Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 233 (2024) 660–669

Procedia Computer Science

www.elsevier.com/locate/procedia

5th International Conference on Innovative Data Communication Technologies and Application (ICIDCA 2024)

An Empirical Investigation of Docker Sockets for Privilege Escalation and Defensive Strategies

Rajyashree R[a], Senthilkumar Mathi[b, *], Saravanan G[c], Sakthivel M[d]

[a, b]*Department of Computer Science and Engineering, Amrita School of Computing, Coimbatore, Amrita Vishwa Vidyapeetham, India.*
[c]*Department of Artificial Intelligence and Data Science, Erode Sengunthar Engineering College, Erode, India.*
[d]*Department of Computer Science and Engineering,* Erode Sengunthar Engineering College, Erode, India.
[b*]m_senthil@cb.amrita.edu

Abstract

Cloud-based infrastructures often leverage virtualization, but its implementation can be expensive. Traditional coding methods can lead to issues when transitioning code from one computing environment to another. In response, the container paradigm emerged to offer cost-effective and agile delivery. Containers differ from full machine virtualization by compactly encapsulating the entire software and its dependencies. Leveraging containers, developers can create more secure and efficient applications. Docker, a prominent containerization platform, facilitates the execution of docker images. The Docker Hub serves as a popular repository for various images. Given the importance of application security, especially in the face of threats like malware, ransomware, and data breaches, ensuring robust security is imperative. The paper investigates vulnerabilities within Docker containers and proposes defensive strategies to mitigate potential breaches. In addition, it investigates attacks involving Docker sockets and suggests preventive measures for non-root users.

Keywords: Attack; Containerization; Data heist; Docker, Malware; Virtualization; Vulnerabilities.

1 Prologue

The demand for effective and secure virtualization is inclined towards a safeguarded, portable, and adaptable environment. These solutions can be broadly categorized into two main approaches: hypervisor-based virtualization

10.1016/j.procs.2024.03.255

and container-based virtualization, often called OS-level virtualization. Among these, containers stand out for their resource efficiency and versatility, making them a highly sought-after solution. They achieve efficiency by minimizing the overhead typically associated with traditional virtual machines. Docker, an open-source virtualization platform, caters to software developers and system architects across Windows, Linux, and macOS environments; with Docker, various applications' installation, testing, and hosting become feasible. This platform facilitates rapid software deployment while optimizing resource utilization and reshaping multi-tier systems' dynamics in cloud infrastructures.

Despite its achievements in container services, vulnerabilities persist within these systems, making them susceptible to attacks such as denial of service, distributed denial of service, ARP spoofing, and image poisoning [1]. These potential threats target various components, including hosts, the Docker engine, and applications.

Docker's architectural design is based on a client-server model comprising three pivotal components: the Docker client, daemon, and registry. The Docker client plays a crucial role in creating, running, and managing Docker containers interacting with the Docker daemon. This interaction takes place via a UNIX socket or network interface. Through this socket, the Docker daemon connects to the Docker engine and initiates the establishment of a connection. This mechanism depends on the nature of the socket, predominantly Unix, although alternatives like TCP and file descriptors can be employed.

Utilizing Unix sockets in Docker allows users to access Docker as either a privileged root user or a non-root user. Managing multiple containers is facilitated by a stream of Docker clients termed "docker-compose."

Docker images are central to effective Docker management, managed by Docker registries to generate Docker containers. These images encapsulate essential components like code and dependencies to establish a Docker environment. Images can be created interactively, involving manual modifications to an existing image, resulting in a new version. Alternatively, they can be generated through Docker files to create a new image. These files are organized as layers, stacked where changes in the uppermost layer necessitate fewer computations to rebuild an image.

The executable instance of an image is a container, which can be created, launched, stopped, or removed using the Docker API or command line interface [2]. These containers can access one or more networks or produce a distinct image based on the ongoing state. Following creation, a Docker container can be activated using the "docker run" command, establishing a writable container layer above the specified image. In conjunction with "docker commit," this command can modify commands within the container.

The Docker socket performance can be improved by using monitoring tools like Prometheus and cAdvisor to monitor the performance metric, including socket latency and throughput. Socket caching library tools like docker-sock proxy cache the socket request, thereby improving the performance. Another efficient approach to elevate the socket performance is to leverage the Docker Container to Kubernetes for effective socket access and resource allocation.

The paper's structure is outlined as follows: Section 2 delves into related works, followed by section 3, which offers insights into attack and defence orchestration. Section 4 encompasses result analysis and image examination, while section 5 concludes the paper.

2 Literature Survey

The current section discusses the vulnerabilities in security and potential attack scenarios that can be executed within the Docker environment.

Aparna et al. devised a comprehensive threat model, as documented in their work [1], which encompasses a spectrum of potential attacks targeting the host system and its various layers. This study delves into the specifics of the denial of service (DoS) attack. Vulnerabilities exist within the container layer, rendering it susceptible to diverse attacks such as malware infiltration, DoS incidents, privilege escalation, ARP, MAC spoofing, and container escape attempts. Furthermore, the investigation examines intrusions at different levels, encompassing the application layer and the Docker engine. These incursions include DoS occurrences, malware infiltration, manipulation of images to carry malicious payloads or the utilization of outdated software, the injection of evil code, exploits targeted at the kernel, and instances of crypto-jacking. An intriguing observation from the research is the immediate surge in CPU usage observed after the execution of a DoS attack. This phenomenon sheds light on the impact of such an attack on system resources.

Vipin et al. explore Docker images, focusing on static and dynamic security analyses to detect potential bugs. This emphasis is driven by containers' reliance on these images [2]. The authors outline various factors that can render Docker images vulnerable to tampering, including insecure production practices, cryptographic misconfigurations, issues related to decompression, and inherent vulnerabilities associated with Docker and lib-container. To address these concerns, the research highlights using tools such as Clair, Anchore, Dagda, Notary, and Grafes. These tools assess and identify vulnerabilities in the Docker images, ensuring a comprehensive evaluation of their attack susceptibility.

Ahmet et al. provide valuable insights into vulnerabilities associated with web applications [3]. They highlight the significance of the "docker content trust" security feature, introduced in version 1.8. Once enabled, this feature safeguards against downloading unsigned Docker images. It plays a critical role in ensuring the verification, authentication, and preservation of the integrity of Docker images. Furthermore, the researchers delve into enhancing security measures by controlling container resource consumption. It can be achieved by utilizing the "docker run" command, which limits the resources allocated to containers. This practice improves security considerations by preventing excessive resource consumption and potential vulnerabilities.

The research outlined in [4] conducted comprehensive penetration testing within the Docker environment. This testing encompassed various attack scenarios, including DoS, Docker container escape, and side-channel attacks, which were meticulously analyzed. Viewing the container as an initial point of entry, an attacker, upon gaining access, undertakes scrutinizing the container's privileges and potential vulnerabilities. Once identified, these weak points become the focus of the attacker's efforts, often leading to targeted attacks such as ARP poisoning, MAC flooding, and sniffing. In the context of the Docker daemon, vulnerabilities emerge during the parsing of files. A crucial consideration is whether these vulnerabilities can extend their impact beyond the container to compromise the host system. It is especially relevant due to the potential of misconfigurations and the likelihood of a series of attacks originating from remote access authentication. Also, the research highlights how the broader environment influences security risks. Importing images from external sources introduces the possibility of man-in-the-middle attacks or image hijacking, as these externally sourced images might be tampered with or compromised. The research underscores the multifaceted nature of security threats within Docker environments, ranging from the container's internal vulnerabilities to the implications for the host system and the broader ecosystem.

Jyoti et al. have conducted a comprehensive assessment of the security landscape surrounding Docker containers, a review currently under evaluation [5]. Given the surge in cloud computing and Docker adoption due to their advantageous portability and adaptability in application development, a significant concern arises regarding the security of images sourced from diverse repositories. To address this, the authors propose using continuous integration and continuous deployment (CI/CD) practices as part of the software development lifecycle [6]. This approach ensures a meticulous evaluation of Docker images' integrity. The investigation leverages tools such as VirusTotal to detect potential malicious elements within images. In parallel, they execute a Docker instance with tcpdump, facilitating the detection of suspicious network activities. To estimate the efficacy of their proposed technique, they conducted experiments involving deliberately vulnerable images. The subsequent evaluation measures the precision with which their methodology identifies these vulnerabilities, stating its robustness.

In the investigation detailed in [7], an examination encompassing 2,227,244 images was conducted, focusing on their metadata sourced from the Docker Hub. This investigative effort yielded valuable insights. The researchers additionally propose the implementation of dynamic scans for individual packages installed within active containers, an approach that significantly enhances security. Moreover, they advocate for utilizing "apt-get-upgrade" to elevate packages to secure versions within operational containers, contributing to an elevated level of safety.

The emphasis lies on enhancing docker image security in the context of scientific data analysis, as evidenced in a study [8]. This research involves a comparative assessment of four vulnerability scanners, delineating their efficacy. The impact of image upgrading and mitigation measures on threat reduction is also quantified. Notably, vulnerabilities are addressable through image refinement and removing extraneous packages. The combined application of these strategies contributes to reducing vulnerabilities arising from unutilized software and patches managed by package maintainers. The study also sheds light on the broader docker infrastructure, exploring its security trends [9]. Among the security recommendations presented by this research is the endorsement of streamlined operating system projects like CoreOS and Ubuntu Core, which foster a more secure container environment. Introducing Stackelberg's model, facilitated by linear programming, aids in selecting optimal security practices and operational efficiency.

While Docker offers lightweight virtualization, its security is compromised as container isolation is weaker than traditional VMs. Vulnerabilities like inter-container traffic, insecure image environments, and runtime susceptibilities jeopardize system security, integrity, and access controls. Addressing these concerns, researchers in [10] provide insights into hardware and software security approaches. Hardware security involves trusted platform modules that expedite algorithms, ensure data encapsulation, and enable secure boot. Software solutions encompass namespaces, segregating hostnames, end-users, file systems, and other resources. This process validates separation before establishing connections between containers. The abovementioned issues also give rise to virtualization security challenges, including insider attacks within Docker. Such attacks originate internally from malicious users who gain access through commands, potentially compromising host folders and data. This vulnerability analogy aligns Docker security with ship security, illustrating that while a dock might be secure, a vulnerable ship can undermine its safety. Therefore, guarding against external attacks is crucial. Measures to counter external threats include reducing container privileges, implementing access control policies, adhering to secure deployment guidelines, managing Daemon privileges, enabling logging/auditing, employing SELinux/AppArmor, and utilizing cgroups. These defensive strategies are elaborated upon in [11].

The insights presented in [12] shed light on the diverse dimensions through which Docker can be exploited. The primary emphasis lies in cultivating user awareness regarding the utilization of Docker packages. This imperative arises due to the consequential ramifications of employing malicious packages, which can result in losing valuable assets such as confidential data, financial resources, and an organization's reputation. The study delves into the context of Elastic Containers, employing it as a testing ground to showcase potential attacks within the Docker environment. The researchers illustrate conceivable attacks in the Docker ecosystem within this experimental framework. These include injecting malicious data facilitated by cURL commands, unauthorized password cracking, and similar exploitative activities. The purpose is to underline such attacks' inherent vulnerabilities and potential impact. Ultimately, the study underscores the need for vigilance and caution among users when navigating the Docker landscape to mitigate these security risks.

The study in [13] provides essential guidelines for organizations to ensure a secure and successful implementation of Docker technology. In this context, a cybersecurity training environment known as a "cyber range" [14] is recommended. This system leverages Docker to facilitate efficient security exercises, effectively simulating vulnerabilities and incidents for training purposes. Containers offer advantages such as resource efficiency and cost-effective deployment. Nevertheless, empirical findings suggest that Dockers are ill-suited for integration into a cyber range environment. It is primarily because when a Docker container is compromised, it can lead to a complete system crash. Consequently, threats rooted in the operating system and kernel vulnerabilities should be carefully excluded from such implementations. The research highlights the intricate balance between the benefits of Docker technology and its limitations, emphasizing the need for strategic decision-making and cautious implementation in cybersecurity training scenarios.

Leveraging large datasets for comprehensive insights, known as big data analytics, values Docker for its adaptability and user-friendliness [15]. This technology proves particularly beneficial in establishing big data clusters through platforms like Hadoop and Spark. User authentication is achieved via MD5 encryption. In [16], researchers assess Docker's performance using hardware tools such as Bonnie++ and psutil. The host exhibits exceptional speed and resource management in Bonnie++, while psutil reveals similar performance between the host and Docker concerning CPU, memory, and network usage. Though the marriage of Docker with the cloud is revolutionary, it contends with significant vulnerabilities, including Container escape, root access, buffer overflow, and Image poisoning. These threats are elaborated upon and proposed measures to mitigate them [17]. The research in [18] pioneers an approach outlining three culpability-driven Docker use cases, offering a nuanced perspective on each. Similarly, in [19], researchers employ CI/CD through Jenkins jobs to conduct image scanning. Opting for a public registry over a private one is advised for image security. Notably, the study identifies malicious DNS requests exposed to an evil image propagating cryptocurrency pursuit via the SSH daemon. For simplified use, building multi-container configurations for web applications atop Docker is endorsed [20]. It is facilitated by Docker swarm technology, which caters to scalability, availability, and load balancing. Due to its advantages, the work underscores the preference for setting up containers over conventional cloud infrastructure.

In [21], an experimental investigation has been conducted to evaluate the performance of Docker when executing heterogeneous microservices. The Cloud Evaluation Experiment Methodology (CEEM) is employed to assess container interference, considering the impact of resource constraints. This approach contributes to formulating intelligent resource allocation within the containerized environment. Similarly, [22] explores the amalgamation of Docker with Blockchain technology, resulting in the container as a Service (CaaS). This innovative integration is underpinned by three major orchestrators: Docker Swarm, Kubernetes, and OpenShift [23]. Among these, Kubernetes is the preferred choice due to its adaptable templates, lenient security policies, easy installation across various platforms, and open-source nature. A crucial advantage is its capability to accommodate multiple masters, enhancing resilience against single-cluster failures.

Despite Docker's efficiency and widespread adoption across various technologies, it grapples with significant cluster synchronization and resource management drawbacks. While certain issues have been addressed at a basic level, higher-level problems remain unattended. As a response, researchers have conceptualized a graphical model tool named Docker Designer. This tool emphasizes stringent validation of Docker during the design phase, aiming to alleviate these concerns. In [24], static code analysis tools such as Go Reporter and Go Meta Liner have been explored. The goal is to dissect the flaws and vulnerabilities, notably the Common Vulnerability Exposure (CVE)-2015-3630. This vulnerability involves files with write access disregarding network security constraints, accentuating security's overarching significance within network environments [25-27].

Thus, the current paper underscores the importance of reducing the privileged user status to that of a non-root user. This seemingly simple action effectively prevents root directory attacks and potential data breaches, as the user's access is confined within their container.

3 Methodology

The current section discusses attack and defence orchestration.

3.1 Orchestrating attacks

The UNIX socket is pivotal in container management, which functions as a conduit for data exchange between software entities. This socket assumes a central role in establishing a connection with the Docker daemon, and Docker clients utilize this socket to issue corresponding Docker commands. The socket is mounted for operational purposes during the adoption of images from online sources and the initiation of Docker containers. However, this socket mounting introduces a vulnerability wherein an attacker with a shell on the container can exploit the situation. Through this shell access, the attacker might escalate their privileges to the root user level, potentially gaining unauthorized access to all files on the host machine.

In this context, the procedure unfolds by creating a file within the root directory. Echo commands are employed to input data into this file. By utilizing the capabilities of the Docker socket and client, containers are initiated on the host system. Following this, the host's root directory is accessed using specific commands executed within the newly launched container, along with the activation of a shell to breach the host's root directory. The 'sh' command is utilized to trigger the shell that facilitates the acquisition of root access. Once the attack gains root access, the host is compromised and may lead to exploitation of files owned by the root user on the host. Fig. 1. outlines the sequence of steps in the orchestrated attack process.

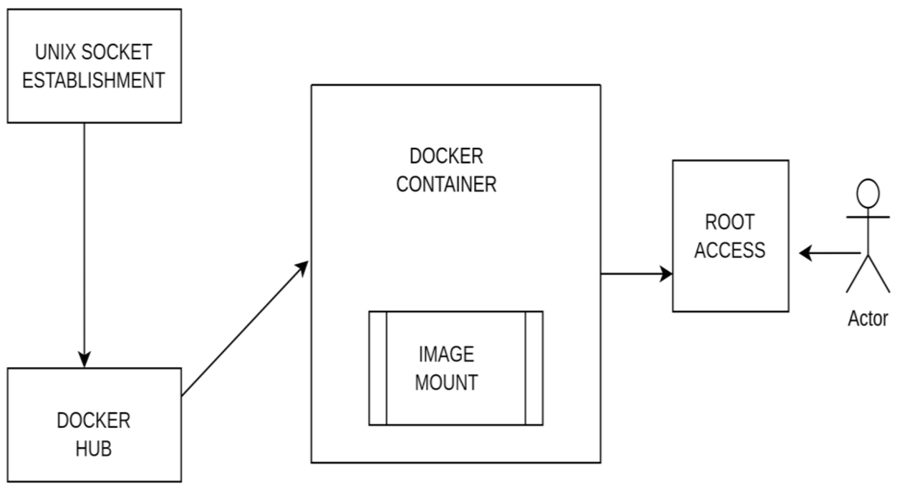

Fig. 1. Sequence of orchestrating an attack in the docker image.

3.2 Defence Orchestration

The Docker Daemon is connected via a UNIX socket instead of a TCP port. The UNIX socket remains under the control of the root user, allowing other users to access Docker by utilizing the sudo command. Consequently, root privileges are the default configuration when executing an application within a Docker Container. This is exemplified by the Ubuntu Docker Container Bash, which logs in as the root user. However, this practice poses a significant risk to application security, allowing potential attackers to compromise the container and various applications. Hence, it is crucial to prioritize the execution of even the most basic operations as a non-root user whenever feasible. This approach is vital to enhance the security posture of Docker containers and safeguard the applications they contain.

The initial action within the defensive process involves generating a fresh Docker image and placing it within the designated directory. By default, the Docker registry retrieves the Ubuntu image, which serves as the base. The *useradd* command appends a novel user with a designated identity using the docker RUN directive. Meanwhile, the USER directive showcases an array of currently logged-in users during the execution of the Docker container associated with the precise image.

The docker image is built and run using the command "*sudo docker build -t my-image"*. To run the docker container associated with the docker image, the command, "*sudo Docker -H unix:///var/run/docker.sock run -it -v /:/test:ro -t my-image bash"* is used. The user is denied access to escape his container by accessing files from the host root user.

Many Docker users either overlook or do not perceive the importance of modifying their user privileges and transitioning to non-root user accounts. These habits frequently expose vulnerabilities, particularly when an application is deployed comprehensively. In such scenarios, malicious actors can manipulate the shared file system and compromise other critical applications within the container. The defense orchestration process is illustrated in Fig. 2, outlining its sequential progression.

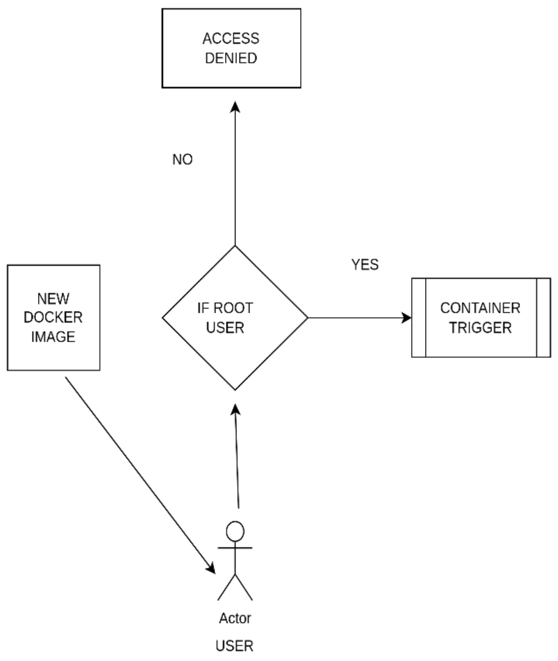

Fig. 2. Progression of defense orchestration

4 Result analysis with image scrutiny

During the stages of security analysis, a significant focus is placed on examining Docker images. This segment compares three notable tools for scanning Docker images: Anchore, Trivy, and Clair.

Anchore tools are proficient at conducting comprehensive scans of images, scrutinizing them thoroughly, documenting their findings, and categorizing them. These tools play a pivotal role in identifying vulnerabilities before they can infiltrate the production environment. Anchore is conveniently available as a Docker image and can be implemented as a standalone application or integrated into an orchestration platform. However, Anchore has a larger footprint and longer scan with a complex interface and configuration, making it less preferred.

For statically surveilling vulnerabilities within containers like Docker and OCI (Open Container Initiative), there exists an open-source tool named Clair. The term “Clair” conveys transparency and clarity, as its primary focus is to provide an insightful view of container-based infrastructure security.

Clair’s functionality is divided into three core components: indexing, where contents undergo scanning and yield an interim report; matching, where vulnerabilities are correlated, and notifications are generated for subsequent actions.

A comprehensive scanner catering to a diverse range of security threats across different targets is Trivy. This tool stands out for its ease of implementation, speed, and reliability. Trivy can scan container images, file systems, and Git repositories. It incorporates various scanners, including Software Bill of Materials (SBOM) for OS packages and software dependencies, Common Vulnerabilities and Exposures (CVE) database, and Infrastructure as Code (IaC) misconfigurations for complaince and transparency benefits.

In Fig. 3, vulnerabilities identified by these three tools are illustrated. Clair and Trivy collectively identified 500 CVEs, whereas Anchore only detected 450. Interestingly, the number of false positives generated in Anchore was much less compared to Trivy. Trivy uses static analysis for image analysis, making it faster in installation and dependable performance; Trivy is the preferred choice among these tools. Whereas Anchore uses both static and dynamic analysis making it complex and less preferred. False positive comparison is shown in figure 4.

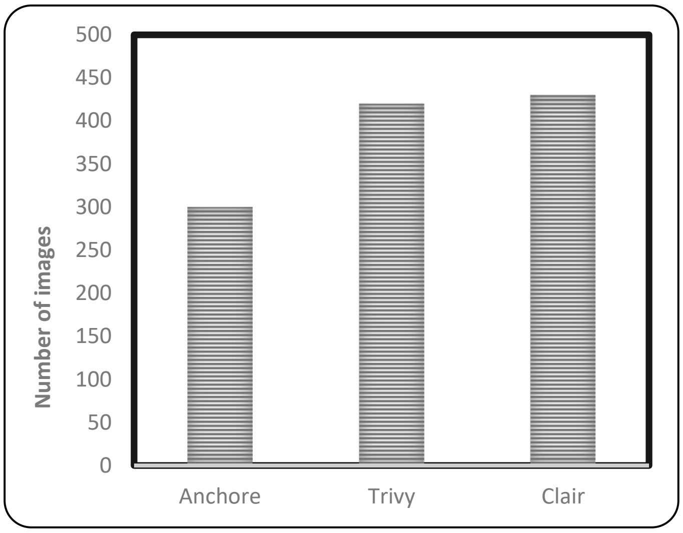

Fig. 3. Comparison of static tools for detecting common vulnerability exposure in docker images

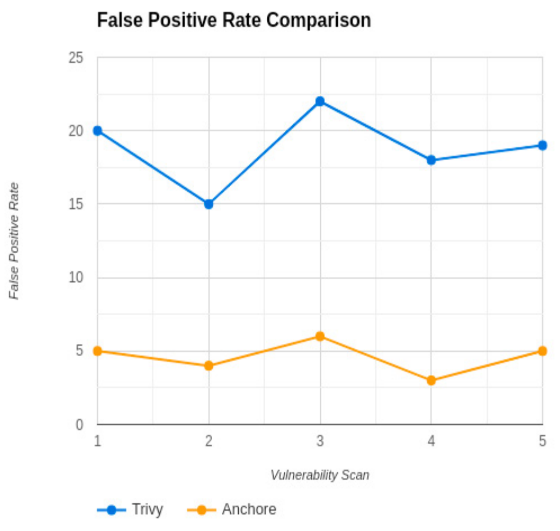

Fig 4. False positive comparison

5 Conclusion

Docker is a commanding technology, fulfilling the virtualization requisites developers sought. Its attributes, including rapidity, portability, and efficiency, have endeared it to the developer community. Notwithstanding these merits, security emerges as a salient concern. Within the context of this paper, a threat model is introduced, addressing the potential for privilege escalation within a Docker-host system. This model spotlights attackers who exploit root file access through sockets to gain unauthorized entry. To counter such risks, a defensive strategy advocating for non-root user operations is proposed to thwart potential container breaches. The effectiveness of a privilege escalation attack hinges upon the intricately woven connections and mutual impact between the kernel and the container. The security of containers relies on the prudent selection and crafting of container images. Given the dynamic nature of environments like Docker containers, proactive monitoring becomes essential. It necessitates the availability of scalable, container-aware tools that exhibit rapid responsiveness. In the future, the incorporation of machine learning techniques shows potential for keeping pace with changing landscapes. Furthermore, an avenue for enhancing security lies in the potential adoption of Docker container encryption, strengthening their safeguarding measures.

References

[1] Aparna Tomar, Diksha Jeena, Preethi Mishra, Rahul Bisht.Docker Security: A Threat Model, Attack Taxonomy and Real-Time Attack Scenario of DoS. International Conference on Cloud Computing, Data Science and Engineering (Confluence), IEEE, 2020.

[2] Vipin Jain, Baldev Singh, Medha Kenwar, Milind Sharma. Static Vulnerability Analysis of Docker Images. IOP Conference Series: Materials Science and Engineering, 2021

[3] Ahmet Efe, Ulas Aslan, Aytekin Mutlu Kara. Securing Vulnerabilities in Docker Images. International Journal of Innovative Engineering Applications, 2020

[4] Tao Lu, Jie Chan. Research of Penetration Testing Technology in Docker Environment. Advances in Engineering Research, 5th International Conference on Mechatronics, Materials, Chemistry and Computer Engineering 2017.

[5] Jyoti Shetty, Raja Rajeswari, Sahana Upadhya, Shoba A State-of-Art Review of Docker Container Security Issues and Solutions. American International Journal of Research in Science, Technology, Engineering & Mathematics, 2017.

[6] Kelly Brady, Seung Moon, Tuan Nguyen, Joel Coffman. Docker Container Security in Cloud Computing, IEEE Annual Computing and Communication Workshop and Conference, 2020.

[7] Peiyu Liu, Shouling Ji, Lirong Fu, Kngjie Lu. Understanding the Security Risks of Docker Hub. Springer, 2020.

[8] Bhupinder Kaur, Aiman Hanna, Mathieu Dugre, Tristan Glatard. An analysis of security vulnerabilities in container images for scientific data analysis.GigaScience, 2021.

[9] Jiang Wenhao, Vulnerability Analysis and Security Research of Docker Container. IEEE 3rd International Conference on Information Systems and Computer-Aided Education, 2020.

[10] Suganthi Subramanian, SS Shylaja, Prasad P Honnavalli. Container Security: An Extensive Roadmap. Atlantis Highlights in Computer Sciences, Proceedings of the 3rd International Conference on Integrated Intelligent Computing Communication & Security, 2021.

[11] Robail Yasrab. Mitigating Docker Security Issues. 2021.

[12] Pooja P, Puneeth.An approach of exploiting Docker container security. International Journal of Advanced Research in Engineering and Technology,2020.

[13] Murugaiah Souppaya, John Morello, Karen Scarfone. Application Container Security Guide. NIST Special Publication 800-190.

[14] Ryotaro Nakata, Akira Otsuka. Evaluation of Vulnerability Reproducibility in Container-based Cyber Range. Proceedings of the 7th International Conference on Information Systems Security and Privacy, 2021.

[15] Gu Ruijuin. A Lightweight Experimental Platform for Big Data Based on Docker Containers. Journal of Physics, 2020.

[16] Preeth EN, Mulerickal FJ, Paul B, Sastri Y. Evaluation of Docker containers based on hardware utilization. In 2015 International Conference on Control Communication & Computing India 2015 Nov 19 (pp. 697-700). IEEE.

[17] Silva JP, Assis A, Martins M, Oliveira R. The Challenges of the Data Privacy in the Cloud using Docker: A Systematic Review. Anais da IV Escola Regional de Informática do Piauí. 2018 Oct 16:190-5.

[18] Martin A, Raponi S, Combe T, Di Pietro R. Docker ecosystem–vulnerability analysis. Computer Communications. 2018, 122, 30-43.

[19]Manish Kumar Abhishek, Rajeswara Rao, "Framework to Secure Docker Containers", Fifth World Conference on Smart Trends in Systems Security and Sustainability, 2021.

[20] Vivek Sharma, Harsh Kumar, Akhilesh Kumar Singh. Docker for multi-containers web application. International Conference on Innovative Mechanisms for Industry Applications, 2020.

[21] Devki Nandan Jha, Saurabh Garg, Premprakash Jayaraman, Rajkumar Buyya, Zheng li. A Holistic Evaluation of Docker Containers for Interfering Microservices. IEEE International Conference on Services Computing, 2018.

[22] Priyanka Kumar, Maharishi Shah. To build scalable and portable Blockchain Application using Docker. International Journal of Computer Networks & Communications, Springer Singapore, Singapore, 2020.

[23] Martin Kontsek, Marek Moravcik. Overview of Docker container orchestration tools. International Conference on Emerging eLearning Technologies and Applications, 2020.

[24] Ana Duarte, Nuno Antunes. An Empirical Study of Docker Vulnerabilities and of Static Code Analysis Applicability. Latin-American Symposium on Dependable Computing, IEEE, 2018.

[25] Mathi S, Srikanth L. A New Method for Preventing Man-in-the-Middle Attack in IPv6 Network Mobility. In Advances in Electrical and Computer Technologies 2020 (pp. 211-220). Springer, Singapore.

[26] Mathi S, Joseph E, Advaith MS, Gopikrishna KS, Gopakumar R. A flattened architecture for distributed mobility management in IPv6 networks. Journal of Intelligent & Fuzzy Systems. 2020 Jan 1;38(5):6583-93.

[27] Shakil Muhammed, Bashir Ali Khasif, Arul Rajakumar. A novel dynamic framework to detect DDoS in SDN using metaheuristic clustering. Transactions on Emerging Telecommunications Technologies, Vol. 33, 2022.