


RESEARCH

Open Access



CONTAIN4n6: a systematic evaluation of container artifacts

Anand K. Mishra, Emmanuel S. Pilli*  and Mahesh C. Govil

Abstract

A container provides an environment where applications are packaged and run with the supporting libraries and dependencies. Due to scalability and efficient software deployment, the popularity of container technology has increased and its services are also available on cloud platforms. The container environment is prone to a variety of threats and vulnerabilities that lead to security breaches and attacks. Investigation is required to analyze the attack and the digital forensics processes have also been implemented in the container environment. In this paper, we present a systematic evaluation of container artifacts. An interface named CONTAIN4n6 is developed to collect data from container environment that extracts the data using introspection libraries, container file systems, and is also capable to trace the system call of running container. The functionality of system calls traces is implemented in an open source containerization software, i.e, Moby project. Container's artifacts are associated with environmental information, log files, directories, link files, repositories, etc. Data collected from multiple sources are stored in a database and created a hash values to maintain the integrity of collected data. A case study of privilege escalation attacks has been demonstrated which is used to validate the data collection tool, called, CONTAIN4n6. Research challenges associated with security and forensic investigations on containerized applications are also presented.

Keywords: Container, Security, Investigation, Logging, Forensics

Introduction

Containerization creates an abstraction layer over the operating system. In contrast, server virtualization creates an abstraction layer over computer hardware, where multiple virtual machines can run as a simulation of a physical computing machine. A container is a standardized unit of software that packages up code and all its dependencies, so the application runs quickly and reliably from one computing environment to another [1]. In server virtualization, the virtual hardware resources of a Virtual Machine (VM) such as processor and memory are managed by a software called the Hypervisor. Similarly, the OS kernel sharing in a containerized environment is operated by container management software. Also, large scale deployment of containers requires a class of software called container orchestration software. Examples of

orchestration software include Kubernetes Engine [2] and cloud-based platforms such as Amazon Elastic Container Service (Amazon ECS) [3].

NIST (2016) [4] defines application container as “a construct designed to package and execute an application or its components running on a shared Operating System”. Files, environment variables, and libraries are the main components of containers. Application containerization gains efficiency for memory, CPU, storage, and portability, but a potential drawback is the lack of isolation from the core OS. Since application containers are not abstracted from the host OS, security threats have easier access to the entire system. Thus, it requires monitoring the system with proper tamper-proof logging of the events and close inspection of the host and the application containers.

Motivation and contribution

Security practitioners and researchers are working to secure the container environment from attacks that launch malicious process and finally result into service

*Correspondence: espilli.cse@mnit.ac.in

Department of Computer Science and Engineering, Malaviya National Institute of Technology, Jaipur, India

unavailability. There is a need not only to protect the container environment but there is also a need to investigate the attacks. Container forensics is an approach that attempts to investigate and analyze container security threats. It will ensure that attackers will be more cautious to avoid prosecution for their illegal actions. It acts as a deterrent, reducing network crime rate and improving security. Besides protecting the container environment, we need to focus on this issue. Unfortunately, there has been little research on a framework for digital forensics container environments. We are addressing this problem, which will help to collect the evidence from container environment, analyzing them, and finally performing attribution of attack.

Several incidents of container security breaches have been reported in many container application scenarios. These events arise for many reasons, such as exploitation of access control, vulnerabilities of container images, privilege escalation, etc. A systematic evaluation of container's artifacts is required to identify such events. Data collection in a container environment is challenging due to containers' volatile nature and distribution of containers at multiple hosts. The objective of this paper is to describe a methodology for the container's data collection and to find evidentiary values. The salient features of this methodology are:

- 1 Container's environmental information is collected using introspection to find correlation of attack data.
- 2 Data collection of various objects (container, images, storage driver, etc.) of containerization platform to examine the evidentiary values.
- 3 A system call trace functionality is enabled in an open source project of Docker, called, Moby project. The sequence of system calls is analyzed to detect malicious processes.
- 4 A data collection interface was implemented on a container-based platform. It demonstrates the detailed information collected from docker introspection, container's files and folders, and system calls.

Docker environment

For a systematic approach to extract the data from a container environment, we have used Docker [5]. Docker is a client-server application where the server is known as Docker daemon, and the client-side is Docker CLI. Docker CLI interacts with Docker Daemon using REST API as shown in Fig. 1. Docker is written in GO programming language [6]. Linux kernel features have been used in Docker Engine to provide container services as follows: 'namespaces' for isolated workspace, 'control groups' for a specific set of resources, 'union file systems' for layering the Docker images, and 'libcontainer' for container formatting. Docker images contain all necessary data and info needed to create a group of processes with defined properties. Containers are created from Docker images and run the actual application. The Docker CLI's request to Docker Daemon to run an image and Docker Hub is a repository of Docker images.

The rest of this paper is structured as follows: "Security and vulnerability issues in container" section discusses the security and vulnerability issues in container system. "State of the art" section discusses the related work of container forensics. "CONTAIN4n6: architecture for container forensics" section presents CONTAIN4n6 and its components. "Introspection of docker objects - image and container" section demonstrates the data collection process using introspection methods. "Directories and files" section explain the artifacts from the directories and files of container. "System calls traces" section extract the system call traces of container. "Research challenges" section discusses the research challenges associated with security and forensics of container environment. Finally, "Conclusion and future work" section presents the conclusions and future work.

Security and vulnerability issues in container

The National Institute of Standards and Technology (NIST) [7] has provided a catalog of two hundred twenty four security and privacy controls for federal information systems and organizations and a process for selecting controls to protect organizational operations. These

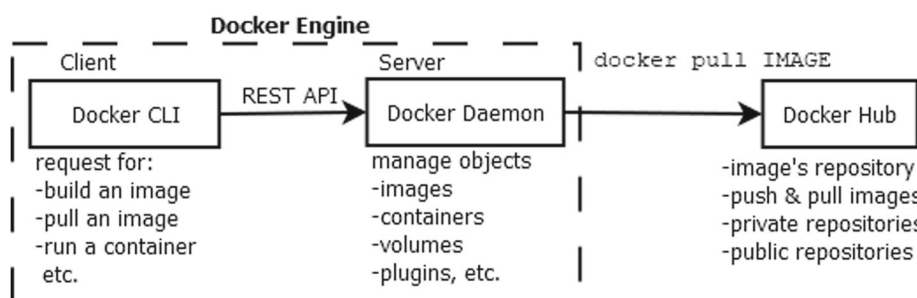


Fig. 1 An overview of Docker Engine

controls are access controls, audit and accountability, security assessment and authorization, identification and authentication, incident response, personnel security, risk assessment, system and communications protection, and system and information integrity, etc. Open Security Controls Assessment Language (OSCAL) [8] is attempting to address several challenges around security controls and security control assessment developed by NIST.

NIST [9] has published a guidance on application container security outlining the challenges and recommendations for addressing these challenges. Container technology risk is described at five major levels which are image risk, registry risk, orchestrator risk, container risk, and host OS risk. Recommendations to counter the attacks due to exploitation of these risks are: (a) using container-specific host OS to avoid large attack surfaces, (b) using separate hosts for different groups of containers, (d) adoption of container-specific vulnerability management tools, (e) the use of hardware-based countermeasures, and (f) the use of container-aware runtime defense tools. Five phases have been discussed for the security of container technology- initiation phase, planning and design phase, implementation phase, operations, and maintenance phase, and disposition phase.

Containers share the same host kernel, which can be the single point of failure for system breakout. Mouat [10] has suggested some mechanism to secure containerized environment such as running containers in virtual machine (VM), using minimum resources (process not running in container as a root, enabling file-system as read only etc.), using separate Docker host in multi-tenancy environment, image labeling, avoiding unsupported drivers, image provenance (cryptographic signing), reproducible and Trustworthy Dockerfiles, running regular auditing, incident response (Ex. Docker diff, logs, commit). Mouat has discussed the following challenges to Docker security such as kernel exploits, denial of service attacks, container breakouts (privilege escalation attacks), poisoned images (host and data are at risk), and compromising secrets. A list of security mechanism for containerized environment is also suggested such as running containers in a virtual machine (VM), using minimum resources, using separate Docker host in a multi-tenancy environment, image labeling, and image provenance (cryptographic signing), reproducible and Trustworthy Dockerfiles, and running regular auditing, and incident response.

Reshetova et al. [11] have discussed the security issues and solutions of OS-level virtualization. An attack model is discussed in a containerized environment that results in unauthorized data access, control flow error, denial of services, and privilege escalation. Feature of container-based OS (FreeBSD, Linux-VServer, OpenVZ, etc.) is described along with its container management capabilities. Security requirements of container technology are discussed in

detail, including isolation of process, file system, network, and devices. This study highlights the critical challenges of the container ecosystem from a security perspective that need a solution.

Bui [12] has presented a study on Docker internal security and its Linux based security features. Denial of service and privilege escalation attacks are discussed and their countermeasures such as isolation of process, filesystem, and device, limiting inter-process communication and network access, and finally specifying limits on the usage of resources are described. As Docker is based on the Linux system, Bui's study also includes Linux features. Features of Linux security products such as SELinux and AppArmor are also discussed as these features are built into the Linux.

Combe et al. [13] have presented an overview of Docker, its functionality, and security challenges. The authors explained the base of Docker security as isolation of processes, kernel security modules, and network security. Challenges of Docker containers discussed include insecure local configuration, malicious images, and weak local access control.

Vulnerability issues

A detail of container vulnerabilities can be found in the National Vulnerability Database (NVD) [14] that can be categorized on the basis of various severity level using the Common Vulnerability Scoring System (CVSS). These vulnerabilities can be further categorized in various types such as: Path traversal, Code injection, Unauthorized modification, Bypassing user authentication, Improper input validation, Deserialization of untrusted Data, Data processing error, etc. Below, the study on the container vulnerability are discussed:

Gummaraju et al. [15] have studied the Docker Hub images and found security vulnerabilities. Official images are analyzed to examine the severity level found in Debian packages, OpenSSL, Ubuntu repositories, etc. Non-official Docker Hub images are also analyzed and seen a higher number of vulnerable images. The authors have discussed the vulnerabilities that result in significant security threats such as privilege escalation and container breakout. Solutions are suggested to scan the images, run them into a virtual machine, and rebuild the image from scratch. Official images need to be updated regularly to remove redundant layers to enhance the Docker ecosystem's security mechanism.

Mostajeran et al. [16] have proposed a vulnerability analysis and risk assessment model to increase the security strength in container based cloud environment. Authors have focused on the container's image vulnerability analysis to assess the risk factor involved in the container ecosystem. Docker images such as NGINX, tomcat, and linux packages are analyzed to find the type of attack

and its base score. Based on these scores, risk factor is calculated on the scale of 0 to 10. Vulnerable images should be assessed before its download on host system so that containers could be run in safe mode and avoid attacks such as privilege escalation.

Martin et al. [17] have presented a detailed study on the Docker ecosystem's vulnerability. Authors have discussed the various aspects of container environment such as a comparative study with virtualization technology, unikernel runtime models, supported libraries, Linux containers, dependencies. The strength and weaknesses of Docker containers are also discussed against possible attacks. The usability of Docker Swarm is also presented along with Amazon ECS and Kubernetes orchestration. Vulnerability analysis is studied at various Docker ecosystem components such as insecure system configurations, vulnerable Docker image distribution, Linux kernel vulnerability, and maliciousness of Dockerfiles. The concept of Container-as-a-Service is also discussed in the cloud computing environment with its dependency on virtual machines and automation functionality. Security issues and the forensic aspect of the Docker ecosystem need to be identified and discussed in detail.

Zerouali et al. [18] have presented the study of a security vulnerability in Docker containers. The proposed model is used to analyze only Debian packages of Docker Hub images. Based on the vulnerability database's historical details and Debian Security Bug Tracker [19], vulnerability analysis is performed. Information such as version name and number, distribution type (testing, stable, old stable), and release date are extracted to compare the attributes available in the database. A bug report is also generated using the Ultimate Debian Database, which is being checked based on the version specification. The proposed study can be extended to other package distributions of Docker images.

Wenhao et al. [20] have discussed the architecture of Docker container and vulnerability issue in it. Docker vulnerabilities are studied in four categories: file system isolation, process and communication, device and host resources, and network and image transmission. The authors have presented Docker and kernel security features, including the network framework, integrity protection, access control, security enhancement mechanism, etc.

State of the art

In this section, we are presenting the summary of related work associated with container forensics.

Abed et al. [21] have presented a model to detect malicious behavior in the Docker ecosystem. An SQL injection attack has been launched to collect the system call for examination. Docker introspection method and `strace`

tool are used to manage the running container's environmental information and system call. True positive and false positive rates are calculated for the malicious detection technique. The proposed study can be applied to an intrusion detection system with additional functionalities such as monitoring and alert generation for malicious container process.

SANS ISC InfoSec [22] has explained to capture the process ID of a running the Docker system on an Ubuntu virtual machine. Each running container is a process that is assigned a process ID and that can be captured in RAM. After taking a snapshot of memory, it is analysed using Volatility Foundation Framework 2.4 [23]. The author has explained the process of PID extraction of Docker from *.vmem image of a virtual machine. Linux mount files and layered filesystem are investigated during this process and the recommendation is to run the system as a privileged user. Dirty "Copy on Write" and vDSO exploitation are described as the reason for Docker escape.

Winkel [24] has presented a framework for the forensic investigation of a Docker container escape attack. There are three modules in the proposed ELK framework-Elastic search for document extraction from the store, logstash as a log routing and management engine, and a web-based visualization tool "kibana." Container escape can be effective if the kernel is vulnerable or due to erroneous file configuration. Primary solutions for container escape have been suggested, such as image layer segregation and continuous system monitoring. Docker escape is implemented in the Linux kernel that results in the Dirty Copy on Write method. The NGINX instance is created to generate logs captured using the Kibana interface. DMLA architecture is presented that describes the monitoring, logging, and alerting function for Docker Host and containers. A vulnerable image (vDSO- responsible for container escape) is executed in the Docker environment as a case study examined using DMLA architecture to extract the log files system call traces. Docker introspection methods and the Host OS file system also provide evidentiary values that could be used in a proposed architecture.

Jian et al. [25] have presented a method to defend against Docker escape attack, which inspects namespace and detects malicious activities. Escape attack is explained by switching namespaces and modifying shared memories, which exploit virtual dynamic shared objects (vDSO). Docker introspection methods such as acquiring meta-data of Docker containers and namespace tags are used to detect malicious activity. Meta-data acquisition, status inspection, and response measures are used to inspect Docker container objects. Analysis of Docker objects (images, storage, etc.) is also required to understand the attack pattern and security solutions.

Stelly et al. [26] have developed a scalable real-time forensic framework named SCARF for forensic analysis. Container-based systems are used for data-parallel execution and low-cost extensibility of modules. Forensic tools such as ExifTool, OpenNSFW, Bulk Extractor, and Tika are tested in the container environment for metadata parsing, file streaming, text, and feature extraction. Tools are containerized using Dockerfile to automate the data operation process. This research shows that ExifTool's scalability, and Bulk Extractor, has better performance in the proposed architecture. A real-time investigation of the container-specific attack is required to examine the proposed model strength.

Dewald et al. [27] have presented a study to acquire data from Docker containers for forensics purposes using traditional approaches such as file recovering, file carving, and file system analysis. Docker commands are used for image and container introspection to extract environmental information. The authors have focused on recovering files of the Docker image layer, either deleted or corrupted. Host OS is examined to get the Docker related information such as directories and configuration files of image layers. Docker's file system (AUFS, overlay2) is reviewed to get the information of deleted image layers. This research attempts to acquire evidence from Docker host for forensic purposes.

Xiang et al. [28] have proposed forensics solutions for the Docker platform using Docker APIs. A discussion over Docker container forensics challenges has been provided: evidence volatility, evidence integrity, cross-platform, and cross host container forensics. Docker file system can be found at Host OS that provides Docker object (container, network, storage, images) details such as read/write layers, routing table, network address, etc. Docker APIs provide container and image information such as log files, host-name, and configuration files. Docker forensics architecture is proposed in which forensic agent modules receive and store the data to be examined by the investigator. "docker-py" library is implemented for evidence extraction. Investigation of an attack using the proposed model could be provided as a case study.

Lin et al. [29] have analyzed the attack parameters in a container environment and proposed a defensive mechanism against privilege escalation attack. An attack dataset is created that considers the information escape, and DoS attacks, etc. Attack has been launched on the vulnerable kernel and Docker engine using an exploited program that runs into the container. Authors have discussed container, kernel, and CPU's security and protection mechanism for privilege escalation attacks. A defense mechanism is provided to manage the ROOT privilege security measures of the container system. Further, the merits and demerits of the proposed model are discussed as model work

for ROOT access exploitation but need to improve against kernel-level exploitation.

Williams et al. [30] have discussed the Docker ecosystem's logging and alerting metrics. Authors have addressed the concept of microservices, containers, applications, and cluster nodes that is the basic unit of containerization. These units are monitored to debug operational issues. A requirement of an authentication server is mentioned in the route of microservices that is backed up with database server and HTTP routing. Alerting and visualization tools are mentioned in a container ecosystem that provides that also store and analyze the log files. Sysdig [31] tool is also discussed to analyze application container's metadata at the orchestration layer. Data introspection methods and log collection tools are discussed to capture the container data.

Lu et al. [32] have presented a model to detect temporary files in Docker images to avoid file redundancy. Build process (FROM, COPY, RUN) of Dockerfiles and image layers are analyzed to find the pattern of temporary files (TF). Static analysis such as state-dependence file matching, syntax determination, and verification method is applied to detect TF in Dockerfiles. Authors have also suggested eliminating the temporary files using the direct copy method, instruction merge, and external storage methods. This model has used manual checking method to examine the TF in Dockerfiles, that should be automated as Docker Hub has more than thousand image repository.

Awuson et al. [33] have proposed a trust model using Blockchain technology in a cloud computing environment. Hyperledger fabric Blockchain is used along with Docker engine that interacts with audit logger and membership managers. These modules manage the transaction logs and chaincode logs with Docker Host and communicate with cloud service providers (CSP), forensic investigators, and cloud customers. The authors also discussed concepts such as differences between container and virtual machine environments, cloud ecosystem, and digital forensics. A practical implementation is required as a proof of concept of the proposed model to be analyzed in real case studies.

Table 1 presents the comparison of existing solutions to our approach to evaluate container's artifacts systematically.

CONTAIN4n6: architecture for container forensics

In this section, we describe a systematic evaluation of container artifacts that is collected using introspection method, from directories, and system call trace. Collection of container artifacts are integrated in a monitoring tool called CONTAIN4n6 (Container Forensics). A graphical user interface (GUI) of this tool is implemented using

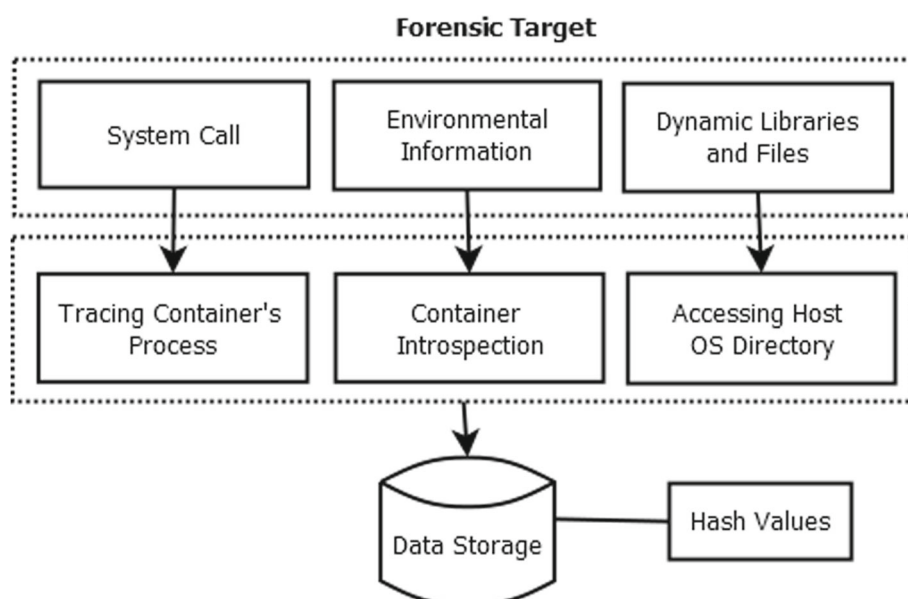
Table 1 Comparison of related work with this work

Reference	Platform	Attack Launched	Introspection	File System	System Call	Interface
Abed et al. [21]	Docker	SQL injection	Yes	No	Yes	No
SANS InfoSec [22]	Docker	Container escape attack	Only Process ID	No	No	No
Winkel [24]	Docker	Container escape attack	Only log file	No	Yes	ELK model
Jian et al. [25]	Docker	Container escape attack	Yes	No	No	No
Dewald et al. [27]	Docker	No	Yes	Partially	No	No
Xiang et al. [28]	Docker APIs	No	Yes	Yes	No	No
Lin et al. [29]	Docker	Privilege escalation attack	No	Partially	No	No
Williams et al. [30]	Docker	No	Yes (Sysdig tool)	Yes	No	No
Lu et al. [32]	Docker	File redundancy	Yes	No	No	No
Awuson et al. [33]	Docker	Trust based issue	Yes	No	No	No
CONTAIN4n6	Docker	Privilege escalation attack	Yes	Yes	Yes	Yes

python libraries for the Docker Engine API which manages Docker objects such as image, containers, volumes, etc.

Figure 2 provides an overview of the functional components and data flows of CONTAIN4n6. We have focused on three areas from where container related data can be collected for forensic purpose - (i) system calls of a running container (ii) environmental information of container system from Docker daemon (iii) dynamic data libraries and files from host OS. These data are stored in a database, and a hash value is also created for this database to maintain its integrity. These components of the CONTAIN4n6 architecture is described below:

- 1 Environmental Information using Container Introspection- A container-based system provides logs, and environmental information using introspection commands in Docker daemon. There are approximately fifteen management commands in the Docker engine to manage Docker objects. Some of these commands help to extract logs, contents from container's filesystem and low-level information to support the introspection of Docker objects. This information is further processed by an analysis engine to generate logs and reports.
- 2 Dynamic Data Libraries and Files- Information on Docker objects such as container, image, network,

**Fig. 2** Architecture of CONTAIN4n6

and storage driver can be extracted from the host operating system. Container forensics requires data from host OS which is generally stored as a file system, network packets, and memory dumps, etc.

- 3 Tracing System Call of a Container Process- A sequence of system calls can be analyzed to classify between malicious and non-malicious process. In an investigation of vulnerabilities, a knowledge base of attack can be collected from Common Vulnerabilities and Exposures (CVE), Remote Procedure Call (RPC), and signature of the attack. Along with the knowledge base, the effect of attack should also be known to understand the attacker's behavior.

Attack environment

A malicious Docker image (chrisfosterelli/rootplease) has been executed that can perform "privilege escalation" in Docker daemon. Containers can access the root shell on the host OS [34]. When the instance of Docker image is initiated, it loads a chroot into that volume. This instance's volume gets the access of root filesystem of the host machine. The Dockerfile of this image contains the command ["/bin/bash", "exploit.sh"]. This exploit.sh file calls the chroot() operation, that changes the root directory of the calling process to the specified path. The containers of chrisfosterelli/rootplease image have root access, that can perform malicious activities such as creating new files and directories, editing data and deleting the files and directories.

In next three sections, we are examining the collected data and correlating it with the evidentiary values and focusing to provide a systematic evaluation of container artifacts. Through some concrete questions which might occur during an investigation, we are attempting to answer them using extracted artifacts.

Introspection of docker objects - image and container

Docker commands are helpful to extract its object (image and container) data. This section is providing the answer of forensic related question that can be asked during investigation.

Question 1- What is the environmental information of Docker?

Docker info provides system-wide information that includes more than forty-five items of information of the Docker daemon, Docker objects, Host OS information, registries, Docker directory, containerd version, runc version, init version, and product license, etc. Details of Docker objects such as containers, images, plugins, volumes, etc. are provided in numbers with its status (running, paused, and stopped). The information of Docker host is extracted using docker info command shown in Table 2.

Table 2 docker info output

Host OS	Docker host
Kernel Version: 5.3.0-46-generic	Containers: 62; Images: 48
OS: Ubuntu 18.04.4 LTS	Logging Driver: json-file
OSType: linux	Cgroup Driver: cgroupfs ;
Architecture: x86_64	init version: fec3683
CPUs: 8	containerd version: 7ad184331fa3e5
Total Memory: 3.731GiB	runc version: dc9208a3303fee
Name: anand-XXXX-5040	Docker Root Dir: /var/lib/docker

Question 2- What is the name of the image repository and its ID? When was it created? What was the command used to run the script?

Docker images provides the list of images that are pulled from the Docker hub or any image at the local-host. It has five attributes which are repository name, the tag of image, image ID, creation time, and size of the image. A repository of image is found using docker image ls that are pulled from docker hub to gain root access shown in Table 3. Docker history command shows an output of Docker image history. It has five attributes including image ID, creation time, created by commands used, size of the image, and comment if any. Table 4 provides the history of command.

Question 3- What is the statistics of resources used by the container object?

Docker stats command provides the information statistics of container resource usage. It has eight attributes which are container ID, container name, host CPU in percentage used by the container, memory in percentage used by the container, total memory used and allowed limit of total memory use, data sent and received over network interface by container, read and write data on block devices, and processes created by the container. Docker top command is useful to extract information on the running processes of a container. docker stats extracts the statistics of resources used by container shown in Table 5.

Question 4- Can you find the object type and its current status?

Docker events command provides the real events (or activity performed) from the Docker daemon (or server). There are multiple options available to receive output from server such as output filtering, output formatting, and a specific timestamp for output. This command keeps monitoring the events of Docker objects and provides a detailed output which includes attributes such as

Table 3 Command: docker image ls

Repository	Tag	Image ID	Size
chrisfosterelli/rootplease	latest	0db941813769	188MB

Table 4 Command: docker history IMAGE_ID

Image ID	Created	Created by
0db941813769	3 months	/bin/sh CMD [/bin/bash exploit.sh]

timestamp, object type, the action performed, container ID, execution ID, image name, and container name, etc. `docker events` command does not require any container ID, but data can be captured in specific format, for example *.json format as shown in Table 6.

Question 5- What is the environmental information of image and container objects? What is the location of its directory?

Docker `inspect` command provides low-level information of Docker objects (images, containers, volumes, network, node, plugin). By default, it presents the output in JSON array format. There are multiple options available such as specific format and extracting specific values, container size, and specific return type. This command provides some useful information that can be used for further analysis of Docker objects. For a container object, the `inspect` command provides more than one hundred fifty information including container basic details, its process ID assigned by Host OS, image name and ID, related directories, host configuration details, drivers, and detail of network settings. Result of inspection of the Docker image and container is shown in Tables 7, and 8 respectively.

Other than above commands, there are more command which can provide valuable information such as `docker diff` command monitors the container's filesystem, and `docker export` can export the container's filesystem. Just like the `docker export` command, Docker `save` command is used to save the tar file of Docker images. This tar file is saved at home directory. Docker `logs` command shows the standard output of a container. Docker `ps` command shows the list of running containers. With an additional argument '-a', this command shows the list of all containers. There are multiple filtering options available to list in a specific format.

Directories and files

An analysis of Docker object such as i.e., container, image, and overlay2 storage driver has been performed. `chrisfosterelli/rootplease` directories and files have been accessed to analyse the data. Directory of Docker file systems can be found at `/var/lib/docker` location in the Ubuntu system by default. We found twelve folders with two hundred three items. These folders are

Table 5 Command: docker stats CONTAINER_ID

Container ID	Mem.	NET I/O	Block I/O	PID
2f0a35717b86	0.04%	4.75kB	3.41 MB	2

Table 6 Command: docker events --format '{{json .}}'

Status, Action, Object, Scope	pause, pause, container, local
ID and Image	2f0a35717b8604, chrisfosterelli/rootplease
Time	2020-04-22T20:13:14.476933724+05:30

builder, buildkit, containers, image, network, overlay2, plugins, runtimes, swarm, tmp, trust, and volumes.

Question 6- What information is stored in the container object directory? Can it provide log files and configuration settings?

Directory of container object is available at `/var/lib/docker/containers/` - container's log file (*.log), configuration file (*.json), host configuration file (*.file), hostname, hosts, resolved configuration file (*.conf), and hash of resolved configuration file (*.conf.hash). Docker container object shows two folders, and seven files shown in Table 9. Information of `config.v2.json` file and `hostconfig.json` file are similar to Docker `inspect` command.

Question 7- What information is stored in the image object directory? What storage driver has been used to store the image information?

"image" folder contains an image database, layer database, and list of repositories in JSON format. Another directory named `distribution` contains two subfolders which include digest code and metadata. Docker images may have multiple layers and layers of one image can be used in another Docker image. Details of image files can be found at: `/var/lib/docker/image/overlay2/`. Docker image object shows multiple files and folders which consist of valuable information such as a list of image repositories, environmental information, and various layers of Docker image shown in Table 10. "imagedb" directory holds the hash value of the Docker image. This file holds valuable information such as image layers. `docker inspect IMAGE_ID` also shows image layers of Docker image

Question 8- What information can be extracted from storage driver? Can you find the underlying layers of

Table 7 Command: docker inspect IMAGE_ID

Image ID	0db94181376938
Time	Creation time- 2020-01-22T23:46:17.826922425Z
Host, OS-Arch	3f37dbc61890, linux-amd64
Command	/bin/sh CMD [/bin/bash exploit.sh]
Storage Driver	overlay2 [/var/lib/docker/overlay2]
Lower directories; UpperDir MergeDir	0fc35e49c37e6d, 824a07f975d78e 1a0e693bdf8c92, 280c54be553be9 bf74be84df0510; 4a8f1e135fb92
RootFS : Layers	8ceab61e5aa8fe, 2b01876942154c 739d48883bb86d, 5f70bf18a08600 db674e90f639c9, 5f70bf18a08600

Table 8 Command: docker inspect CONTAINER_ID

ID, State, PID	2f0a35717b8604, running, 31773
Image ID	0db94181376938 (chrisfosterelli/rootplease)
Log	2f0a35717b86.json.log
N/k, Endpoint	e7ea9a20c60f2c, 9a7ebee5fba14a
Gateway and IP	172.17.0.1 and 172.17.0.2
Sandbox Key	/var/run/docker/netns/76c394fbb415
Lower Directory- /var/lib/docker/overlay2/	1222bd69f09956, 4a8f1e135fb924, 0fc35e49c37e6d, 824a07f975d78e, 1a0e693bdf8c92, 280c54be553be9, bf74be84df0510
MergedDir, UpperDir, WorkDir	/var/lib/docker/overlay2 5703fac0a10a6b
Env	PATH=/usr/local/sbin:/usr/local/bin
Cmd	"/bin/bash", "exploit.sh"
Mac Address	02:42:ac:XX:XX:XX

images? Are these information matching with introspection of Docker images? “overlay2” driver is a storage driver used by Docker daemon which is supported on kernel version 4.0 or newer. Directory is available at /var/lib/docker/overlay2. cache-id of layered folder hold the hash value, and this value is found in /var/lib/docker/overlay2 directory as a sub-folder which consist detail information of Docker image. Docker overlay2 object contains the details of the image layer shown in Table 11. These directories (D1, D2, D3, D4, D5, DCI1, DC1, DC1-init, DC2, DC2-init) are also mentioned in container, and image inspect. Image layers of Docker image also mention the name of these directories in cache-id. Table 12 shows the output of

Table 9 var/lib/docker/containers/CONTAINER_ID

Container files and folders	Information
*-json.log	"log": "You should now have a root shell on the host OS", "stream": "stdout", "time": "2020-04-27T12:34:13.510502855Z"
config.v2.json	"State": "Running", "Paused": false, "Restarting": false, "OOMKilled": false, "RemovalInProgress": false, "Dead": false, "Pid": 31773, "ExitCode": 255, "Error": "", "StartedAt": "2020-04-27T12:34:13.201814101Z", "FinishedAt": "2020-04-29T08:32:39.99711289+05:30",
hostconfig.json	"Binds": ["/:/hostOS"], "ContainerIDFile": "", "LogConfig": {"Type": "json-file", "Config": {}}, "NetworkMode": "default", "PortBindings": {}, "RestartPolicy": {"Name": "no", "MaximumRetryCount": 0}
hostname	2f0a35717b86
hosts	[127.XX.XX.XX : localhost], [172.XX.XX.XX : 2f0a35717b86]
resolv.conf	nameserver 172.XX.XX.XX
resolv.conf.hash	sha256: e942cd686d2a2d

Table 10 var/lib/docker/image/

var/lib/docker/image/overlay2/imagedb/content/sha256 Image ID: 0db941813769 (plain text document) various layers of image: diff_ids : sha256 Layer1: 8ceab61e5aa8fe Layer2: 2b01876942154c Layer3: 739d48883bb86d Layer4: 5f70bf18a08600 Layer5: db674e90f639c9	
var/lib/docker/image/overlay2/distribution/v2metadata-by-diffid/sha256 (plain text document) Layer1: 8ceab61e5aa8fe Digest: sha256: 2de59b831a235 Source Repository: docker.io/chrisfosterelli/rootplease Layer4: 5f70bf18a08600 Digest: sha256: a3ed95caeb02ff Source Repository: docker.io/chrisfosterelli/rootplease and docker.io/library/nginx	
var/lib/docker/image/overlay2/distribution/diffid-by-digest/sha256 (plain text document) 2de59b831a2357 sha256: 8ceab61e5aa8fe	
var/lib/docker/image/overlay2/layerdb/mounts Container ID of image 0db941813769: (directory) 2f0a35717b8604 init-id: 5703fac0a10a6b mount-id: 5703fac0a10a6b parent: sha256:537702ca7185c0	
var/lib/docker/image/overlay2/layerdb/sha256 (directory) Layer1: 8ceab61e5aa8fe cache-id: bf74be84df0510 diff: 8ceab61e5aa8fe size: 188104128 tar-split.json: payload NOTE: Layer2, Layer3, Layer4, and Layer5 are missing.	
repositories.json chrisfosterelli/rootplease:latest: sha256:0db94181376938	

link file (L1, L2, L3,..., L10) and **diff** directories. link file contains the name of the shortened identifier, whereas **diff** directory contains data of the layer. Table 13 shows the output of lower file which contains the information of link files of the second-lowest layer, and higher layer.

Other than above directories, *network* and *volumes* directories also provide valuable information such as: “network” directory is available at /var/lib/docker/network/files location. File name is “local-kv.db” also known as local key-value database which store metadata like network used, IP addresses, endpoint connections etc. “volumes” object is used to persist data in Docker. This object is used to store the container’s data either locally or on a remote host. By default volume is mounted in directory /var/lib/docker/volume/volume_name.

Table 11 Lower directories of image and container(s): var/lib/docker/overlay2/

image: chrisfosterelli/rootplease container1: 2f0a35717b86 and container2: bf79bc63a259	
Common directories of image and container(s) D1: 0fc35e49c37e6d3 D2: 824a07f975d78e D3: 1a0e693bdf8c92 D4: 280c54be553be9 D5: bf74be84df0510	
Common directories of container(s) of same image DCI1: 4a8f1e135fb924	
Additional directory of container1: 2f0a35717b86 DC1: 5703fac0a10a6b DC1-init: 5703fac0a10a6b-init	
Additional directory of container2: bf79bc63a259 DC2: 1222bd69f09956 DC2-init: 1222bd69f09956-init	

Table 12 var/lib/docker/containers/CONTAINER_ID/ diff and link

Dir.	Values in link file	diff directory
D1	L1: 5BNGLR76Q4IXP7	exploit.sh file
D2	L2: Z254T2TS7NAQ6F	empty
D3	L3: QNLTGGU53BMUWC	sources.list
D4	L4: 4LMHQA3NYLZMHN	etc, sbin, usr, var
D5	L5: Z53TJ66XAVA6AW	Linux file systems
DC1	L6: 7Z325R5PU5HQYX	empty
DC1	L7: KF4DWGXQ2SO7CF	linux FS, exploit.sh
DC1-I	L8: 7HQNC3JN26V2G3	dev; etc
DC2	L9: XZLXDRLDNN2VU2	hostOS
DC2-I	L10: ZNSDYPNYTDPGH3	dev; etc

System calls traces

The potential use of this implementation is: (i) to establish a correlation between program calls and associated system calls, (ii) monitoring system call sequences, and (iii) match attack signatures to system call sequences in the log. In the future, we would like to create tamper-proof storage of all logs and traces and analyzes the combination of program trace and system call for evidence of attacks.

The extraction of system call functionality is not available in Docker daemon. We have implemented this functionality using “strace” utility in the development environment of Docker Moby project. It is “a collaborative project for the container ecosystem to assemble container-based systems.” A detailed description of Moby project development can be found at [35].

We have added the system call trace functionality to the Moby project by the following steps:

- 1 We obtained the process ID (PID) of the running container
- 2 We provided this PID to system call trace utility
- 3 It records the system call used by a running container
- 4 These records are maintained in a database which are then hashed
- 5 We analyze the data to extract useful information

With the capture of system calls, our subsequent steps in the development of a forensic analysis framework are as

Table 13 var/lib/docker/containers/CONTAINER_ID/ lower file

Directories	Values in “lower” file
D1, D2, D3, D4	L2:L3:L4:L5, L3:L4:L5, L4:L5, L5
D5, DC1	null, L1:L2:L3:L4:L5
DC1	L8:L6:L1:L2:L3:L4:L5
DC1-init	L6:L1:L2:L3:L4:L5
DC2	L10:L6:L1:L2:L3:L4:L5
DC2-init	L6:L1:L2:L3:L4:L5

follows: To determine what data is required to match the signature and what is the footprint? What was the effect of the attack and whether we have enough data to detect that attack? What new fields and records are needed to detect the attacks? What other attacks/ variations can also be detected leveraging the additional data?

The system call of the running container’s process is captured. The output of introspection and dynamic libraries show that containers of chrisfosterelli/rootplease image have root access, which means this container can perform malicious activities such as creating new files and directories, editing data and deleting the files and directories. There are various types of system calls such as process control, file management, and device management etc. Various operations on linux files such as ls, mkdir, rmdir, touch are performed.

List 1 shows the trace of system call for command cd, and mkdir. The chdir(), read(), and write() calls indicate that the attacker changed the directory, and created a directory TESTabc. For chdir(), return value is zero, which means operation is successful. Return value of read() and write() calls are greater than one which means system call were executed successfully. Success of these system calls execution demonstrate that legitimate linux commands were executed from a container to host system. This complete process displays the privilege escalation via Docker.

In a Linux based system, a group is a collection of users, which provides a set of permission like read, write, or execute operations. In our implementation, we have added docker in group, which means docker has permanent, non-password-protected root access, and likewise docker containers. Though Docker has provided the guidelines to prevent privilege-escalation attack, it is still a manual process, and also a security issue [36].

Listing 1 Traces of system call- container’s process

```
read(0, “cd anand\n”,_8192)=_9
chdir(“/home/anand”)= 0
wait4(-1, 0x7fff5af8b98c, WNOHANG|WSTOPPED, NULL)= -1
ECHILD (No child processes)
write(2, “#”, 2) = 2
*****
read(0, “mkdir TESTabc\n”,_8192)=_14
stat(“/bin/mkdir”,{st_mode=S_IFREG|0755,
st_size=80056})= 0

clone(child_stack=NULL,
flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD,
child_tidptr=0x7f732e9d810) = 21

setpgid(21, 21) = 0

wait4(-1, [{WIFEXITED(s) && WEXITSTATUS(s) == 0}],
WSTOPPED, NULL) = 21
--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED,
si_pid=21, si_uid=0, si_status=0, si_utime=0,
si_stime=0}---
...
write(2, “#”, 2) = 2
```

Information obtained from collected data

After data collection, examiners analyze the data to understand the attack scenario and connect the events. Examiners attempt to find valid reasons of questions like who, what, when, where, and how. Significance of collected log files and metadata has to be understood on a case basis. In this attack scenario of privilege escalation attack, each relevant data has to be explained the malicious events when it was created, accessed, modified, received, sent, viewed, deleted, and launched. Information obtained from Tables 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 and 13 can be useful in following case scenario:

- Information obtained from Tables 3, 4, 5 and 6 using introspection method such as Image ID, Size of Image file, Creator, Container Statistics, Status and Action will help the examiners to identify the malicious image.
- Information obtained from Table 7 using introspection method such as Creation Timestamp, Location of Storage Driver, Root File System Layers will help to examiners to understand the lower layers of Docker image, its entry time to system, and location of its storage driver i.e. Overlay2 to get other details of its containers.
- Tables 8-9 help examiners to understand all of the containers created using malicious image. If these containers are in running state, a process ID is generated that can be used for further action either to remove the process or to analyze its system calls. Other information such as host name, container's log files, and environment path support the investigation process.
- From Docker image directory shown in Table 10, root file system layers is found that can be cross-examined with the information obtained from introspection method shown in Table 7. These image layers can be examined separately as it stores the changes compared to the image it's based on.
- From Docker storage directory shown in Table 11, all the container ID can be found that is created using malicious image along with common directories of image and its containers, and additional directories. These containers can be cross-examined with list of containers obtained from Docker introspection.
- From link files, lower files and diff directories, examiners can analyze working behavior of an image. For example, source list, exploit.sh files, Linux file systems etc.
- From system call analysis, examiners get the name of system call, arguments, return value etc. A sequence of system call is further analyzed to differentiate between normal and malicious process.

CONTAIN4n6 dashboard for container's artifacts

Additionally, a snapshot of the dashboard is shown in Fig. 3. Docker daemon provides a logging facility for containers and images, but these log files are not persistent. To enable persistent logging, we keep back up of log files in a log server so that the investigator can analyze these log files at any instantaneous time. With the dashboard which we have implemented for data acquisition from Docker Engine, we can observe the log files and we can check random errors, and the investigator can configure undesirable and abnormal activities. As Docker, objects log files are stored in the database as a backup file, so these data can be available for independent examination, statements, records, and analysis which is the part of auditing. An administrator can check the performance of the Docker Engine based on available data. At any instantaneous time, if the administrator or investigator is getting undesirable log entry, it can be taken as a quick defense mechanism to stop the Docker services and the system can be protected. Administrators can decide to defend the whole system by looking into available logs and stored files.

Comparing CONTAIN4n6 with other container forensic tools

At present CONTAIN4n6 is focusing on the collection of container's artifacts and maintain its integrity in the database. Evidentiary values can be extracted using container introspection method, and also from the container's directories. Additionally, tracing the system call of container process is also enabled in open source project of Docker, called, Moby Project. CONTAIN4n6 strengthen the Docker users to collect artifacts of containers outside of the Docker engine environment. Docker users do not need to install any Docker's image repository from Docker Hub. Most of the tools are implemented in such a way that it need to be downloaded from Docker Hub and to install within the Docker engine as another container. One of the example is Sysdig [31] tool - Sysdig Inspect container (<https://github.com/draios/sysdig-inspect>), that is required to install as a Docker container image. Though Sysdig Inspect container is open source and provide the environmental information and system call, there are issues such as - Inspect container itself may be compromised and deleted from the Docker environment. StackRox (<https://www.redhat.com/en/technologies/cloud-computing/open-shift/advanced-cluster-security-kubernetes>) also provide container security and forensic functionality but it is not open-source. Docker Explorer (<https://github.com/google/docker-explorer>) focus only on Docker filesystems.

Research challenges

During our research, we identified a set of challenges associated with security and forensics of container environment. These challenges are described below:

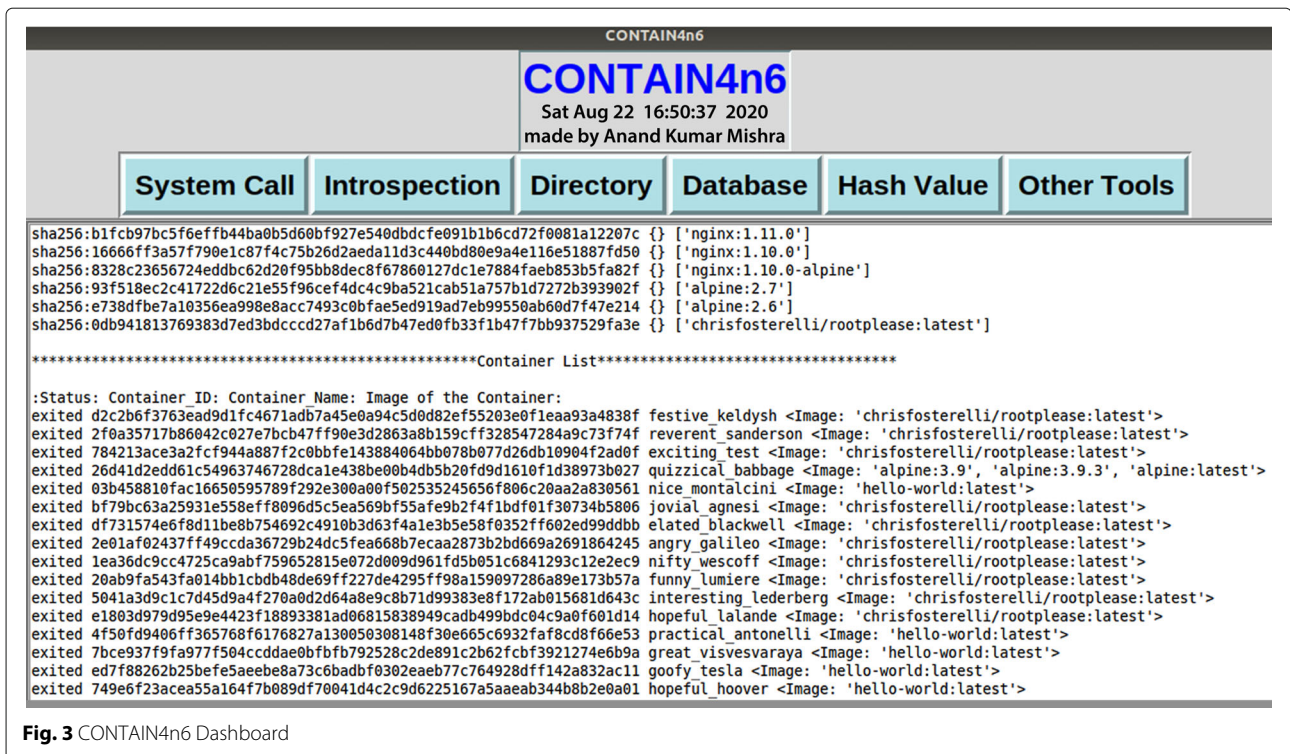


Fig. 3 CONTAIN4n6 Dashboard

- Data deletion in the application containers raises several challenges for users, security experts, and forensic investigators. As users are modifying their data pertaining to a containerized application, data at rest needs to be modified or deleted. If data at rest is deleted by marking the sectors storing it as available in the volume's map, then sectors will maintain information until overwritten. Even when overwritten, older information is still partially recoverable. This issue is challenging for users, particularly in multi-tenant environments since they would like the data to be completely shredded or zeroized. When data is deleted by marking the sectors storing it as being available in the volume's map but without proper zeroization or shredding of the data is difficult since it gets overwritten often by other user(s) in a multi-tenant environment and correctly identifying data provenance. In general, building the chain of custody is very difficult and often impossible from forensic point of view.
- Timestamp synchronization and event correlation: Synchronizing timestamps for log data collected from different containers operating on different hosts located in multiple geographical regions is challenging and constitutes a barrier to correlating forensic evidence. The challenge can be addressed by using a centralized time-server to create normalized timestamps for aggregated log data obtained from multiple containers. Creating a consistent semantics through a common log format is necessary to facilitate event correlation. Centralizing the collection and aggregation of log data from multiple sources leveraging normalized timestamps and common log format establishes a framework for integrated event correlation that also implements tamper-proof mechanism for all relevant forensic artifacts.
- Live Forensics and detection of the malicious act: Live Forensics involves collection and analysis of data in volatile memory when a malicious act was detected and is investigated. When conducting live data forensics, the processes used in data acquisition will result in changes to the system. To collect volatile evidence, the suspect computer must remain on, and the suspect operating system must be used to access the needed data. Implementing an outside triggering mechanism for application container forensic framework in order to generate volatile memory snapshots that are stored in the centralized secure location supports proactive approach to live forensics.
- Faulty configuration may circumvent container isolation: Faulty configuration of orchestration software may result in breaking the desired isolation among containerized applications in some hosts within a cluster. Faulty configuration also results in breaking the desired vertical isolation among containerized applications. Collecting and securely

storing snapshots of configuration data and performing analysis of the deviations from the desired secure configuration parameters may resolve the issue.

- Logging capability for orchestration layer: The orchestration layer creates key events such as assignment of application containers to various hosts/ platforms and migration application containers between these environments. Logging such events leveraging normalized timestamps and a common format is necessary. Forensic frameworks extension to support collection of event logs from orchestration layer that provide tamper-proof mechanisms for storing logs with normalized timestamps and a common format.

Conclusion and future work

With the widespread deployment of container based systems for microservices, it is important to develop and deploy architecture for container forensics to identify and prevent attacks. This paper explored the security and vulnerability issues of container-based environment and discussed the container forensics models. It also has presented the CONTAIN4n6 interface for a systematic evaluation of container's artifacts. Data is extracted from container environment using the Docker introspection feature, acquires data from host operating system and system call traces. CONTAIN4n6 dashboard integrates the containers' data extraction feature from multiple sources. As a future work, the capability of this dashboard will be extended with the aim of adding an analysis engine. As Moby project is open source, it provides opportunity to add forensics features such as response time for monitoring and reporting, and to make it a compact forensicated version of Moby. We also see a need for a comprehensive method which would cover all phases of digital forensics to be applied in container environment.

Acknowledgements

We sincerely thank the reviewers and the Editor for their valuable suggestions.

Authors' contributions

Anand Kumar Mishra, Emmanuel S Pilli and Mahesh C Govil designed the study. Anand Kumar Mishra performed simulation and write the paper. All authors reviewed and edited the manuscript. All authors read and approved the final manuscript.

Funding

No funds have been received from any agency for this research.

Availability of data and materials

The data used to support the finding of this study are available from the corresponding author upon request.

Declarations

Competing interests

The authors declare that they have no competing interests.

Received: 29 July 2021 Accepted: 27 July 2022

Published online: 19 August 2022

References

1. Docker (2019) What is a Container? <https://www.docker.com/resources/what-container>. Accessed 20 Apr 2022
2. Google Cloud (2020) Google Kubernetes Engine. <https://cloud.google.com/kubernetes-engine>
3. AWS (2019) Amazon Elastic Container Service. <https://aws.amazon.com/ecs/>. Accessed 11 May 2022
4. Karmel A, Chandramouli R, Iorga M (2016) NIST definition of microservices, application containers and system virtual machines. No. NIST Special Publication (SP) 800-180 (Draft). National Institute of Standards and Technology. <https://csrc.nist.gov/publications/detail/sp/800-180/draft>. Accessed 10 Mar 2022
5. Docker (2019) Docker Platform. <https://www.docker.com/>. Accessed 20 Mar 2022
6. GO Programming (2019) GO Programming Language. <https://golang.org>. Accessed 17 Feb 2022
7. Ross RS (2013) Security and privacy controls for federal information systems and organizations. Technical report, National Institute of Standards and Technology
8. The National Institute of Standards and Technology (2020) Open Security Controls Assessment Language (OSCAL). <https://pages.nist.gov/OSCAL/>. Accessed 24 July 2022
9. Souppaya M, Morello J, Scarfone K (2017) Application container security guide. NIST Spec Publ 800-190:1–56
10. Mouat A (2015) Docker Security: Using Containers Safely in Production. O'Reilly Media, Sebastopol
11. Reshetova E, Karhunen J, Nyman T, Asokan N (2014) Security of os-level virtualization technologies. In: Nordic Conference on Secure IT Systems. Springer, Tromsø. pp 77–93
12. Bui T (2015) Analysis of docker security. arXiv preprint arXiv:1501.02967. <http://arxiv.org/abs/1501.02967>. Accessed 4 Jan 2022
13. Combe T, Martin A, Di Pietro R (2016) To docker or not to docker: A security perspective. *IEEE Cloud Comput* 3(5):54–62
14. NIST (2018) NATIONAL VULNERABILITY DATABASE. <https://nvd.nist.gov/>. Accessed 5 Mar 2022
15. Gummaraju J, Desikan T, Turner Y (2015) Over 30% of official images in docker hub contain high priority security vulnerabilities. Technical Report, Banyan Ops
16. Mostajeran E, Mydin MNM, Khalid MF, Ismail BI, Kandan R, Hoe OH (2017) Quantitative risk assessment of container based cloud platform. In: IEEE Conference on Application, Information and Network Security. IEEE, Sarawak. pp 19–24
17. Martin A, Raponi S, Combe T, Di Pietro R (2018) Docker ecosystem–vulnerability analysis. *Comput Commun* 122:30–43
18. Zerouali A, Mens T, Robles G, Gonzalez-Barahona JM (2019) On the relation between outdated docker containers, severity vulnerabilities, and bugs. In: IEEE 26th Int. Conf. on Software Analysis, Evolution & Reengineering. IEEE, Hangzhou. pp 491–501
19. Debian's security team (2020) Security Bug Tracker. <https://security-tracker.debian.org/tracker/>. Accessed 23 June 2022
20. Wenhao J, Zheng L (2020) Vulnerability analysis and security research of docker container. In: IEEE 3rd International Conference on Information Systems and Computer Aided Education. IEEE, Dalian. pp 354–357
21. Abed AS, Clancy TC, Levy DS (2015) Applying bag of system calls for anomalous behavior detection of applications in linux containers. In: IEEE Globecom Workshops. IEEE, San Diego. pp 1–5
22. Clausen J (2016) SANS ISC InfoSec Forums: Forensicating Docker. <https://isc.sans.edu/forums/diary/Forensicating+Docker+Part+1/20835/>. Accessed 8 Jan 2022
23. (2019) The Volatility Foundation. <https://www.volatilityfoundation.org/>. Accessed 19 Feb 2022
24. Winkel S (2017) Forensicating docker with elk. The SANS Institute. <https://sansorg.egnyte.com/dl/J3Zw8Npj4F>. Accessed 18 Apr 2022
25. Jian Z, Chen L (2017) A defense method against docker escape attack. In: International Conference on Cryptography, Security and Privacy. ACM, Wuhan. pp 142–146
26. Stelly C, Roussev V (2017) Scarf: A container-based approach to cloud-scale digital forensic processing. *Digit Investig* 22:39–47

27. Dewald A, Luft M, Suleder J (2018) Incident Analysis and Forensics in Docker Environments. ERNW WHITE PAPER. https://static.ernw.de/whitepaper/ERNW_Whitepaper64_IncidentForensicDocker_signed.pdf. Accessed 12 Apr 2022
28. Xiang J, Chen L (2018) A method of docker container forensics based on api. In: 2nd Int. Conf. on Cryptography, Security and Privacy. ACM, New York. pp 159–164
29. Lin X, Lei L, Wang Y, Jing J, Sun K, Zhou Q (2018) A measurement study on linux container security: Attacks and countermeasures. In: Proc. 34th Annual Computer Security Applications Conference. Association for Computing Machinery, San Juan. pp 418–429
30. Williams A, Ball B, Hoang Dinh G, Hecht L (2019) Monitoring and Management with Docker and Containers. <https://thenewstack.io/ebooks/docker-and-containers/monitoring-management-docker-containers/>. Accessed 29 Apr 2022
31. Sysdig (2020) Run Confidently with Secure Devops - Security for containers, Kubernetes, and cloud services. <https://sysdig.com/>. Accessed 18 Mar 2022
32. Lu Z, Xu J, Wu Y, Wang T, Huang T (2019) An empirical case study on the temporary file smell in dockerfiles. *IEEE Access* 7:63650–63659
33. Awuson-David K, Al-Hadhrami T, Funminiyi O, Lotfi A (2019) Using hyperledger fabric blockchain to maintain the integrity of digital evidence in a containerised cloud ecosystem. In: International Conference of Reliable Information and Communication Technology. Springer, Johor. pp 839–848
34. Chris Foster (2019) Root Please. <https://hub.docker.com/r/chrisfosterelli/rootplease/>. Accessed 16 Jan 2022
35. Docker-CE (2019) Fork and clone the Moby code. <https://github.com/docker/docker-ce/blob/master/components/engine/docs/contributing/set-up-git.md>. Accessed 28 Apr 2022
36. Docker (2019) Isolate containers with a user namespace. <https://docs.docker.com/engine/security/usersns-remap/>. Accessed 14 May 2022

Publisher's Note

Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Submit your manuscript to a SpringerOpen[®] journal and benefit from:

- Convenient online submission
- Rigorous peer review
- Open access: articles freely available online
- High visibility within the field
- Retaining the copyright to your article

Submit your next manuscript at ► [springeropen.com](https://www.springeropen.com)