ISSN: 2278-4632
Vol-14, Issue-7, No.03,July: 2024

AN ANALYSIS OF VULNERABILITIES AND SECURITY ASPECTS OF DOCKER CONTAINERS

Vipin Jain, Research Scholar, Department of CSE, Vivekananda Global University, Jaipur, India - 303012& Associate Professor, Department of IT, Swami Keshvanand Institute of Technology, Management Gramothan, Jaipur, India-302017
Dr. Baldev Singh, Professor, Department of CSE, Vivekananda Global University, Jaipur, India - 303012
Dr. Nilam Choudhary, Associate Professor, Department of CSE,Swami Keshvanand Institute of Technology, Management Gramothan, Jaipur, India-302017

ABSTRACT :
Docker is one of the most popular containerization platforms for building, shipping, and running applications in containers as it provides a lightweight and efficient way of packaging and distributing software applications. It has been the most popular containerization solution since its introduction in 2013, owing to its portability, smooth deployment, and setup. However, with its widespread use comes the need for understanding and addressing the security risks and vulnerabilities associated with Docker.This paper explores the security risks associated with Docker and the steps that can be taken to mitigate them. The vulnerabilities that can arise due to improper configuration, weak passwords, and unpatched software are discussed in detail. Additionally, this paper also discusses the measures that can be taken to secure Docker containers, such as using strong passwords, regularly updating the software, and implementing access controls. Furthermore, the paper highlights the importance of security tools and techniques that can be used by different organizations to effectively protect Docker environments from potential security threats.
Keywords: Containers, Security, Docker, Virtualization, Vulnerabilities, Risks, Measures, Attacks.

INTRODUCTION :

Docker was first published in 2013, but its first final version was not until June 2014. Docker began as a Platform as a Service (PaaS) called dotCloud, but it was ultimately released as a Docker project. It is a free and open platform that enables users to create, share, and execute programs [1]. Containers are a collection of components that enable us to build an environment in which applications may run irrespective of the operating system being used [2]. These are significantly lighter than virtual machines and the code that runs in these containers is isolated from one another. Containers can share the machine's resources without the cost that the hypervisor layer adds while Virtual Machines on the other hand require the installation of an operating system, disc allocation, CPU, and RAM to function [3].

Virtual Machine

Virtualization is a notion that was most probably first developed by IBM in the late 1960s and the early 1970s. It enables the production of valuable IT services by utilizing hardware-bound programs [4]. It spreads the machine's capabilities among users or locations, allowing us to utilize the system's absolute capacity. A physical computer may operate numerous Virtual Machines (VMs) with various Operating Systems without conflict when employing virtualization since they are segregated from one other [5]. The component accountable for making this possible is the virtual machine monitor (VMM), also recognized as the hypervisor. This component is in charge of establishing a virtual environment between the hardware of the physical computer and the virtual machines [2].

Container

A container provides an isolated place for individual programs to run. Unlike the virtualization of operating systems, containerization allows for the creation of smaller images while still utilizing the same operating system, allowing for lighter workloads [3]. Containerization has grown rapidly due to the numerous benefits it provides in terms of ease, flexibility, efficiency, and scalability. As a result,

ISSN: 2278-4632
Vol-14, Issue-7, No.03,July: 2024

many businesses are moving their web applications to this model. According to IT Trends magazine, "Container technology is developing, which is fueling the market for orchestration tools, which might increase at a compound annual rate of 17.2% in the next few years" [1].

Containerization vs Virtualization

With virtualization and containerization, applications may be segregated, allowing for cross-environment functioning. The main differences lie in mobility and size. Since virtual machines are bigger (measured in gigabytes) and have their operating system, they may do several resource-intensive activities at the same time. Because they have more capacity, VMs may abstract, divide, replicate, and imitate whole servers, operating systems, desktops, databases, and networks [6]. A container is typically a few gigabytes in size, and it mainly includes space for software and its execution environment. Containers were created to be compatible with newer and developing technologies like clouds, CI/CD, and DevOps, whereas virtual machines (VMs) work well with the traditional, monolithic IT architecture. The differences between virtual machines and containers are listed in Table 1 [7].

Table 1: Virtual Machines Vs Containers

Virtual Machines(VMs)	Containers
Represents virtualization at the level of hardware	Represents virtualization at the level of the Operating system.
Heavy	Light
Slow Provisioning	Provisioning and scalability in real-time
Limited performance	Native performance
Complete isolation,therefore,is safer	Isolation at the process level, therefore, is less secure

DOCKER ARCHITECTURE :

Docker has a client-server design, with three key components: the Docker Host, Docker Client, and Docker Registry as depicted in Figure 1. The Docker Client command line interface communicates with the Docker daemon using commands and REST APIs [5]. When a Docker command is executed on the client terminal, it is sent to the Docker daemon, which retrieves the appropriate image from the Docker repository (known as Docker Hub) and deploys a container based on that image. The Docker Host manages the execution of applications and their operating environments. It includes storage, containers, images, and the Docker daemon. Docker images are managed and stored by the Docker Registry. When a program is executed in a container, the output is sent from the Docker daemon to the Docker Client, which then sends it to the terminal. Docker Engine, which is built in Golang and runs on native Linux systems, is the layer on which Docker operates. Docker does not currently support checkpointing, restoring, or live migration across hosts, but this capability may be added in the future[6].

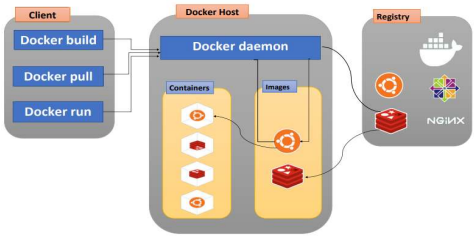

Figure 1: The Docker Platform

RELATED SURVEY WORK :

In a published research paper [8], an investigation was carried out to explore the vulnerabilities that exist in the Docker Environment and the potential safety implications that arise when using containers with traditional applications. Docker offers a variety of container security choices and is

ISSN: 2278-4632
Vol-14, Issue-7, No.03,July: 2024

more than just a package approach, but rather a complete production and distribution network. The research included a thorough survey of relevant tasks in the field, which were categorized according to protection, and the security environment within containers was evaluated [9]. Using a top-down strategy, the research explicitly found various vulnerabilities present in specific parts of the Docker environment, whether during the creation or execution of any unique use cases. Moreover, the research highlighted actual situations where specific vulnerabilities can be exploited and suggested potential solutions, which include addressing issues related to Docker-provider PaaS implementation [10]. The paper does, however, address security issues with containers, such as their lightweight character and use of the same kernel as the Host operating system. The research showed four scenarios and their answers, which take into account all of the security protocols for the hosted container, resulting in complete guidance for safe Docker deployment. The use of container-based virtualization is more rapid and lighter in comparison to other types of virtualization. However, there are some security concerns with this form of emulation. For container-based virtualization, Docker, an open-source technology, is used [11]. As a result, a thorough investigation was performed to evaluate Docker's domestic security and to investigate how Docker can be combined with Kernel security features such as SELinux and AppArmor. Docker's security could be improved if it is run as a non-privileged process and an additional layer of protection is applied using AppArmor or SELinux. [12].Docker is a popular choice among software developers due to the flexibility, portability, and scalability it offers. However, Concerns regarding vulnerabilities have grown in tandem with the growing relevance of the privacy of images that form the basis of applications. With the move in development operations to the cloud, evaluating the security of pictures from various sources is crucial. We propose a continuous integration and continuous deployment (CI/CD) solution that validates the privacy of Docker images across the software development life cycle in this article. We present photos with faults and assess the efficiency of our technique in detecting problems. Moreover, we demonstrate how dynamic analysis may enhance static research for security assessments by measuring the security of Kubernetes depending on their activities. [13].

DOCKER VULNERABILITIES :

The kernel is one of the docker ecosystem's weak areas [14]. It is recommended to use containers with the lowest privilege level with the default Docker configuration, which has recently been improved in terms of security. This is not only safer, but it also provides a higher level of isolation. Docker has several especially problematic features as a result of its broad adoption. Numerous research papers discuss specific attacks such as namespace exploitation and other subjects. The Common Vulnerabilities and their exploitation is shown in Table 2. Containers have full access to the kernel of the host and as a result, it is possible to target and compromise the host, resulting in sudo access or system blocking [15].

Table 2: Docker Vulnerabilities Effects

Vulnerabilities Effect	Description
Gain Privileges	Unauthorized privileges are obtained by an attacker, opening up access to system restrictions.
Execute Code	Attackers can execute malicious code.
DoS	This attack may cause the system's resources to run out, decreasing its accessibility.
Bypass	The attacker can circumvent security checks and get access to the system without authorization.
Gain Information	Unauthorized access to private data is obtained by an attacker, which can be exploited to target other flaws.

When a vulnerability in software is exploited, it allows attackers to acquire control, steal information, or affect the functionality of the product. Although there isn't much standard

ISSN: 2278-4632
Vol-14, Issue-7, No.03,July: 2024

classification of vulnerabilities, various organizations and researchers have tracked and publicized exploit findings by the year. The CWE is in charge of categorizing the vulnerability's class, also known as the vulnerability type. Some of them are :

Injection: When an attacker delivers a malicious script to an interpreter, various problems emerge. The data source may be abused as an injection vector, leading to data loss, distortion, information disclosure, or denials of access.

Sensitive Data Exposure: Vulnerable data may allow a hacker to get keys, perform man-in-the-middle hacks, or steal plain text data, allowing access to classified information.

Broken Access Control: An attacker can obtain privileges, circumvent access control, and edit sensitive files if authenticated user restrictions are not effectively enforced.

Security Misconfiguration: Attackers can get unauthorized access and privileges to the system due to security misconfiguration or a lack of upgrading.

Insecure Deserialization: Exploiting deserialization results in the execution of code remotely, injection of attack, and privilege escalation.

Leveraging Vulnerabilities in Components: The usage of susceptible components facilitates the exploitation of such system vulnerabilities, allowing for a wide range of attacks and consequences [16].

DOCKER SECURITY :

The latest updates to Docker technology have enhanced automated security by adding layers between applications and hosts, reducing the host surface and thereby preventing unauthorized access to both the host and containers [5]. Administrators are encouraged to modify procedures and imposerestrictions on any unused applications. The Docker container model supports and enforces these restrictions by enabling the use of multiple user accounts and running programs within their root systems. Docker further improves security by limiting the resources that containers can access, use,and observe, as well as their interactions with the hosting system and other containers.Docker Containers with Linux word spaces and clusters provide application sandboxing and constraints.

Docker simplifies the usage ofthese powerful partitioning mechanisms, which have long been part of the Linux kernel and allow administrators to establish and maintain barriers for distributed programs as if they were independent and separate entities as shown in figure 2 [8].

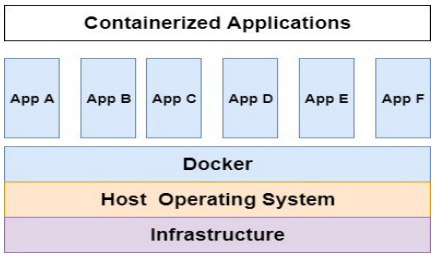

Figure 2: Containerization of Application with docker

Namespaces: Linux namespaces are used to offer the container workspace, which is an isolated workspace. When a container is deployed, Docker creates a set of namespaces for it to isolate it from all other running containers[5].

Control groups: Control groups (also known as c-groups) are the kernel-level feature that allows Docker to regulate which resources each container may access, ensuring successful container multi-tenancy [8].Control groups allow Docker to share existing hardware resources or, if necessary, impose container constraints and barriers. An excellent example of this is the application of memory restrictions on certain containers to remove household resources and prevent potential exploits, as discussed in the sections that follow [9].

Seccomp: Docker Engine supports the Linux kernel's secure computing mode (seccomp). As a consequence, the controller can limit the processes that can be executed within the container to one

system call. This feature limits the actions that your application container may conduct while it is executing on the host system. The default seccomp profile for Docker is a white list of approved calls, that blocks more than 50 distinct syscalls. The default profile should work perfectly with the vast majority of applications. In practice, the default profile may be modified frequently to protect against a few unknown Linux concerns for Dockerized apps. This is referred to as a security non-event [10].

Process Restrictions: Modern Linux has grown to allow for more nano-assisted model: capabilities, whereas the traditional Linux idea of OS security considers root privileges in terms of user rights against root privileges [4]. Limiting access and authority reduces the possible attack surface.Because of Linux features, granular user access may be configured. The root user has total authority by default; non-root users have fewer privileges but can get access to the root level by using sudo or setuid pairs which might endanger your safety. Docker's default settings are designed to limit Linux's capabilities and mitigate this risk[11].

Device and File Restrictions: By restricting physical devices' access to host-based containerized software, Docker has reduced the attack surface. The previously stated device resource control groups (cgroups) method is used to accomplish this. Containers cannot be allowed device access automatically; it must be explicitly granted [11].

DOCKER SECURITY MODULES :

Docker supports many security measures, such as kernel security, namespaces, and cgroups, to protect the Docker daemon, kernel features, and container settings, from possible attacks. This section discusses Docker's additional tools for container security, including AppArmor, Linux Capabilities, and the Docker Daemon [12].

I. **AppArmor** is a Linux kernel component that protects the operating system against application security risks. It enables the creation of security profiles for each program, allowing users to govern which system resources and features each application has access to. Docker has a default profile called docker-default, but custom profiles can also be created for containers [8].

II. **Linux Capabilities** are unique components associated with administrative privileges. Before kernel version 2.2, there were two types of permissions: privileged and unprivileged. Since version 2.2, rights can be individually activated or removed through Capabilities. The host is responsible for most of the container's tasks, and in most cases, containers only need rights to specific Capabilities rather than root privileges. This allows for flexible configuration of Docker containers, allowing users to add and remove capabilities as needed [15].

III. The Docker daemon is the program in charge of container management and image preparation for creating, executing, loading from a disc, or fetching from a repository. Because the daemon operates with root access, only trustworthy users should run it. Since its first release, the Docker daemon has undergone several modifications to become safer and less susceptible to unscrupulous users who may attempt to construct arbitrary containers. One of the adjustments was the substitution of UNIX sockets for a RESTful API endpoint. Docker's documentation describes the daemon as "possibly susceptible" while loading data such as pictures. Docker has saved every image with cryptographic checksums on their datasets since version 1.10.0 to avoid collision attacks with previously present image files in the system, making this functionality less susceptible [16].

IV. According to [17], the authors have proposed a new security mechanism named Docker-sec for Docker, which uses AppArmor. Docker-sec adds a layer of protection to Docker's standard security settings by producing separate AppArmor profiles for every container. In contrast to Docker's standard AppArmor policy, the extended AppArmor policy provided by Docker-sec offers three major advantages. Firstly, it safeguards the container over its full life cycle by generating secure profiles for all important Docker components. The standard

ISSN: 2278-4632
Vol-14, Issue-7, No.03,July: 2024

AppArmor profile, on the other hand, only protects the containers after it has been started by the RunC. Furthermore, rather than having a basic secure profile that is suitable for all containers, it produces a unique AppArmor profile for each container. Lastly, the AppArmor profile for every container is changeable and can be changed to accommodate any modifications in the container's behavior by watching the container's behaviors throughout the training phase and identifying the rights that are truly required for the container to run efficiently.

V. The LiCShield framework was proposed in [18] as a solution for protecting Docker containers and their payloads by automatically producing AppArmor policies for both the hosts and the Docker container. LiCShield does this by utilizing the SystemTap tool to trace all kernel operations whereas the Docker daemon is constructing and conducting operations. The remnants are then translated into AppArmor rules, which are subsequently used to build two distinct AppArmor profiles: one is for processes within the container and the other is for actions on the host.

While the architectural ideas of automated AppArmor profile creation are shared by LiCShield and Docker-sec. LiCShield provides various advantages, such as employing SystemTap as a monitoring tool, which happens to be more versatile than Docker-Auditd. sec's LiCShield can construct rules that Docker-sec could not, such as the pivot root rule, mount rule, access rule, execution rule, and link rule. Additionally, LiCShield creates the profile for the Docker daemon depending on its operations, limiting the power to the bare minimum necessary for successful operation. However, Unfortunately, LiCShield has several shortcomings when compared to Docker-sec, especially with the most recent Docker versions. LiCShield does not create a RunC profile and fails to offer adequate runtime container security [19]. This is because LiCShield depends on the host profile's pivot root rule to perform container security. Before actually initiating container operations, the Docker daemon called the pivot root, facilitates the move from the host profile to the container profile. Nevertheless, RunC does this work in current Docker releases, and the pivot root rule could not be utilized for this reason.

PROPOSED SOLUTIONS AGAINST ATTACKS :

This section discusses the areas in Docker containers where the attack may arise with their prevention solutions.

Kernel Attacks:

Docker containers share the host kernel. Even minor flaws in the container can swiftly bring the host kernel to a halt. However, because virtual machines provide two layers of protection, a process running within one is much less likely to disrupt the kernel host. To access the host kernel, the process would need to modify the machine's kernel first, followed by the hypervisor layer [20].

One way to avoid this attack is to deploy containers alongside VMs. Using this hybrid technique, separate services that must be kept apart may be bundled into containers, which can be placed inside virtual machines. Other ways to prevent exploits of the kernel are[11]:-

I. **By avoiding the - -privileged flag:** The "- -privileged" flag eliminates the cgroup controller's constraints and also grants privileges to the containers, which might lead to a possible attack.

II. **Setting the volumes to read-only:** To avoid malicious updates, set the files that no longer need to be edited to read-only.

III. **Install only the required packages in the container.**

IV. **By using secure seccomp profiles:** The Secure Computation Mode (seccomp) of the Linux kernel is utilized to restrict the processes permitted within the container. The seccomp() system function restricts application access by modifying the seccomp state of the calling process. This functionality will be useful only if Docker is built with seccomp and the kernel has CONFIG SECCOMP enabled.

Denial of Service (DoS) Attacks :

Docker containers operate in a way that each container believes is the only process running on the system. When the ps command is run within a container, it will only show the processes currently running in that container. However, if a container loses its isolation, it may begin to consume host system resources such as CPU time, memory, user IDs, etc. It can also potentially interfere with other containers and prevent users from accessing certain parts of the system. Some ways to prevent DoS attacks are [11]:.

I. **Limiting the CPU shares for containers**: By default, all containers receive an equal percentage of CPU cycles, with a total weight of 1024 [7]. This percentage may be changed at runtime by adjusting the container's CPU share weighting relative to the weighting of all other containers that are executing.

II. **Setting the container file system to read-only:** If it is not necessary to edit files in a container, set the filesystem to read-only.

III. **Turning off inter-container communication**: By default, all containers on the same host have unlimited network traffic enabled. The only communication between containers that have been explicitly linked together is enabled when containers are run with icc = false.

IV. **Setting the amount of memory a container can use**: Another feature of Docker is the ability to limit the amount of RAM that a container may use. We can prevent a container from draining the resources of other containers by restricting the amount of memory it can use.

V. Containers are widely used for applications that require access to databases and other services on the host system. Database passwords and API keys may be required for the bulk of these services. An attacker with access to these secrets can also use these services. As a result, these keys and passwords must be kept safe.

VI. **Environment Variables should not have confidential information**: When debugging, environment variables are readily leaked and accessible in too many locations, including child processes and connected containers. It is safer to use volumes to pass secrets in a file rather than environment variables.

VII. **Setting container file systems to read-only**: Setting read-only container file systems and running containers without the - -privileged argument, as explained above, can both assist in improving security.

CONCLUSION:

Docker provides a powerful platform for deploying applications in containers, but it also presents significant security risks and vulnerabilities that must be addressed. Organizations that fail to properly secure their Docker environments risk compromising sensitive data, disrupting business operations, and damaging their reputations. Today, several efforts have been made to address these vulnerabilities, but there is still room for improvement.To mitigate these risks, organizations must adopt a proactive approach to Docker security by implementing best practicesand leveraging security tools and techniques. This research paper proposes solutions to many potential security issues and flaws that can be prevented by following the recommendations outlined in the preceding sections. Even with secure containers and images, there may still be risks and potential consequences. With careful planning and attention, the potential damage can be greatly reduced or even avoided. As Docker continues to evolve, these issues will continue to be researched and addressed.

REFERENCES :

1. Martin, A., Raponi, S., Combe, T. and Di Pietro, R.: “Docker ecosystem–vulnerability analysis,” Computer Communications, vol. 122, pp. 30–43 (2018).
2. Rodrigues, L.R., Koslovski, G.P., Pasin, M., Pillon, M.A., Alves, O.C. and Miers, C.C.: “Time-constrained and network-aware containers scheduling in GPU era,” Future Generation Computer Systems, vol. 117, pp. 72–86 (2021).

ISSN: 2278-4632
Vol-14, Issue-7, No.03,July: 2024

3. Dziurzanski, P., Zhao, S., Przewozniczek, M., Komarnicki, M., and Indrusiak, L.S.: "Scalable distributed evolutionary algorithm orchestration using Docker containers," Journal of Computational Science, vol. 40, pp. 101069–101083 (2020).
4. Kwon, S. and Lee, J.H.: "DIVDS: docker image vulnerability diagnostic system," IEEE Access, vol. 8, pp. 42666–42673 (2020).
5. Reis, D., Piedade, B., Correia, F.F., Dias, J.P. and Aguiar, A: "Developing docker and docker-compose specifications: a developers' survey," IEEE Access, Vol. 10, pp. 2318–2329 (2022).
6. Alyas, T., Javed, I., Namoun, A., Tufail, A., Alshmrany, S. and Tabassum, N.: "Live migration of virtual machines using a mamdani fuzzy inference system," Computers, Materials & Continua, vol. 71, no. 2, pp. 3019–3033 (2022).
7. Tabassum, N., Ditta, A., Alyas, T. et. al.: "Prediction of cloud ranking in a hypercon-verged cloud ecosystem using machine learning," Computers, Materials & Continua, vol. 67, no. 3, pp. 3129–3141 (2021).
8. Jain, Vipin, Singh, Baldev and Choudhary, Nilam: "Audit and Analysis of Docker Tools for Vulnerability Detection and Tasks Execution in Secure Environment." International Conference on Emerging Technologies in Computer Engineering. Springer, Cham, (2022).
9. Shu, R. and Enck, W.: "A study of security vulnerabilities on docker hub," in Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy, vol. 5, pp. 269–280, Scottsdale, AZ, USA, March 2017.
10. Jiang, Y., Liu, W., Shi, X. and Qiang, W.: "Optimizing the copy-on-write mechanism of docker by dynamic prefetching," Tsinghua Science and Technology, vol. 26, no. 3, pp. 266–274, (2021).
11. Sultan, S., Ahmad, I. and Dimitriou, T.: "Container security: issues, challenges, and the road ahead," IEEE Access, vol. 7, pp. 52976–52996 (2019).
12. V. Medel, R. Tolosana, J. Bañares, U. Arronategui, and O. Rana, "Characterizing resource management performance in Kubernetes," Computers & Electrical Engineering, vol. 68, pp. 286–297, (2017).
13. A. P´erez, G. Molt´o, M. Caballer, and A. Calatrava, "Serverless computing for container-based architectures," Future Generation Computer Systems, vol. 82, (2018).
14. R. Morabito, "Virtualization on the Internet of Things edge devices with container technologies: a performance evaluation," IEEE Access, vol. 5, pp. 8835–8850, (2017).
15. R. Shu and W. Enck, "A study of security vulnerabilities on docker hub," in Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy, vol. 5, pp. 269–280, Scottsdale, AZ, USA, March (2017).
16. B. Rad, B. Bhatti, and H. Ahmadi, "An introduction to docker and analysis of its performance," International Journal of Computer Science and Network Security (IJCSNS), vol. 17, no. 3, p. 228, (2017).
17. A. Martin, S. Raponi, T. Combe, and R. Di Pietro, "Docker ecosystem–vulnerability analysis," Computer Communications, vol. 122, pp. 30–43, (2018).
18. L. R. Rodrigues, G. P. Koslovski, M. Pasin, M. A. Pillon, O. C. Alves, and C. C. Miers, "Time-constrained and network-aware containers scheduling in GPU era," Future Generation Computer Systems, vol. 117, pp. 72–86, (2021).
19. N. Tabassum, T. Alyas, M. Hamid, M. Saleem, and S. Malik, "Hyper-convergence storage framework for eco-cloud correlates," Computers, Materials & Continua, vol. 70, no. 1,pp. 1573–1584, (2022).
20. P. Dziurzanski, S. Zhao, M. Przewozniczek, M. Komarnicki, and L. S. Indrusiak, "Scalable distributed evolutionary algorithm orchestration using Docker containers," Journal of Computational Science, vol. 40, pp. 101069–101083, 2020.