state of the shared memory devices associated with the container.
- bind.mounts - this file contains information needed when restoring the container to mount all external files and directories.
- network.status - this file contains the network configuration for the container, such as the IP address, gateway, and mac address information.
- rootfs-diff.tar - this archive file contains all changed files on the container's file system.
- spec.dump and config.dump - these files contain OCI specification format metadata for the container, which describes how the container should be run and configured and the metadata needed to restore the checkpoint.
- stats-dump - this file contains statistics information about CRIU the memory dumping process of the container.

The structure of files and directories presented here is not exhaustive since it is based on the result of my experimentations checkpointing and restoring various containers using Podman on a Fedora 37 host system. So, it is possible to find additional files or directories inside the checkpoint folder depending on the type of container being checkpointed.

III. Implementation plan

A. Requirements

The implementation plan is based on a set of functional requirements to provide a user-friendly experience using checkpointctl to analyze a container checkpoint. Considering that we won't support incremental checkpoints initially, the user of checkpointctl will be able to:

- List all processes within a container checkpoint. Displaying for each process: PID, process name, and some additional data about the process.
- Display the process tree in a user-friendly manner.
- Display the memory mapping information of processes.
- Display the arguments used to run processes.
- Display environment variables of processes
- Display a list of open files within the checkpoint or per process.
- Display the network connections within the checkpoint or per process.
- Display the files that have changed within the container.

- Extract the files that have changed within the container for further analysis.
- Display the list of mountpoints within to container checkpoint.
- Display the show a list tmpfs filesystems.
- Export memory pages of a process for further memory analysis.

B. Extend go-crit

go-crit will be extended to provide functions designed to retrieve various information for forensic analysis from CRIU images found in the container checkpoint images. It is important to mention that, `go-crit/crit` package is currently capable of exploring the process tree, file descriptors, and memory mapping of processes within a checkpoint.

The following public functions will be added to `go-crit/crit` package:

- `GetPsTree(dir string, pid int) (tree *Process, err error)` This function will return the process tree from the specified PID. Will return the entire process tree if PID is 0.
- `GetPsList(dir string, pids []int) (ps []Process, err error)` This function will return a list of processes with specified PID. Will return the list of all processes if no PID is given.
- `GetMemMaps(dir string, pids []int) (maps []MemMap, err error)` This function will return a list of memory maps of a process or processes.
- `GetArguments(dir string, pid int) (args []string, err error)` This function will read the memory address `env_start` and `env_end` of a process, then, read this address space from memory pages and return the process arguments.
- `GetEnvVars(dir string, pid int) (envVars []EnvVar, err error)` This function will return the environment variables of a process.
- `GetFds(dir string, pids []int) (fds []Fd, err error)` This function will return a list of file opened file descriptors.
- `GetBashHistory(dir string, pids []int) (entries []HistEntry, err error)` This function will try to retrieve bash history from memory pages.
- `GetSockets(dir string, pids []int) (sockets []Socket, err error)` This function will return socket file associated with the specified processes ids. If no PID is provided will return all sockets of the container.
- `GetMountPoints(dir string) (mnts []MountPoint, err error)` This function will return a list of mount points of the container namespace.
- `GetMemPages(dir string, pid, start, end uint64) (pages []byte, err error)` This function will try to retrieve the memory pages of a process.

NOTE: The dir argument present in the functions above is the path to the checkpoint/ director where the container checkpoint is extracted.

A preliminary work containing the function definitions and data types presented above can be found here.

C. Extend checkpointctl

Currently, the checkpointctl CLI tool is very limited as to what information it can display from a container checkpoint file. The checkpointctl CLI tool will be updated to provide a better user experience for forensic analysis of container checkpoints.

The following command will be added to checkpointctl:

- pslist show all processes within the container checkpoint.
- pstree show the entire list of processes within the container checkpoint to help visualize the parent/child relationships.
- ps_mmaps show memory mapping information for processes.
- ps_args show the arguments used to run processes.
- ps_env show processes environment variables.
- lsof show a list process of open files.
- netstat show processes network connections.
- bash show bash commands history
- dump_map extract memory pages of the process
- list_diff list the files that have changed.
- extract_diff extract files that have changed.
- mount show the list of mountpoints.
- tmpfs show a list tmpfs filesystems.

IV. Timeline Proposition

Period	**Work**
Present - 28 May 2023	Community bounding Discuss the features and design with Mentors and the CRIU community. Learn more about memory page files produced by CRIU. Prototype the API for forensic analysis of go-criu.
29 May 2023 - 21 June 2023	Implement container checkpoint forensic analysis API of go-criu/crit package
22 June 2023 - 09 July 2023	Testing and bug fixes of the new features in go-criu/crit
10 July 2023 - 13 July 2023	Midterm evaluation
14 July 2023 - 04 August 2023	Extend checkpointctl with new commands for forensic analysis of container checkpoints using the newly created go-criu/crit API
05 August 2023 - 20 August 2023	Testing and bug fixes of the new commands added to checkpointctl.
21 August 2023 - 28 August 2023	Final evaluation
Henceforward	Continue supporting the CRIU community on go-criu, checkpointctl, and other projects of CRIU in my spare time.

V. About me

A. Why me?

I have been using container technologies like Docker, Podman, and Kubernetes for years. I discovered CRIU this year and have been fascinated by it since then. I see the potential of CRIU in the field of cloud computing. So, I would like to see CRIU projects grow, which is why I am deeply interested in contributing to this project.

I hold a bachelor's degree in computer science from Dagestan state technical university and I have completed 1 year of a master's degree in software engineering at Innopolis University. These studies provided me with a solid foundation in computer science, especially in software engineering.

Besides my academic background, I have been using Golang for over 4 years through personal projects, internships, and open-source contributions. I am also passionate about Linux, containerization, and cybersecurity.

I am motivated to contribute to this project, and I believe I could help to implement features to make forensic analysis of container checkpoints more user-friendly. I am passionate about open-source software and would like to become a regular contributor to the CRIU projects. I believe my technical background, and knowledge of Golang, Linux, and containerization technologies would allow me to quickly learn and complete this project in time.

B. Open-source contributions

- Syzkaller
  - executor, sys/fuchsia, pkg/csource: fix fuchsia support
  - docs: update fuchsia README.md
- Matrix.org
  - Pass cfg by reference around the codebase
  - selectAccountDataByType return ClientEvent pointer instead of slice of ClientEvent
  - Missing "origin" field added to RespSendJoin
  - Response from /send_join now use gomatrixserverlib.RespSendJoin

During the process of familiarizing myself with CRIU, I opened this PR which has been merged into the criu-dev branch of CRIU.

VI. Time commitments

During the contribution period of GSoC, I have no other commitments. So, I will be able to dedicate 40 hours per week (and more if needed) to focus solely on this project during the summer. Finally, I would like to emphasize that I am not applying to any other organizations under GSoC this year so this project will be my top priority before and after the selected students are announced.