Docker Container Security Analysis Based On Virtualization Technologies

Shatha A. Baker*, Hesham Hashim Mohammed‡, Omar I. Alsaif §
* ‡ § Northern Technical University, Iraq.

Abstract

The utilization of virtualization technology, particularly Docker containers, has increased significantly in recent years. As Docker provides a lightweight and efficient virtualization environment for software packages, ensuring its security becomes crucial. This paper performs a security analysis of Docker with two perspectives: the security within Docker and its relationship to the security features of the Linux kernel. Resources that are isolated, controlled, and limited are all examined in the Docker internal security. Linux Cgroup is used by Docker to manage computer resources, while Linux Namespace is used to securely isolate running environments. In this paper discussed how to separate resources such as filesystems, networks, devices, and processes, as well as how to isolate inter-process communication. Docker's interactions with Linux kernel security characteristics such as SELinux, AppArmor, Seccomp, and Linux functions were also discussed. These capabilities boost host system security by deploying Docker containers. AppArmor maintains security policies, SELinux offers further permission checks, and the Linux function limits container rights. Network-based assaults are defended against with the aid of Seccomp and the network framework. Additionally, the study makes recommendations for possible enhancements to improve Docker's security. This involves configuring Docker to deactivate specific functions within containers to thwart possible breaches and enhancing interoperability with Linux kernel security mechanisms.

Keywords: Docker Container, Virtual Machine, Security Analysis, Linux Kernel, SELinux, AppArmor, Seccomp.

1 Introduction

In today's world, virtualization has become essential. It provides a way to share resources among several users and is an affordable way to cut down on resource underuse, especially as cloud services become more and more common. Virtual machines (VM) and containers are two widely recognized technologies that facilitate virtualization [1].

Docker is a significant platform utilized by many developers. It is an open-source platform for packaging apps and running them in containers. The engine's objective is to deliver a quick, light environment for running the applications of developer and making deployment easy and efficient. However, ensuring the security of Docker containers is crucial to protect applications and sensitive data from potential threats [2].

Several approaches have been proposed to address Docker container security. Bui [3] offered a comprehensive examination of Docker technology, concentrating on both its aspects of internal securityand the modules of external security that may be integrated with Docker. Bélair et al in [4] introduced a novel taxonomy of container security that focuses on the infrastructure level, distinguishing it from previous works. They propose a classification of defense frameworks based on this taxonomy, specifically highlighting the methods by which the host OS can enhance container security. The authors in [5], extensively described the various attack within Docker technology and proposed effective strategies to mitigate these security risks. Another study [6] introduced a method of dynamic analysis that evaluates the Docker images security by analyzing their activities. The dynamic analysis method has been proven to be a useful addition to static analyses, which are frequently used in security evaluations of Docker images. In [7], The authors investigated denial of service attacks against the architecture of Docker and advocated the usage of a memory limit technique to mitigate against resource exhaustion induced by malicious containers.

In this paper, we are going to highlight the Docker architecture and its key components and analyze the Docker container security. The analysis looks at several topics: Docker's internal security depends on the isolation level that Docker provides for its virtual environments, the way Docker handles Linux kernel's security features like AppArmor and SELinux to

* shathaab@ntu.edu.iq
* ‡ § Computer Systems Department

strengthen the host system, and the challenges which face Docker container.

The paper is set up like follows: The second section provides a brief explanation to the Docker platform. The third section explains the Architecture of Docker. Docker security analysis is introduced in the fourth section. The fifth section focuses on discussing Docker security analysis and what can be done to improve it. The conclusion forms the final section.

2 Docker Overview

The "Docker" was initially established in March 2013 following the introduction of an innovative approach termed containerization, which prompted OS-level virtualization. Docker provides automated deployment capabilities for applications through containers. It adds an additional layer to the container environment, allowing virtualization and execution of applications. Docker is intended to build a lightweight and fast environment for efficient code execution. Additionally, it provides a convenient workflow for testing code before deploying it to production [8].

The container model is often misrepresented. While container technology appears secure due to its ability to contain all dependencies in one package, it does not guarantee overall security. Container platforms, like other cloud platforms, are also vulnerable to various threats from both internal and external sources. Operating secure services within a multi-tenant cloud system poses several challenges in virtual environments. It is widely acknowledged that VMs created through hypervisor-based virtualization methods offer higher levels of security compared to containers, table 1 shows the differences between the two technologies [9].

Table 1: The distinction between Docker containers and virtual machines [9]

Docker container	Virtual Machine
Within a few seconds, boots	It needs a few minutes to boot.
The Docker engine is utilized by Docker.	VMs are operated by Hypervisor.
The utilization mechanism for Docker is complex.	The use of and interaction with VM tools is simple.

The VMs increase the level of isolation between the host and the apps. VM-based applications are restricted to communicating solely with the VM kernel and are unable to access the host kernel directly. Therefore, for an attacker to target the host kernel, they would need to bypass both the hypervisor and the kernel of VM. Conversely, the container model enables direct application access to and communication with the kernel of host, as illustrated in figure 1.

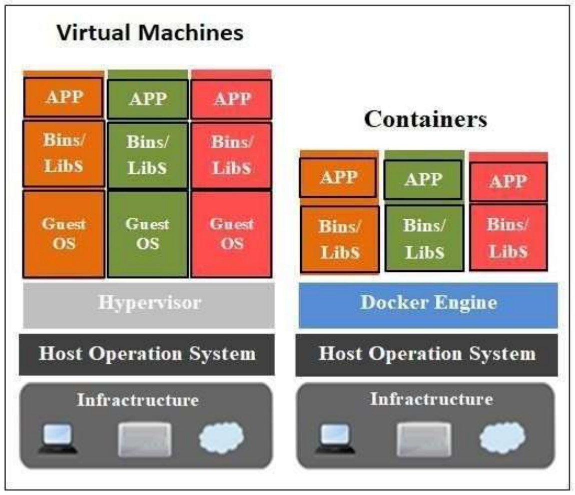

Figure 1: Docker and VM [10]

This situation enables an attacker to directly attack the host kernel, making container technology more susceptible to security concerns when compared to VM platforms [10].

3 Architecture of Docker

Docker architecture consists primarily of major components, as shown in Figure 2: Docker host and client, Docker image, Docker registries, and Docker container[11]. In the sections that follow, these elements will be covered in more detail:

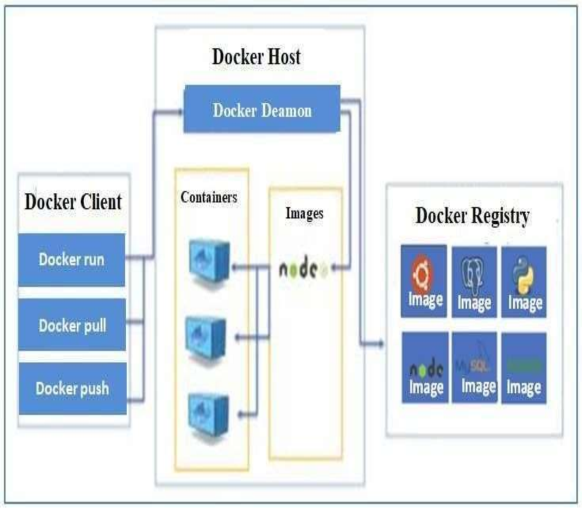

Figure 2: Docker Architecture [11]

3.1 Docker Host and Client

Docker client and Docker host or daemon can operate on the same machine, or a local Docker client can connect to a host or daemon operating on a different machine.

Docker clients allow users to interact with Docker. When a Docker command is executed, the client sends it to the Docker daemon, that executes it and is in responsible of all container-related activities and collects commands. The Docker client can communicate with multiple daemons. A complete environment for executing and running applications is provided by the Docker host. This includes the Docker daemon, image, container, storage, and network [12].

3.2 Docker Images

It is significant to build docker images based on two methods. The first one includes building an picture from a read-only pattern. A base image serves as the foundation for all images. These operating system images allow the creation of a container with a fully functional operating system. Alternatively, a center image can be produced from scrape and modified by adding the required applications. The second step involves creating a Dockerfile containing a list of instructions. When the "Docker build" command is executed, it uses these instructions to construct the image. With the use of this technology, creating images can be automated [13].

3.3 Docker Registries

Registries, which function similarly to source code repositories, are where Docker images are kept. These registries serve as centralized locations for uploading and downloading images. Private and public registrations are the two different categories of registers. Anyone may push and pull their images from public registries like Docker Hub without having to start from scratch. It is commonly known that Docker Hub is a public registry that makes it simple to access and share Docker images. Docker has the possibility to construct private registries in addition to public ones. These private registries use Docker's hub feature to limit access and let businesses or individuals share images to particular public or private spaces. Finally, Docker registries are essential for organizing and distributing Docker images because they provide both private and public choices to meet varying security needs and use cases. [14].

3.4 Docker Containers

A Docker image is used to create a Docker container, which contains all the parts needed to run an application in an isolated manner. Docker containers have a number of advantages. By doing so, developers may ensure consistent behavior across development, testing, and production environments and package apps and their dependencies into a single unit. Additionally, containers offer isolation, which eliminates issues while running many applications on the same host. They provide easy orchestration, scalability, and rapid deployment, which simplify the management and scaling of applications in a distributed, cloud-native environment. [15].

4 Docker Container Security

When it comes to running services in virtual environments like Docker, which is built on container-based virtualization technology security is a major concern. A number of factors are examined in order to evaluate the security that Docker offers for applications that are executing. This analysis explores the internal security measures implemented by Docker containers, along with the enhanced security features provided by the underlying kernel. It also addresses the challenges that arise when working with Docker containers from a security standpoint.

4.1 Docker Internal Security

Docker containers differ from traditional virtualization methods as they do not rely on virtualizing hardware or employing a separate operating system. Instead, Docker utilizes the Linux Namespace mechanism to achieve safe isolation of operating environment. Furthermore, it leverages the Linux Cgroup mechanism effectively to manage computer resources. The employing process of these mechanisms, Docker ensures the secure separation of virtual objects without the need for hardware virtualization or an independent operating system [16 - 17].

4.1.1 Isolation of a resource

Docker ensures secure isolation by leveraging the Linux Namespace mechanism. By using this mechanism, isolated containers are certain not to be able to access resources associated with other containers, thereby providing transparency in computer resource allocation. The details of resource isolation achieved through namespaces are presented in table 2 [18].

Table 2: Details of resource isolation [18]

Contents Isolated	Call Arguments of System	Namespace
Isolation of Process	CLONE _ NEWPID	PID
Isolation of Inter-Process Communication	CLONE _ NEWIPC	IPC
Isolation of a device	CLONE _ NEWUTS	UTS
Isolation of a network	CLONE _ NEWNET	Network
Isolation of a Filesystem	CLONE _ NEWNS	Mount

Isolation of Process PID: Process isolation in Docker is achieved through the utilization of PID (Process Identifier) namespaces. The main objective of this process, prevent compromised containers from interfering with other containers through management process interfaces. Docker implement this procedure by encapsulating processes within containers and restricting their permissions and access to other containers that deals with host device [18].

Isolation of Inter-Process Communication (IPC): shared memory, semaphores and message queues blocks are good examples of IPC objects used for exchanging data among processes. When processes operate in containers, it is important to restrict their communication to a specific set of IPC resources and prevent interference with processes running on the host machine or other containers. Docker employs IPC namespaces to achieve IPC isolation. By setting the parameter CLONE_NEWIPC during the clone operation, Docker creates separate IPC namespaces. Each IPC namespace consists of a collection of IPC object identifiers. Processes within a namespace cannot access or modify IPC resources in another namespace[18].

Isolation of a device: In Unix, Device drivers can be accessed by device nodes, which are special files, allowing the kernel and programs to access hardware. However, if a container has unrestricted access to critical device nodes, it can potentially cause significant damage to the host system. As a result, it is critical to limit a container's access to device nodes [3].

Isolation of a network: Network isolation is crucial to prevent network-based attacks like ManintheMiddle and ARP spoofing. Docker creates separate network stacks for each container, allowing them to interact through their respective network. By default, connectivity is provided via the Virtual Ethernet Bridge, which forwards packets between its network interfaces. However, this model is vulnerable to ARP spoofing and Mac flood attacks as it forwards all incoming packets without filtering [19].

Isolation of a Filesystem: Docker uses mount namespaces to isolate the filesystem hierarchy associated with different containers, providing processes with different views of the file system structure. However, some kernel file systems do not have a namespace, causing containers to inherit the host's view and access them directly. Docker limits threats through two filesystem protection mechanisms: revokes container permissions to write to these filesystems and prohibits processes from remounting a file system inside the container. Additionally, Docker uses a copy on write file system, allowing each container to write content to its specific filesystem[20].

4.1.2. Control of Resources

Docker containers use Linux's Control Group, or Cgroup to ensure efficient utilization of system resources like memory, CPU, block I/O and bandwidth. As a result, each container instance is able to compete for resources equally, and it is nearly impossible for any container instance to run out of the host computer's system resources. When a system resource is depleted, the Linux kernel will trigger out of Memory, that will terminate all active containers or processes. As a result of the Cgroup mechanism, Denial-of-Service (DoS) attacks are able to successfully controlled [21].

4.1.3. Limiting of Resources

A common attack on multi-tenant systems is a DoS attack, where a process consumes all system's resources, disrupting the normal operation of remaining processes. Limiting the resources allotted to each container ought to be achievable in order to stop this kind of attack. The primary tool used by Docker to address this problem is Cgroups. Resource management in Docker involves overseeing the allocation of key resources like CPU disk I/O and memory for each container. This ensures equitable distribution of resources among containers and prevents any single container from monopolizing them[3].

4.2. Docker and Kernel Security System

Two kernel security mechanisms, the Linux function and the Linux Security Module (LSM), that can improve the Docker security and the underlying kernel. The Linux function limits each process's permissions, while the Linux kernel manage many security models using the infrastructure provided by the LSM. LSM integrates security features like AppArmor, SELinux and other implemented measures into the official Linux kernel. This kernel security system contains the following items:

4.2.1. Linux function

Users are classified into two categories on traditional UNIX systems: root users and non root users. The first type possess high privileges and can bypass the kernel's permission checks, while the second type must adhere strictly to execution authorizations to perform tasks. Linux introduced the capability mechanism in version 2.2, which enables non-root users to execute tasks previously restricted to root users. This mechanism operates on processes or files, facilitating access control.

Furthermore, in the event that an intrusion manages to gain root access within the containers, Docker employs measures to mitigate its impact on the host system. This is achieved by restricting a specific set of Linux functions. Table 3 displays some of the disabled functions within a Docker container [19] .

4.2.2. SELinux

A security improvement for the Linux system is SELinux. Linux includes the common Discretionary Access Controls (DAC) method to manage access to objects, including owner/group and permission flags. Following the normal DAC, SELinux offers a second level of permission checking known as Mandatory Access Control (MAC). MAC can offer greater security than DAC. As users are unable to alter their security level or the object security features, MAC relies on matching of user and data security levels to make specific and objective decisions. The subjective factors of user are secured, resulting in an overall improvement in system security. MAC is typically used in conjunction with DAC in the Linux system, and specialized security modules are developed within the LSM framework. Everything is managed via labels in SELinux. Each process, system object, and file/directory has a label. These labels are used by the system administrator to create rules that regulate access among system objects and processes. They are referred to as policies [22].

Table 3: list of disabled Docker container functions[19]

process name of Docker	Disabled functions
CAP _ SETPCAP	Modifying process functions
CAP _ SYS _ MODULE Insert	delete kernel module
CAP _ SYS _ RAWIO	Change the memory of kernel
CAP _ SYS _ PACCT	Configuration process record
CAP _ SYS _ RESOURCE	Covering resource constraints
CAP _ SYS _ NICE	Change the process priority
CAP _ SYS _ TIME	Change the clock of system
CAP _ AUDIT _ WRITE	Writing audit logs
CAP _ MAC _ OV ERRIDE	Disregard the MAC policy in the kernel.
CAP _ SYSLOG	Changing the behavior of kernel printk
CAP _ SYS _ TTY _ CONFIG	Configuring TTY devices
CAP _AUDIT _ CONTROL	Configuring audit subsystem
CAP _ MAC _ ADMIN	Set up the MAC configuration
CAP _ SYS _ ADMIN	Select all
CAP _ NET _ ADMIN	Configure the network

4.2.3 AppArmor

AppArmor is a model for improving Linux security that relies on required access control, but it limits its scope to individual programs. This restricts the functionality of the program by downloading files of security configuration by administrators in each file. When a new container is started by Docker on a system that supports AppArmor, an interface is provided for loading a AppArmor profile. The profile is installed in application mode, ensuring that processes within the container adhere to the profile's restrictions. If no profile is specified during container startup, a default profile is automatically loaded into the container by the Docker daemon. This default profile prevents access to the host's crucial filesystems [23].

4.2.4. Seccomp

Seccomp is a mechanism that enables the restriction of system calls made by user processes and allows the filtering of system call parameters. System calls play a crucial role in connecting kernel states and user. Processes can function within a secure and controlled range through restricting system calls.

In Docker, a whitelist approach is adopted for Seccomp, where a configuration file is utilized to specify the allowed system calls, while over 50 specific system calls are blacklisted. This approach ensures that processes running within Docker containers are restricted to a safe subset of system calls and mitigates potential security risks[24].

5 Enhancing Docker Containers Security

Docker Containers have gained popularity in the software development community because they allow developers to bypass time-consuming library and dependency setting. However, along with the benefits of containerization, there are also security challenges that may make data vulnerable to attackers so must be addressed to ensure the integrity and protection of containerized applications[26]. Let's explore some of the key security challenges facing Docker containers and the solutions to address them shown in table 4.

Docker Containers supports the usage of SELinux and AppArmor frameworks to improve the Docker containers security. These frameworks enables to specify rules of the activities and access rights for containers, such as Docker-sec and Lic-sec.

Docker-sec offers an extra layer of security on top of Docker's default security by automatically constructing per-container AppArmor profiles.Docker-sec creates profiles of container based on behavior of application and configuration instructions, see figure 3. This procedure combines dynamic monitoring and static analysis to generate and enhance profiles of container operations during a predetermined test period.

Table 4: Security challenges for container networks

Security Challenge	**Description**	**Solution**
Container Breakouts	Compromise within a container can lead to the entire host being compromised.	Implement strong isolation between containers, apply container runtime security measures, and regularly update container runtimes.
Image Vulnerabilities	Vulnerabilities within container images can be exploited to compromise containerized applications.	Use trusted base images, regularly update container images, perform vulnerability scanning, and enforce secure image sourcing practices.
Inadequate Isolation	Weak isolation between containers can result in unauthorized access and data leakage.	Utilize container network segmentation, apply proper access controls, and enforce least privilege principles for containerized applications
Network Security	Ensuring secure communication, traffic control, and protection against network-based attacks.	Implement network security policies, utilize secure overlay networks or service meshes, and employ network monitoring and intrusion detection systems.
Dynamic IP Changes	Dynamic IP addresses make it challenging to implement IP-based access control and modify firewall rules.	Explore container network security solutions that offer dynamic IP-based access controls and leverage container orchestrators' network policies.
Orchestration Platform Security	Misconfigurations or vulnerabilities in the container orchestration platform can lead to unauthorized access and data exposure.	Follow security best practices for container orchestration platforms, regularly update them, enforce strong access controls, and implement security auditing.
Insider Threats	Malicious insiders with legitimate access can misuse privileges and compromise containers or data.	Implement strong access controls, enforce the principle of least privilege, monitor and audit user activities, and provide security awareness training.

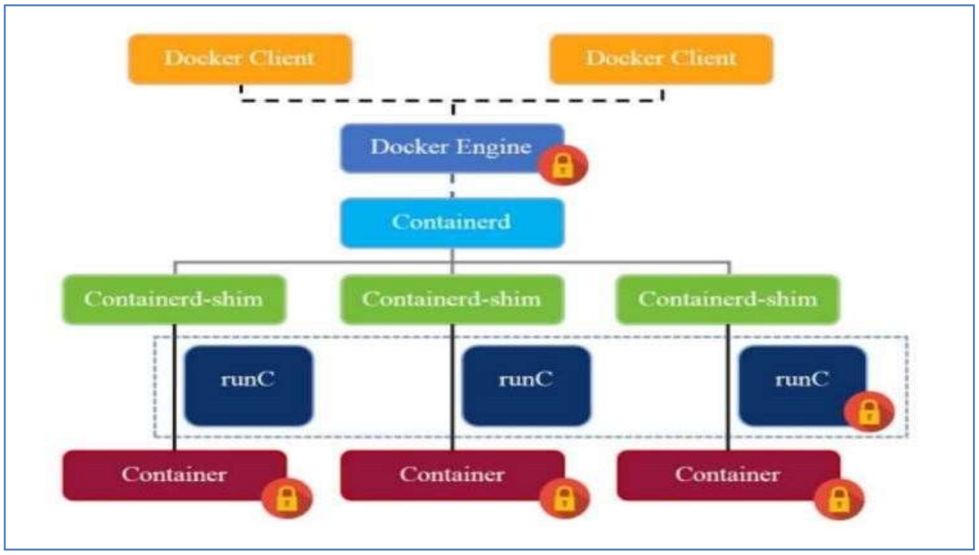

Figure 3. Components of the counter that the Docker-sec can protect

Dynamic Monitoring allows users to schedule container training time, allowing Docker-sec to capture behavior data, identify necessary permissions, examine audit log, adjust execution profile, potentially decreasing capabilities, and repeat until needed functionality is documented. Static Analysis gathers container and operation data from command-line arguments supplied by the user or data produced by Docker. The data is employed to create rules basic security and presentation new containers profiles. Docker-sec collects critical details from arguments of command line and utilities of Docker, like container volumes, privileges of user, and the SHA256 checksum of the container. Docker-sec may briefly enforce an AppArmor profile during the container setup process before switching to the one used during the container's runtime. Despite the capabilities provided by Docker-sec, its protection against user space program targeted exploits is limited[27].

Lic-Sec is an AppArmor profile generator which merges Docker-sec and LiCShield's best features in order to offer a more complete and effective security solution for Docker containers, figure 4 provides an overview of the Lic-Sec. It concentrates on creating AppArmor profiles for all Docker components automatically, assuring greater confinement of rights inside the container, and facilitating expansion of functionality, although it still has limits when it comes to web-server attacks. The LiCShield framework secures Docker containers and workloads by automatically generating AppArmor rules for both the host and container. It traces kernel operations using SystemTap5, converts traces to AppArmor rules, and constructs two profiles: one for operations inside the container and one for operations on the host[28].

6 Discussion of the Docker Container Security

Docker Containers provide an effective method and lightweight of packaging an application and all of its dependencies. However, several security concerns are preventing their wider deployment [29]. According to the analysis, Docker allows a high level isolation of process, inter-process communication, device, network, and filesystem. It also has the ability to control resources and limiting resources for its containers.

LiCShield and Docker-sec are popular ways to improve Docker container security based on MAC and allow container protection without manual configurations. The design principle of LiCShield is comparable to Docker-sec. It does not, however, build rules for capabilities and network accesses, but it does generate other critical rules that Docker-sec cannot generate, such as file access rules, mount rules, and so on. Table 5 shows the differences between the two methods [28]. Lic-Sec an AppArmor profile generator combines the strengths of both methods and provides stronger protection. Lic-Sec provides a mechanism for creating AppArmor rules that are specific to each container. It will be possible to create a generator that provides a greater level of safety with more rules to protect various portions.

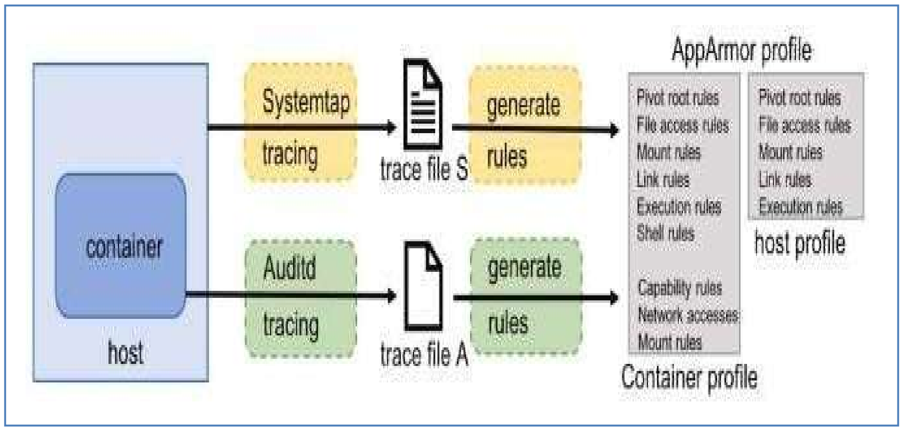

Figure 4. Overview of the Lic-Sec framwork

Table 5 **:** Comparison between and LiCShield and Docker-sec.

LSM	**MAC**	**Year**	**Tracing tool**	**Generated profiles**	**Generated rules**	**Pre-defined profiles**	**Effective protective range**
LiCShield	AppArmor	2015	SystemTap	Container runtime Docker daemon	Pivot root rule access rule mount rule link rule execution rule	-----	Docker daemon
Docker-sec	AppArmor	2018	Auditd	Container runtime	Capability rule network rule	RunC Docker daemon	Container RunC Docker daemon

7 Conclusions and Future Work

The migration of critical applications to containers is anticipated to accelerate as container applications continue to gain popularity. While assuring convenient deployment and optimum resource usage, improving container security will become an increasingly important problem.

This paper illustrates the fundamental concepts of Docker and explains how Docker achieves secure isolation through the use of Linux Namespace and Cgroup mechanisms, which ensure resource isolation and control within containers, as well as the importance of considering Linux kernel security features such as AppArmor and SELinux to strengthen the host system when using Docker. A variety of additional measures have been proposed to improve the Docker containers security, and some of the ways to enhance container security are reviewed.

In the future work, we want to improve Docker's scheduling feature and create a safer container variant that will loosen security restrictions.

References

[1] MP, A. R., Kumar, A., Pai, S. J., Gopal, A. “Enhancing security of docker using linux hardening techniques”. 2nd International Conference on Applied and Theoretical Computing and Communication Technology, pp.94–99, 2016.

[2] Lee, Haneul, Soonhong Kwon, Jong-Hyouk Lee. “Experimental Analysis of Security Attacks for Docker Container Communications”. Electronics 12.4 , 2023.

[3] Bui, Thanh. “Analysis of docker security”. arXiv preprint arXiv:1501.02967 , 2015.

[4] Bélair, M., Laniepce, S., & Menaud, J.-M. Leveraging kernel security mechanisms to improve container security: A survey. Proceedings of the 14th international conference on availability, reliability and security, 1–6, 2019.

[5] Souppaya, Murugiah, John Morello, Karen Scarfone. “Application container security guide” (2nd draft). No. NIST Special Publication 800-190. National Institute of Standards and Technology, 2017.

[6] Brady, K.; Moon, S.; Nguyen, T.; Coffman, J. "Docker Container Security in Cloud Computing". In Proceedings of the 2020 10th Annual Computing and Communication Workshop and Conference (CCWC), Las Vegas, NV, USA, 6–8 January; pp. 975–980, 2020.

[7] Chelladhurai, Jeeva, Pethuru Raj Chelliah, Sathish Alampalayam Kumar. "Securing docker containers from denial of service (dos) attacks", International Conference on Services Computing (SCC). IEEE, 2016.

[8] T. Combe, A. Martin, R. Di Pietro, "To Docker or Not to Docker: A Security Perspective" in IEEE Cloud Computing, vol. 3, no. 5, pp. 54-62, Sept.-Oct. 2016.

[9] Jain, V., Singh, B., Khenwar, M., Sharma, M. "Static vulnerability analysis of docker images", IOP Conference Series: Materials Science and Engineering. Vol. 1131. No. 1. IOP Publishing, 2021.

[10] Yasrab, Robail. "Mitigating docker security issues". arXiv preprint arXiv:1804.05039 , 2018.

[11] Garg, Somya, Satvik Garg. "Automated cloud infrastructure, continuous integration and continuous delivery using docker with robust container security." IEEE Conference on Multimedia Information Processing and Retrieval (MIPR). IEEE, 2019.

[12] Loukidis-Andreou, F., Giannakopoulos, I., Doka, K., Koziris, N. "Docker-sec: A fully automated container security enhancement mechanism." IEEE 38th International Conference on Distributed Computing Systems (ICDCS). IEEE, 2018.

[13] Rad, Babak Bashari, Harrison John Bhatti, Mohammad Ahmadi. "An introduction to docker and analysis of its performance." International Journal of Computer Science and Network Security (IJCSNS) 17.3 ,228, 2017.

[14] Brogi, Antonio, Davide Neri, Jacopo Soldani. "DockerFinder: Multi-attribute search of docker images." 2017 IEEE international conference on cloud engineering (ic2e). IEEE, 2017.

[15] S. Winkel, "Security Assurance of Docker Containers: Part 1," ISSA Journal, April 2017.

[16] Alyas, Tahir, et al. "Container Performance and Vulnerability Management for Container Security Using Docker Engine." Security and Communication Networks 2022.

[17] E. Reshetova, J. Karhunen, T. Nyman, N. Asokan. "Security of OS-level virtualization technologies". In Proceedings of the 2014 NordSec Conference, pages 77–93, Norway, 2014.

[18] D. Huang, H. Cui, S. Wen, C. Huang, "Security Analysis and Threats Detection Techniques on Docker Container", IEEE 5th International Conference on Computer and Communications (ICCC), Chengdu, China, 2019, pp. 1214-1220

[19] J. Wenhao, L. Zheng, "Vulnerability Analysis and Security Research of Docker Container", IEEE 3rd International Conference on Information Systems and Computer Aided Education (ICISCAE), Dalian, China, 2020, pp. 354-357,

[20] Sever D , Kisasondi T . "Efficiency and security of Docker based honeypot systems", International Convention on Information &Communication Technology, Electronics & Microelectronics. 2018.

[21] S. Wu, K. Wang, H. Jin, "Research Situation and Prospects of Operating System Virtualization", Journal of Computer Research and Development, vol. 56, no. 1, pp. 58-68, 2019.

[22] Bhalerao, Amol. "Docker and analysis of its security." Journal homepage: www. ijrpr. com ISSN 2582: 7421.

[23] Zhu, Hui, Christian Gehrmann. "Lic-Sec: an enhanced AppArmor Docker security profile generator." Journal of Information Security and Applications 61: 102924, 2021.

[24] Tolaram, Nanik. "Docker Security." Software Development with Go: Cloud-Native Programming using Golang with Linux and Docker". Berkeley, CA: Apress, 89-107, 2022.

[25] Xiang, Jie, Long Chen. "A method of docker container forensics based on api". Proceedings of the 2nd International Conference on Cryptography, Security and Privacy. 2018.

[26] Baker, S. A., Nori, A. S. "Internet of things security: a survey". Advances in Cyber Security: Second International Conference, ACeS 2020, Penang, Malaysia, December 8-9, 2020, Revised Selected Papers 2. Springer Singapore, 2021.

[27]Loukidis-Andreou, F., Giannakopoulos, I., Doka, K., & Koziris, N. "Dockersec: A fully automated container security enhancement mechanism". 2018 IEEE 38thInternational Conference on Distributed Computing Systems (ICDCS), 1561–1564, 2018.

[28] Zhu, Hui, and Christian Gehrmann. "Lic-Sec: an enhanced AppArmor Docker security profile generator." Journal of Information Security and Applications 61 (2021): 102924.

[29] Baker, S. A., Nori, A. S. "A secure proof of work to enhance scalability and transaction speed in blockchain technology for IoT". In AIP Conference Proceedings (Vol. 2830, No. 1). AIP Publishing , 2023.

BIOGRAPHY / BIOGRAPHIES

Dr. Shatha A. Bakr has a Bachelor's degree in Computer Science from the University of Mosul, which she obtained in 1997. In 2013, she earned a master's degree in Computer Science from the same university. Later, in 2022, she completed her Ph.D. from the University of Mosul. Dr. Bakr worked as a Lecturer at the Northern Technical University in Mosul, Iraq. Her research interests encompass mobile phone programming, information security, multimedia communications, and artificial intelligence.

Dr. Hisham Hashem earned his bachelor's, master's, and doctoral degrees from the University of Mosul, Faculty of Computer Science and Mathematics, Department of Computer Science, in 2010, 2013, and 2022, respectively. Currently, he works as a Lecturer at the Northern Technical University in Mosul, Iraq. His research areas encompass big data, artificial intelligence applications, image processing, and bioinformatics

Omar I. Alsaif is currently a lecturer in the Mosul Technical Institute/Northern Technical University in Mosul, Iraq. He received his B.Sc. in electrical engineering from the University of Mosul in 1992. In 2005 and 2018, he obtained his M.Sc. and Ph.D. degrees in Electronics and Microelectronic Engineering from Mosul University, respectively. His research interests encompass microelectronic and solid-state systems, renewable energy, and nanotechnology devices.