

Docker Forensics - Techniques and Tools

Ângelo Borges
DETI University of Aveiro
Aveiro, Portugal

Rafael Remígio
DETI University of Aveiro
Aveiro, Portugal

Wander Filho
DETI University of Aveiro
Aveiro, Portugal

Wilmara Francisco
DETI University of Aveiro
Aveiro, Portugal

Abstract—The rise of Docker has revolutionized the way applications are deployed and managed in recent years. However, the proliferation of Docker technology has raised pertinent concerns regarding digital forensics within dynamic containers. This paper presents a comprehensive exploration into the domain of Docker and Docker forensics, presents the current challenges of forensic investigations in Dockerized environments, and presents techniques and tools for conducting digital investigations in such environments.

This document delves into the current state-of-the-art forensic techniques tailored for Docker environments, encompassing memory forensics, file system analysis, metadata examination, and artifact extraction methodologies. It evaluates the effectiveness of these techniques for capturing critical evidence in Docker environments.

This document also includes a description for the implementation of a tool capable of conducting some of the forensic techniques described.

Index Terms—Docker, Docker forensics, digital forensics, Docker API

I. INTRODUCTION

The industry standard, in the past years, was the use of Virtual Machines (VMs) to run software applications. VMs run applications inside a guest Operating System, which runs on virtual hardware powered by the server's host OS. Containerized environments have recently become the new norm for the development process, and Kubernetes has risen as a standard for container orchestration platforms. More and more cloud workloads are shifting to containers as these make an ideal choice for cloud environments – credit to their lightweight nature and efficiency.

Nevertheless, containers containing workloads running in the cloud are increasingly becoming a target of cyberattackers. This container revolution has brought with it considerable challenges for digital security professionals, especially in the field of forensic analysis. The ephemerality and isolation inherent in containers, combined with the rapid creation and destruction of these instances, present challenges for specialists in the field. In this context, this work proposes the design and development of a tool dedicated to forensic analysis in container environments, with the aim of providing security professionals with an effective approach to investigating incidents, reconstructing suspicious events and preserving the integrity of containers. During the course of this work, we will focus on the unique challenges that arise due to the ephemeral nature of containers, with special attention to the recovery of crucial artifacts. We will analyze best practices, identify common obstacles and, finally, explore the practical

implementation of the proposed tool. All this will be done with the aim of providing a comprehensive understanding of this complex scenario, addressing the particularities and nuances involved in forensic analysis in container environments.

II. DOCKER

Docker It is a tool that relies on existing features in the kernel, initially Linux, to isolate the execution of processes. The tools that Docker brings are basically a container administration layer, originally based on LXC.[8]

A. Docker Containers

Containers are lightweight software packages that contain all the dependencies required to execute the contained software application. These dependencies include things like system libraries, external third-party code packages, and other operating system-level applications. A Docker Container is the name given to the segregation of processes within the same kernel, so that the process is isolated as much as possible from the rest of the environment, both in terms of memory and file system.

Although containers are enabled by operating system-level features, they are generally managed at a higher level of abstraction.



Fig. 1. Container representation.

B. Docker Images

A Docker Container is the active instantiation of a Docker Image. Docker Images encapsulate all requisite data and metadata necessary for orchestrating a set of processes with explicitly defined attributes. This encompasses crucial details such as the network port allocations of contained applications and the execution commands during the instantiation phase.

An image can be created from a *DockerFile* or by creating a *snapshot* of a Docker Container. A Docker image can create multiple containers.

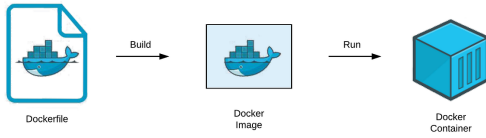


Fig. 2. DockerFile to a Container

The identification of an image is established via a hash function applied to specific segments of the image, currently the function used by the Docker Community is *sha256*. In contrast, container identifiers are generated randomly.

It is common practice for images to be built upon pre-existing images, establishing a hierarchical structure in the construction and deployment of containerized applications.

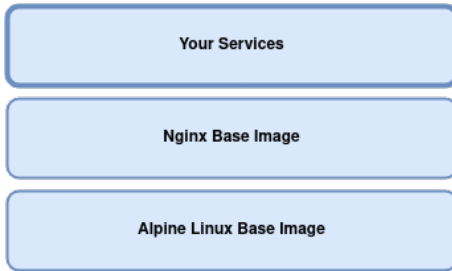


Fig. 3. Docker Images

C. Volumes

Docker Volumes serve currently as the optimal method for data persistence in docker environments. When a volume is created, it is stored as a directory on the Docker host. Upon mounting the volume onto a container, this directory becomes accessible within the container itself. External processes are restricted from modifying this specific part of the file system. When a container is removed, the volume persists and can be mounted to a new container.

A significant advantage lies in the versatility of a single volume, allowing concurrent mounting across multiple containers. It's important to note that even if a volume is not actively linked to any container, it remains accessible and persists, not subject to automatic deletion.

D. Docker API

Docker provides an API for interacting with the Docker daemon (called the Docker Engine API). This tool aid in identifying, storing, and accessing this crucial information. Additionally, Docker's forensic analysis tools, accessible through the Docker CLI and API, enable users to delve into container metadata and logs. Furthermore, Docker offers, alongside the API, software development kits (SDKs) for Go and Python.

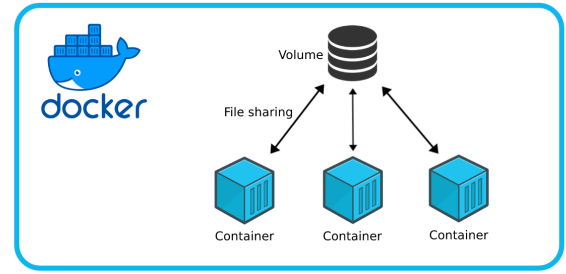


Fig. 4. Docker Volumes

These SDKs empower users to swiftly construct and expand Docker-based applications and solutions, streamlining the development process.

E. Network

Container networking denotes the capability of containers to establish connections and exchange data among the same container environment or machines outside that environment. A Docker Container operates unaware of the underlying network configuration, remaining agnostic to whether its peers are also Docker-based machines or not. It perceives only a network interface encompassing essential networking components. IP address, gateway, routing table, DNS services, and other relevant networking particulars. [9]

Docker has an adaptable network subsystem reliant on various drivers or modes. These modes integrate core networking functionalities:

- **Bridge/Default Mode:** Utilized for enabling communication between containers running on the same host.

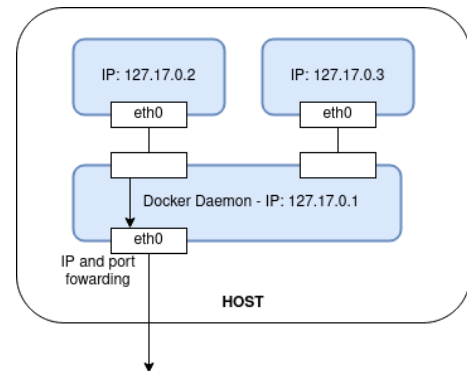


Fig. 5. Bridge network

- **Host:** Use the host's network directly
- **Overlay Mode:** facilitate the formation and operation of Docker swarms, enabling seamless communication between the interconnected daemons.
- **VLans:** *Macvlan* assigns a MAC address to a container, making it appear as a physical device on your network. *IPvlan*, a counterpart to *Macvlan*, differs in not assigning

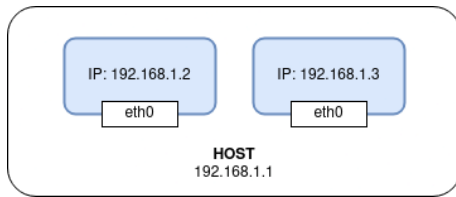


Fig. 6. Host Network

unique MAC addresses to containers but assign an IP address.

- **None:** This mode wholly segregates a container from the host and other containers, ensuring complete isolation.

F. Virtualization and Containerization

The difference between containers and virtual machines lies in the underlying virtualization technology used, which creates a significant difference in their performance.

Containers share the kernel of the host operating system, and with other containers, each running as isolated processes in user space. An advantage of Containers is that resources from memory, file system, and network ports can be shared between different containers and the host itself. Virtual machines, on the other hand, run in a hypervisor environment, where each VM needs to have its own dedicated operating system and other resources like related binaries, libraries, and application files. This not only consumes a significant amount of system resources but also creates considerable overhead when multiple VMs are running on the same physical server.

Virtualization is the process in which a system singular resource like RAM, CPU, Disk, or Networking can be 'virtualized' and represented as multiple resources. The key differentiator between containers and virtual machines is that virtual machines virtualize an entire machine down to the hardware layers and containers only virtualize software layers above the operating system level.

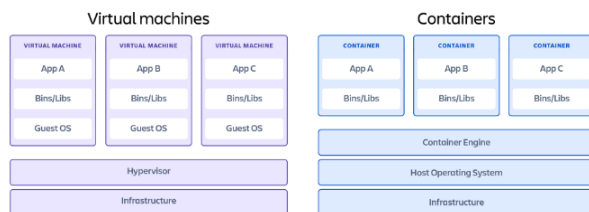


Fig. 7. Comparison of Virtual Machine with Docker[8].

III. DIGITAL FORENSICS

In the current days, Digital Forensics is a crucial tool for solving crimes committed with computers (e.g. phishing and bank fraud), as well as for those where evidence may reside on a computer (e.g. money laundering and child exploitation). Forensic tools have also become a key tool for Information Assurance, due to their ability to reconstruct the evidence left in cyber-attacks.

Electronic evidence is a component of almost all criminal activities and the support of digital forensics is crucial for law enforcement investigations and can be collected from a wide array of sources, such as computers, smartphones, remote storage, unmanned aerial systems, and more.

The growing threat of cyber attacks and data theft has highlighted the importance of digital investigation. Computer forensics experts play a crucial role in tracking malicious activity, recovering lost data, and providing vital evidence to solve cases involving fraud, intellectual property theft, and other cybercrimes.

Computer forensics, a discipline within cybersecurity, focuses on the collection, preservation, analysis, and presentation of digital evidence in legal or investigative contexts. Its main objective is to identify and understand how a cybersecurity incident occurred and who is responsible. In the context of an organization, digital evidence is used as part of the incident response process, to detect that a breach occurred, identify the root cause and threat actors, eradicate the threat, and provide evidence for legal teams and law enforcement authorities.

Digital evidence is described as an interpretation of data, either at rest (in storage) or in transit (network communication), or a combination of both [16]. these tell different, partial stories about the same evidence.[13]

The term "digital forensics" was initially synonymous with "computer forensics", but its definition has expanded to encompass the investigation of all devices capable of storing digital data. Originating in the personal computing revolution of the late 1970s and early 1980s, the discipline underwent haphazard development in the 1990s and began to consolidate national policy in the early 21st century. [3]

Despite the field's quick evolution, advancements in digital forensics are now more difficult to achieve. Its continuous evolution is heavily challenged by the increasing popularity of digital devices and the diversity of the hardware and software platforms being used.[4]

A. Digital Forensic Analysis Process

Preserving the integrity of data on the machine and collecting it reliably are fundamental challenges facing digital investigation. The digital forensic analysis process goes through four main phases:

1. Identification of the Source of Digital Evidence: Consists of identifying the source of digital evidence, which may involve the identification of devices, systems or locations relevant to the investigation.

2. Preservation of Evidence: This phase includes the duplication of evidence in accordance with technical-legal procedures to guarantee its integrity. Preservation is crucial to avoid changes during analysis.

3. Evidence Analysis and Investigation: In this stage, digital evidence is examined and investigated, involving data retrieval, pattern identification and metadata analysis for contextualization.

4. Reporting and Documentation of Results: The results of the forensic analysis are documented in reports,

detailing the methodology, data recovered, conclusions and recommendations. These reports are often required in legal or investigative contexts.

Digital forensic investigations have several applications and, by extracting data from electronic evidence, they can generate valuable information for legal actions. The analysis is conducted by experts in computer forensics, equipped with specialized technical skills in interpreting this data.

The development of virtualization technology has brought new challenges to the traditional digital forensics field. The subject of digital forensics changes from physical machine to VM(virtual machine), from hardware storage device to VM file. Now, with virtualization and cloud computing tech working more closely [7], forensics targeting virtualized environments has become a crucial part of cloud forensics[12].

IV. FORENSIC ANALYSIS IN DOCKER ENVIRONMENTS

Container forensics is the process of collecting, preserving, and analyzing digital evidence from containerized systems and applications. This evidence can be used to identify security incidents (such as data breaches or malware infections) and determine the cause and scope of the incident. The main elements of interest from a digital forensics perspective are how containers store data and how they export logs. The goal of container forensics is to uncover malicious activities, configuration issues, and other security flaws by examining container images, logs, file systems, memory, network traffic, and other artifacts. Additionally, this may necessitate investigating and evaluating the host system, container runtime, container image, and other container environment components for potential threats.

In the event of a security incident in a containerized environment, conducting a forensic examination of the affected container can be a critical step in determining the cause and scope of the incident, and in identifying the necessary steps for remediation. The computer forensic process can be broken down into some distinct key steps for conducting a forensic examination of a container:

- Isolate the affected container from the rest of the environment to prevent any further damage or contamination of evidence.
- Capture a forensic image of the container (which is an exact copy of the container's file system), including all data and metadata. This image can be used for analysis without modifying the original container.
- Analyze the forensic image using appropriate tools and techniques to identify any evidence of the security incident, such as malware, unauthorized access, or data exfiltration.
- Extract relevant data from the forensic image (such as logs, configuration files, and application data) and preserve it for further analysis.
- Correlate the evidence collected from the container with other sources of data, such as network logs, system logs, and security alerts, to build a comprehensive picture of the security incident.

- Report the findings of the forensic examination (including the cause and scope of the security incident) and recommendations for remediation.

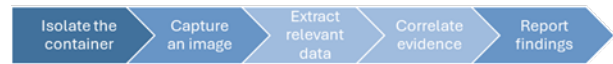


Fig. 8. Steps conducting a forensic examination of a container.

A. Challenges

Unlike the traditional forensic analysis approach, which typically results in a list of files and processes, analysis in Docker environments presents additional challenges. This is because the association of records with specific containers and information about the importance of certain files for rebuilding the actual file system during real-time operation are not readily evident.

Docker containers are inherently volatile and transient, with the ability to be created, destroyed, and re-created easily. This poses a challenge for digital forensics practitioners who rely on stable systems and persistent evidence. Traditional forensic approaches may not be suitable for dynamic Docker environments. Investigators need to adapt their techniques to capture and analyze data from running containers, considering the frequent creation and deletion of containers.

One of the key challenges in container forensics is the ephemeral nature of containers. Unlike traditional servers, which have a persistent disk that can be used to store forensic evidence, containers are designed to be short-lived and can be easily destroyed or replaced. This means that it is important to act quickly and gather evidence as soon as possible when responding to an incident in a containerized environment.

Docker forensics require specialized knowledge, due to the unique nature of containers - their portability, layered architecture, and the interactions between containers, host systems, and networks.

B. File System

Conducting forensic analysis in Docker environments, compared to conventional systems, involves similar steps, such as obtaining logs from the hard drive and main memory. However, it is crucial to recognize that this analysis may become incomplete if we do not take into account the peculiarities of Docker containers. [6]

Searching for files in a host system disk dump follows the traditional process. Data inside docker Containers is accessible transparently to the Host machine however, in Docker environments, there are additional considerations to address:

1. Identify the image that originated a specific file.
2. Track the containers that used the file in question.
3. Verify that the file is deleted at the container level, taking into account potential differences from deletions in conventional file systems.

To answer these questions, we need to understand that an image can be shared across multiple containers using

the read/write (r/w) layer. The association between files and containers is only possible through runtime information and, when available, through specific configuration files.

Core container runtimes use a copy-on-write file system. This approach is advantageous for forensic analysis because all standard system and application files are contained in the container image. Changes made since the container started are stored in a separate directory from the original image. Additionally, any file deletions made to the original image are logged. In the context of Docker, it is possible to manually explore copy-on-write differences within the directory `"/var/lib/docker"`. This ability to examine changes in isolation is valuable for forensic analysis, allowing investigators to understand the activities and modifications made to the container environment over time.

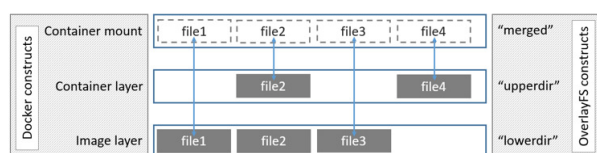


Fig. 9. Overlay File System

The **Overlay File System** currently adopted by the Docker Community is a successor to the **AUFS (AnotherUnionFS)**. Although very similar, some nuances exist, such as the number of Layers in both systems and Page Caching feature in the Overlay File System. This document only discusses the Overlay File System, as it is currently the file system supported and used by Docker Community and Maintainers. Older Docker Environments may still use the AUFS and the differences between the two systems must be taken into consideration when conducting forensic analysis. [6]

1) *Volumes*: Containers are designed to be ephemeral, meaning their existence is temporary, and they are designed to be easily replaceable. However, even though sensitive data is ephemeral, it can be written to Volumes. Identifying volume mounts on existing containers can be done by inspecting the container. These are identified via a Volume and Mount configuration.

Achieving a comprehensive forensic analysis of the file system mandates direct access to and proficient comprehension of the structure of the integral container file system integral. It necessitates an intricate understanding of the existing volumes and their specific container associations.

Notably, container escapes and potential exploits, particularly in cases involving malware or cyberattacks, can be facilitated via volumes as these can be shared across containers.

Determining the originating docker container responsible for modifications to a shared volume can also pose significant complexity during investigation processes.

C. Memory

Containers offer additional benefits when it comes to memory isolation. In the context of Linux, they are not a first-class primitive, this is they cannot exist independently of the host system; instead, they represent an abstraction that uses several subsystems, such as process namespaces and *cgroups*. Processes running inside a container manage memory the same way as any other process. Capturing all memory for each process in a container would be equivalent to capturing all memory allocated to that container. In Linux, there are several utilities for performing memory dumps on a per-process basis. For example, the *gcore* command suspends the process during acquisition without modifying it, being compatible with the **YARA tool** (a tool aimed at malware researchers). Other alternatives include using tools like *gdb*, *pidump*, *LiME*, or even the *dd* command directly on memory devices.

1) *Processes PID's*: Log files that include Process IDs (PIDs) are crucial for post-mortem analysis. However, translating container PIDs to host PIDs is impossible without real-time information. Host PIDs are always unique, while only the container's PID might appear identical to a PID on the host.

Similarly, user namespaces, like PID namespaces, enable the mapping of User IDs (UIDs) or Group IDs (GIDs) from one container to another UID on the host. For instance, a process running with UID 0 inside a container might correspond to a process on the host with UID 65000. This difference in UIDs between container and host poses challenges during analysis if log files contain UIDs. [6]

Attempting memory and process analysis solely from the host could lead to crucial information loss. Conducting memory forensics within the container preserve the context and integrity of processes, offering a more accurate representation of the data.

D. Post-Mortem Analysis

In instances where runtime analysis of the Docker environment is unfeasible, the ability to extract data from the host file system becomes crucial. When examining disks hosting Docker environments, thorough exploration of configuration files housing critical details pertaining to the system's containers is imperative for reconstructing the environment accurately. Docker repository configurations are stored within the `/var/lib/docker` directory. To specifically access details pertaining to a singular container, reference the path `/var/lib/docker/container/ContainerID`, where *ContainerID* serves as the unique identifier. [6] Identification of a running container can be determined by two key indicators:

- 1) Verification through the container's `config.v2.json` file, where the attribute `"Running: true"` signifies its operational state.
- 2) Recognition of a distinct directory allocated for the read/write layer, indicating a container's prior execution.

The determination of a file's origin within a container—whether it derives from the read/write (r/w) layer or

the image layer— is also integral in evaluating file visibility during a container’s execution. Another important aspect is removing a file within the container, it leads to two distinct outcomes:

- Deletion of files from the read/write (r/w) layer follows standard operating system protocols.
- Files originating from an image layer result in a deletion reference within the read/write (r/w) layer, while retaining their presence within the image layer. Understanding these distinctions is vital for effective file management and container security.

In the second case, recovery of deleted files is trivial, as they still exist within the Docker Image. Overlay2 designates a directory within the read/write layer, attributing it with the deleted file’s name and indicating its status with a particular flag (“0/0”)[1]. In the first case, the deletion of a file within the read/write layer of a container, potential recovery avenues exist via the *scrapping* of system memory. The techniques are not different from the ones conducted on physical disks or virtual disks. They include *file carving* and *file system tables analysis*.

File carving involves scanning through a volume, disk image, or file and identifying special patterns (called magic bytes) that signal the start and/or end of files. This method, regardless of the file system, enables the retrieval of both current and deleted files that haven’t been replaced. Distinguishing whether a file comes from a Docker container or the host system relies on metadata. However, files recovered by carving usually lack this information, making it impossible to assign them to the host machine or a container.

File system analysis relies on utilizing file management structures inherent to the respective file systems. These structures include components, such as the *MFT* (Master File Table) within *NTFS* or the *inode* tables situated within group descriptors specific to the *Extended file systems* [2].

V. A REMOTE RUNTIME DOCKER FORENSICS TOOL

A. State of the Art

In the current landscape of Docker forensics tools, there are several advancements in both static analysis and post-mortem investigation. Tools such as Clair, DockerScan, and Anchore are employed for static analysis of container images, enabling the detection of potential vulnerabilities and security risks. Meanwhile, Docker-Forensics-Toolkit stands out as a comprehensive tool specialized in post-mortem analysis specifically tailored for Docker environments.

Monitoring systems like Prometheus and Grafana [5] utilize the Docker API as a foundation, serving as powerful tools for monitoring and time series databases. However, while Prometheus excels in monitoring capabilities, it lacks specific features tailored for forensic analysis. [9]

To address this gap, the started the development of a Docker API-based tool tailored specifically for forensic purposes. This tool aims to enhance the forensic investigation capabilities within Docker environments.

B. Goal

The need for a Runtime Docker Forensics Tool (RDFT) is paramount due to several critical challenges inherent in containerized environments, as described previously. Containers’ transient nature allows easy creation, destruction, and recreation, often resulting in the loss of critical evidence before investigations can begin. Docker’s dynamic networking and IP addressing further complicate investigations by making it arduous to track and analyze network communications. The usage of docker in Cloud-based distributed environments poses a challenge, demanding tools adept at accessing distributed logs and correlating events across multiple host machines with varying operating systems.

The development of a Remote Runtime Docker Forensics Tool, designed to remotely capture volatile data within a Docker environment, holds significant utility in the following aspects:

- Extraction of real-time runtime data for in-depth analysis.
- Facilitating the collection of forensic evidence is crucial for conducting comprehensive security investigations.

C. Architecture and Components

The Docker Runtime Forensics Tool operates within the host machine of containers. Developed using Flask, a Python-based micro web framework, this tool interacts with the Docker Engine through the Docker API. By leveraging this API, the tool accesses data efficiently through a well-defined data pipe, minimizing system strain.

To utilize this service, specific ports must be opened within the Docker environment. This facilitates access to the tool and forensic data by a forensics analyst over TCP/IP through straightforward HTTP requests via a web browser. This approach enhances security compared to directly exposing the Docker API and facilitates gathering forensic data from Docker Environments. [14]

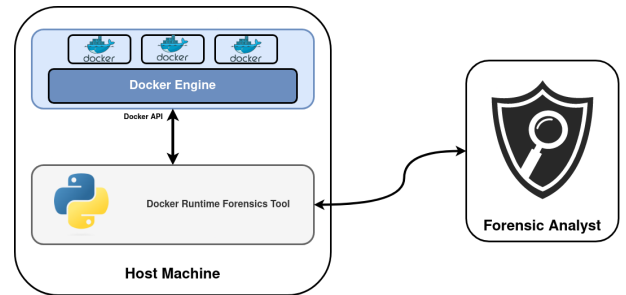
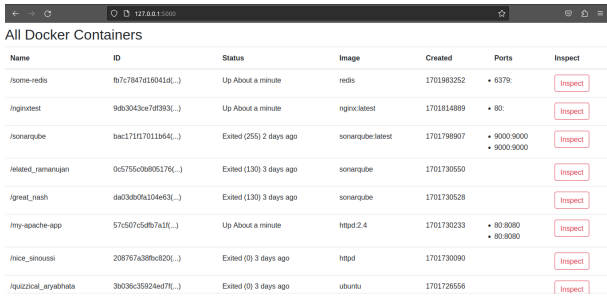


Fig. 10. Architecture

D. Use Cases

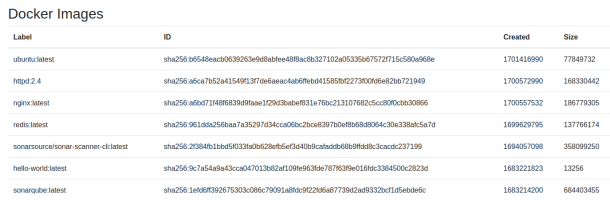
- 1) Container Enumeration Identify and list all Docker containers present on the host machine. List the unique ID, name, status, date of creation (in Unix time), and network ports.



Name	ID	Status	Image	Created	Ports	Inspect
/some-redis	fb7c7847d16041d6...	Up About a minute	redis	170198252	• 6379	Inspect
/nginx-test	9db3043bc7d93b...	Up About a minute	nginx:latest	170181489	• ID:	Inspect
/sonarqube	bac17117011b64...	Exited (255) 2 days ago	sonarqube:latest	170179897	• 9000:9000 • 9000:9000	Inspect
/redis_ramanujan	0c5755c3b005176...	Exited (130) 3 days ago	sonarqube	170173050		Inspect
/great_nginx	d603b0b104e63...	Exited (130) 3 days ago	sonarqube	170173058		Inspect
/my-apache-app	57c507c50b7a2b...	Up About a minute	httpd:2.4	170173023	• 80:8080 • 80:8080	Inspect
/nice_sinuous	208767a38b3d20...	Exited (0) 3 days ago	httpd	170173090		Inspect
/qazwxcz_oryadata	36d39c35924ed7b...	Exited (0) 3 days ago	ubuntu	170172956		Inspect

Fig. 11. List containers

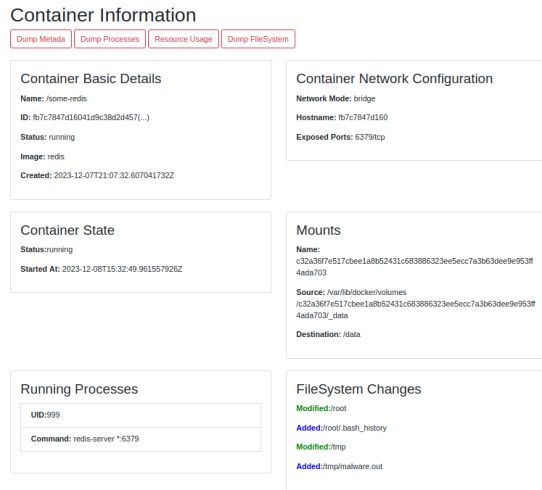
- 2) Image Inventory List and document all Docker images stored on the system. List names/labels, the sha256 of the image or image ID, the date of creation (in Unix time), size of the image in bytes.



Label	ID	Created	Size
ubuntu:latest	sha256:1a654beac0639263e4d01a1ee408ac8b327102a0535a675272715c580a909e	1701416990	77849732
httpd:2.4	sha256:a6ca7b52a41549137edfaeac4ab0fbb415859d2273000fae82b6721949	1700572990	188330442
nginx:latest	sha256:a6b0714808339af9a1f59c3b0b0f831e76bc213107682c5cd000b030966	1700507632	186779305
redis:latest	sha256:961d9a25bba7a35297d34cca09bc2bca8397b0efb8d89064c30a338a5a7d	1699629795	137766174
sonarsource/sonar-scanner-cli:latest	sha256:2f384b11b0f933b0b028b5ef3d40b9cabdd4b0b9d8c3acac237199	1694057098	358099250
hello-world:latest	sha256:9c7e54a943cca0470138d2af109e93b67676b99e0189ac3384500c2823d	1683223823	13356
sonarqube:latest	sha256:1e69f93267530c08679091a89c9226e6d7739c2e9332bc1f5e6bdefc	1683214200	684403455

Fig. 12. List images

- 3) Visual Representation Provide visual representations detailing relevant metadata, ongoing processes, and changes made to files within Docker containers comparing to the base image, and network information.

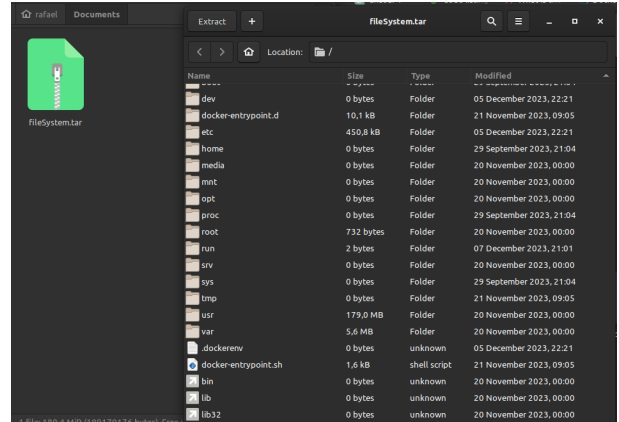


Container Information	
Dump Metadata	Dump Processes
Container Basic Details Name: /some-redis ID: fb7c7847d16041d638c2d4571... Status: running Image: redis Created: 2023-12-07T21:07:32.607041730Z	Container Network Configuration Network Mode: bridge Hostname: fb7c7847d160 Exposed Ports: 6379/tcp
Container State Status: running Started At: 2023-12-08T15:32:49.961557826Z	Mounts Name: c32a39f7e517dbee1a8b52431c683886332ee5ecc7a3b63dee9e953f4ada703a Source: /var/lib/docker/volumes/c32a39f7e517dbee1a8b52431c683886332ee5ecc7a3b63dee9e953f4ada703a_data Destination: /data
Running Processes UID: 999 Command: redis-server *:6379	FileSystem Changes Modified: /root Added: /root/.bash_history Modified: /tmp Added: /tmp/malware.out

Fig. 13. Docker Container metadata

- 4) Metadata Extraction Extract detailed metadata about a specific Docker container in JSON format, including its ID, associated image ID, creation command, configured ports, network settings, mounted volumes, and other pertinent information (see the *Attached Documentation* section)

- 5) Process Retrieval Retrieve and document all active processes running within a targeted Docker container in JSON format (see the *Attached Documentation* section)
- 6) Resource Utilization Snapshot Gather and retrieve real-time resource utilization metrics for a Docker container, such as I/O statistics, CPU count, CPU usage percentage, available memory, memory statistics, and network statistics per network interface in JSON format. (see the *Attached Documentation* section)
- 7) File System Dump Retrieve the entire file system within a Docker container in tar format



Name	Size	Type	Modified
dev	0 bytes	Folder	05 December 2023, 22:21
docker-entrypoint.d	10,1 kB	Folder	21 November 2023, 09:05
etc	450,8 kB	Folder	05 December 2023, 22:21
home	0 bytes	Folder	29 September 2023, 21:04
media	0 bytes	Folder	20 November 2023, 00:00
mnt	0 bytes	Folder	20 November 2023, 00:00
opt	0 bytes	Folder	20 November 2023, 00:00
proc	0 bytes	Folder	29 September 2023, 21:04
root	732 bytes	Folder	20 November 2023, 00:00
run	2 bytes	Folder	07 December 2023, 21:01
srv	0 bytes	Folder	20 November 2023, 00:00
sys	0 bytes	Folder	29 September 2023, 21:04
tmp	0 bytes	Folder	21 November 2023, 09:05
usr	179,0 MB	Folder	20 November 2023, 00:00
var	5,6 MB	Folder	20 November 2023, 00:00
dockerenv	0 bytes	unknown	05 December 2023, 22:21
docker-entrypoint.sh	1,6 kB	shell script	21 November 2023, 09:05
.bin	0 bytes	unknown	20 November 2023, 00:00
.lib	0 bytes	unknown	20 November 2023, 00:00
.lib32	0 bytes	unknown	20 November 2023, 00:00

Fig. 14. Dump of a Docker Container file system

E. Limitations and Future work

Retrieving processes from a running Docker container is distinct from the process of memory dumping on a machine. While it is feasible to extract processes from a live Docker environment, this action differs fundamentally from the comprehensive memory dump procedure executed on a physical or virtual machine.

Obtaining memory dumps solely from the host machine lacks critical runtime information necessary for correlating processes within the container to those on the host. The current optimal solution resides in extracting memory from within the Docker container. This gives us the ability to effectively track and analyze malicious activities occurring within the container environment. One potential avenue for internal memory analysis involves utilizing tools like **Volatility** from within the container. [11] The tool developed lacks this essential feature for a comprehensive forensic analysis.

Filtering and grouping containers by Docker Volumes, although possible with the tool implemented, could be made an automated feature and make the process of identifying shared volumes between containers easier.

VI. CONCLUSION

Due to these benefits, containers (& Docker) have seen widespread adoption. Companies like Google, Facebook, Netflix, and Salesforce leverage containers to make large engineering teams more productive and improve compute resource

utilization. Google credited containers for eliminating the need for an entire data center[10].

The transformative shift from traditional Virtual Machines (VMs) to containerized environments, particularly facilitated by Docker, has revolutionized the landscape of software deployment and development. While containers offer unparalleled benefits in terms of lightweight and efficient cloud workload management, they have also introduced significant challenges in the realm of digital security, necessitating innovative approaches to forensic analysis.

Throughout this exploration, we delved into the unique challenges arising from the transitory characteristics of containers, emphasizing the recovery of crucial artifacts. By analyzing best practices, identifying common obstacles, this work aims to provide a comprehensive understanding of the complex landscape of forensic analysis in Docker Container environments. In doing so, it addresses the particularities and nuances involved in securing containerized workloads, ultimately contributing to the ongoing efforts to enhance the digital resilience of modern cloud-based systems.

In the dynamic landscape of Docker forensics tools, advancements in static analysis and post-mortem investigation have significantly enhanced security measures. While existing tools excel in monitoring and analysis, there's a noticeable gap in specialized forensic capabilities tailored explicitly for Docker environments. The development of this Docker API-based forensic tool addresses this void, aiming to bolster investigative potential within Docker ecosystems and facilitate the process of forensic analysis in docker environments.

REFERENCES

- [1] Neil Brown. 2023. URL: <https://docs.kernel.org/filesystems/overlayfs.html>.
- [2] Brian Carrier. "PERFORMING AN AUTOPSY EXAMINATION ON FFS AND EXT2FS PARTITION IMAGES An Introduction to TCTUTILs and the Autopsy Forensic Browser". In: ().
- [3] E. Casey. *Digital Evidence and Computer Crime*. Elsevier Science, 2004. ISBN: 9780121631048. URL: https://books.google.pt/books?id=Xo8GMt_AbQsC.
- [4] Luca Caviglione, Steffen Wendzel, and Wojciech Mazurczyk. "The future of digital forensics: Challenges and the road ahead". In: *IEEE Security & Privacy* 15.6 (2017), pp. 12–17.
- [5] Prometheus Community. 2023. URL: <https://prometheus.io/docs/introduction/overview/>.
- [6] Andreas Dewald, Matthias Luft, and Julian Suleder. *Ernw White Paper 64/ (02, 2018) INCIDENT ANALYSIS AND FORENSICS IN DOCKER ENVIRONMENTS*. Feb. 2018. URL: https://static.ernw.de/whitepaper/ERNW_Whitepaper64_IncidentForensicDocker_signed.pdf.
- [7] Wei Min Ding, Benjamin Ghansah, and Yan Yan Wu. "Research on the virtualization technology in cloud computing environment". In: *International journal of engineering research in Africa* 21 (2016), pp. 191–196.
- [8] Inc Docker. "Docker". In: *linea*. [Junio de 2017]. Disponible en: <https://www.docker.com/what-docker> (2020).
- [9] Delu Huang et al. "Security Analysis and Threats Detection Techniques on Docker Container". In: *2019 IEEE 5th International Conference on Computer and Communications (ICCC)*. 2019, pp. 1214–1220. DOI: 10.1109/ICCC47050.2019.9064441.
- [10] Hyeon Seung Kim and Sang Jin Lee. "Method of Digital Forensic Investigation of Docker-Based Host". In: *KIPS Transactions on Computer and Communication Systems* 6.2 (2017), pp. 75–86.
- [11] Wilson Mendes. *Forensic investigation in Docker Environments: Unraveling the secrets of containers*. Aug. 2023. URL: <https://eforensicsmag.com/forensic-investigation-in-docker-environments-unraveling-the-secrets-of-containers/>.
- [12] Keyun Ruan et al. "Cloud Forensics". In: *Advances in Digital Forensics VII*. Ed. by Gilbert Peterson and Sujeet Sheno. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 35–46. ISBN: 978-3-642-24212-0.
- [13] Bradley Lawrence Schatz. "Digital evidence : representation and assurance". PhD thesis. Queensland University of Technology, 2007. URL: <https://eprints.qut.edu.au/16507/>.
- [14] Jie Xiang and Long Chen. "A Method of Docker Container Forensics Based on API". In: *ICCSP 2018: Proceedings of the 2nd International Conference on Cryptography, Security and Privacy* (Mar. 2018), pp. 159–164. DOI: 10.1145/3199478.3199506.

VII. ATTACHED DOCUMENTATION

A. Docker Container metadata format

- Id - string
- Created - string (The time the container was created)
- Path - string (The path to the command being run)
- Args - Array of strings (The arguments to the command being run)
- State - object or null (ContainerState) - stores container's running state. It's part of ContainerJSONBase and will be returned by the "inspect" command.
- Image - string The container's image ID
- ResolvConfPath - string
- HostnamePath - string
- HostsPath - string
- LogPath - string
- Name - string
- Restart count - integer
- Drive - String
- Platform - String
- MountLabel - String
- ProcessLabel - String
- AppArmorProfile - string
- ExecIDs - Array of strings or null
- HostConfig - object (Container configuration that depends on the host we are running on)
- GraphDriverData - object (Information about the storage driver used to store the container's and image's filesystem.)
- SizeRw - integer (the size of files that have been created or changed by this container)
- SizeRootFs - integer (The total size of all the files in this container.)
- Mounts - Array of MountPoint objects
- ContainerConfig - object (Configuration for a container that is portable between hosts.)
- NetworkSettings - object

B. Docker Running Processes

- Titles - Array of strings (The ps column titles)
- Processes - Array of strings[items] - (Each process running in the container, where each process is an array of values corresponding to the titles.)

Example:

```
{
  "Processes": [
    [
      "999",
      "8168",
      "8143",
      "0",
      "15:32",
      "pts/0",
      "00:00:02",
      "redis-server *:6379"
    ],
    [
      "root",
      "10636",
      "8143",
      "0",
      "15:46",
      "pts/1",
      "00:00:00",
      "bash"
    ]
  ],
  "Titles": [
    "UID",
    "PID",
    "PPID",
    "C",
    "STIME",
    "TTY",
    "TIME",
    "CMD"
  ]
}
```

C. Docker resource usage statistics

- blkio_stats - array of objects (bulk Input/Output statistics)
- cpu_stats - array of objects (cpu_usage statistics)
- id - string (container ID)
- name - string
- networks - array of objects (network interfaces information)
- precpu_stats - object
- preread - timestamp
- read - timestamp
- storage_stats - object