Hindawi
Security and Communication Networks
Volume 2022, Article ID 6819002, 11 pages
https://doi.org/10.1155/2022/6819002

Research Article

Container Performance and Vulnerability Management for Container Security Using Docker Engine

Tahir Alyas,[1] **Sikandar Ali**,[2] **Habib Ullah Khan**,[3] **Ali Samad**,[4] **Khalid Alissa**,[5] **and Muhammad Asif Saleem**[1]

[1]*Department of Computer Science, Lahore Garrison University, Lahore 54000, Pakistan*
[2]*Department of Information Technology, The University of Haripur, Haripur 22620, Khyber Pakhtunkhwa, Pakistan*
[3]*Department of Accounting and Information Systems, College of Business and Economics, Qatar University, Doha 2713, Qatar*
[4]*Faculty of Computing, The Islamia University of Bahawalpur, Bahawalpur 63100, Pakistan*
[5]*Networks and Communications Department, College of Computer Science and Information Technology, Imam Abdulrahman Bin Faisal University, P.O. Box 1982, Dammam 31441, Saudi Arabia*

Correspondence should be addressed to Sikandar Ali; sikandar@uoh.edu.pk

Received 24 April 2022; Accepted 21 July 2022; Published 10 August 2022

Academic Editor: Muhammad Shafiq

Containers have evolved to support microservice architecture as a low-cost alternative to virtual machines. Containers are increasingly prevalent in the virtualization landscape because of better working; containers can bear considerably less overhead than the conventional hypervisor-based component virtual machines. However, containers directly communicate with the host kernel, and attackers can co-locate containers in the host system quicker than virtual machines. This causes significant security issues in container technology. The security hardening system is currently targeted at implementing universal access management regulations that make it difficult to assess the required procedure for accessing containers. Security mechanisms include an explicit awareness of the purpose and actions of the container and entail manual interaction and configuration. A user-friendly container protection scheme implemented an access policy to comply with its anticipated and legitimate application performance. In this study, container technology constraints have been overcome by proposing a unique Docker-sec mechanism. Docker-sec uses four mechanisms; the original collection has been improved during container runtime by additional rules that constrain the capacity of the container, further representing the applications in practice, file system, processes, network isolation, and vulnerability scanning of Docker images over different workload. Different vulnerabilities have been scanned with a CVE severity level. Results showed that inter-container communication with the system is more secure containers from zero vulnerabilities with an overhead of 3.45%.

1. Introduction

Cloud computing prevails over traditional on-premises environments for many reasons, including discounted prices, seemingly unlimited pay-as-you-go resources, extensibility, and ease of storage. When deploying software and cloud service providers, it may be necessary to allow multiple self-managing virtual systems to use a related group of physical resources, one of the main drivers of cloud computing [1]. Containers have recently been based on a simplified approach to virtualization with a wide range of advantages over traditional "Container Machines with Hypervisor Alternatives." In particular, containers have much less overhead than virtual machines (virtual machines) when they run through a kernel that they share with the host computer as user-space operations. In addition, the device modules can be used as lightweight units, and their delivery and execution are simplified. This allows for the automatic control of large-scale systems [2].

Containers have been used successfully in various applications, and although many new technologies are being introduced in parallel with containers, this has become more common over time [3]. Containers can run on the same operating system, making the server more efficient and allowing for faster application deployment. Images bundle multiple containers into a single operating system and multiple such operating systems on a server, just like on a commercial machine, where one container sits on top of another. Containers perform better to improve software reliability as they switch from one IT environment to another. The container reduces the complexity of the application platform, its dependency, and its structure.

In short, everything involves building the container in whatever running environment, application, and all the libraries, triggers, and configuration files needed to run it. Unlike virtualization technology, which includes virtual machines that span the entire operating system and applications, one controller and three operating systems run on a physical server with three virtual machines. Judging by the cloud environment, container technologies' low implementation level and several studies have identified security problems [4].

While it provides isolation for a specific resource such as processes, file systems, networks, and so on, by introducing quotas for CPU, RAM, and disk usage, containers are much more vulnerable to attacks than virtual machines. Usage is not just the absence of a hypervisor, making containers more vulnerable to attacks than virtual machines. Since containers and hosts use the same kernel, malicious containers can quickly leave their environment and make host kernel attacks possible. Mandatory kernel access control is the best way to improve the security of a Linux container by using tools such as AppArmor or SELinux [5] to prevent unintended operations on both the host and container sides. However, this is a tedious procedure, requiring the application to know its properties and requirements and develop specific safety rules manually. Attempts are underway to automate the extraction of MAC rules [6] on a per-image basis rather than per-file, leaving room for inter-container attacks.

The Core Docker architecture is divided into three main components as shown in Figure 1.

(1) Docker client
(2) Docker Host
(3) Docker registry

Docker Client run, pull, and push the container to the Docker daemon. The Docker daemon is a program that runs in the background. The Docker engine is a Docker component responsible for creating and running Docker containers. A live operating instance of a docker image is referred to as a docker container. Docker Engine is a client-server program that includes the following elements:

(i) A daemon process is a server that is a continuous running service.
(ii) A REST API that allows programs to communicate with the daemon and give it instructions.
(iii) A client with a command line interface.

To overcome, security issues, this study proposed Docker-sec, a user-friendly container protection scheme, by implementing an access policy, in compliance with its anticipated and legitimate application performance. Docker-sec, an automated open-source and easy-to-use "Docker" security system, and typically compatible containers are seen to overcome those shortcomings. It generates a container based on the container settings to provide an initial static list of access rules. Docker-sec uses four mechanisms. The original collection is improved during container runtime by additional rules that constrain the capacity of the container, further representing the applications in practice, file system, processes, network isolation, and vulnerability scanning of Docker images. Docker-sec defends the container from outbreaks on both the server and the container engine, thus reducing container admittance to the resources required for the application to run. For all essential parts of the Docker architecture, it utilizes AppArmor by adding protected profiles on each. Container profiles are created on (a) vulnerability assessment container accomplishment parameters and (b) dynamic control behavior of container actions throughout the operation. Docker-sec enables users to automatically create initial profiles based on the settings provided when initializing the container (for example, only specific directories and files are mounted). Docker-sec will boost the initial profile dynamically with rules extracted during the preproduction phase by tracking the container's execution in real-time if more rigorous protection protocols are required. Docker-Sec will secure containers from zero-day vulnerabilities through the source of its two mechanisms and has only a marginal overall effect on the application output because it was developed [7].

Separation aims mainly to avoid the infected containers communicating with other containers through process control interfaces. The host and container file systems must be protected from unauthorized access to keep the file system isolated. To cater to network-based attacks, including Man-in-the-Middle (MitM) and ARP spoofing, it is essential to isolate the network from such types of attacks. Containers are designed not to eavesdrop on or manage network traffic or server [8]. The Docker base proposed methodology reveals Docker's ability to dynamically draw up new artist rules that limit the container functionality necessary in its setup (via our vital monitoring tool). All methods can have been monitored for containers running in an OpenStack IaaS private cluster. Docker-sec would improve the Docker Web Management user interface to allow participants to communicate with the program, selecting various application containers and simulated attacks [9].

In today's cloud computing settings, container-based virtualization has become the primary option. A significant difficulty for cloud suppliers and customers is finding and evaluating container anomalies. This study offers an online container anomaly detection system by monitoring and analyzing the multiple resource metrics of the containers based on the improved isolation forest method [10].

You may find flaws before they become serious by routinely checking for new vulnerabilities. It would help if you mapped the image vulnerabilities you find to the

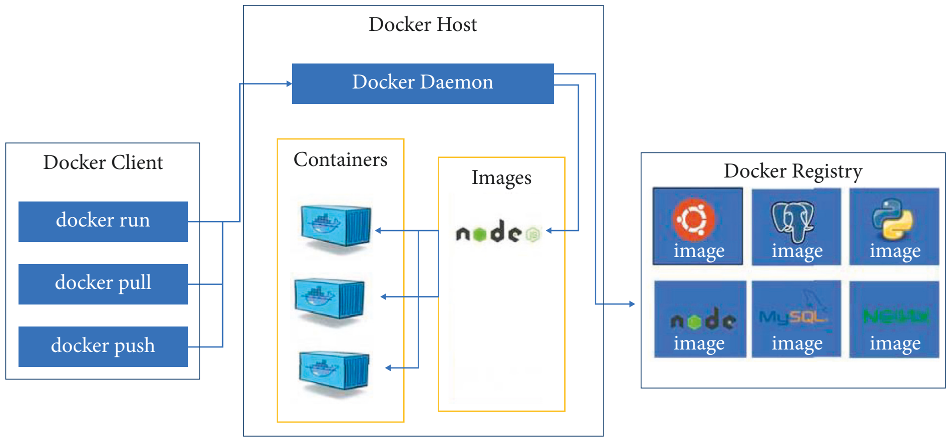


Figure 1: Docker architecture.

operating containers to address any security concerns more quickly. Requiring the usage of only authorized container registries and prohibiting the use of unapproved container images both assist you to avoid introducing vulnerabilities into your developing and container environments [11].

1.1. Security Vulnerabilities in Containerization. Vulnerabilities in container images often result from dependencies being imported that are insecure libraries or other dependencies. Additionally, malicious code that was added during a breach of the development platform or a software supply chain assault may be included in images. There are many types of vulnerabilities in containers like SQL, overflow, DoS, cross-site request forger (CSRF), and cross-site scripting (XSS). The main drawback of container scanning is that it often fails to identify undiscovered vulnerabilities, flaws that have not yet been made known to the general public or otherwise accessible to security experts. To put it another way, if your container image makes use of a library with a security flaw but the "good people" have not yet found the flaw, container image scanners will not find the flaw since the flaw will not be listed in the vulnerability databases that they utilize [12]. Nevertheless, it's crucial to remember that container image scanning is just one stage in the process of container security and is not a whole vulnerability management solution. The main categories of container vulnerabilities are as follows to manage them for container performance and security.

(i) Malicious Container Images

(ii) Detecting Image Vulnerabilities with Container Scanning

(iii) Application Vulnerabilities

2. Related Work

The use of virtualization technology has expanded significantly in recent years. This technology demonstrates a greater need for efficient and stable technologies for container virtualization. The industry's two necessary forms of virtualization technology are container-based virtualization and hypervisor-based virtualization. Container-based virtualization of these two classes cannot provide a lighter and more efficient virtual world without security concerns. Another study developed a DIVA system that automatically finds installations and parses public and group images in the Docker Hub. Using this system, they analyzed 356,218 images and found: (1) all formal, and group images contain more than 180 errors on average, taking into account all models, (2) many images have not been updated for hundreds of days [13]. This result shows that more automated, standardized approaches are required to implement security updates to Docker images. The existing Docker image validation system is the basis for this automatic security update for images [14]. Docker provides several features that are useful for developers and managers. It is an open platform that can be used as a compact, lightweight runtime and packaging engine called the Docker Engine for designing, deploying, and running applications. By replacing standard VMs with Docker modules, costs can be minimized. The risks of restoring the cloud development infrastructure are significantly reduced.

A study [15] explored the vulnerability-oriented of the Docker Environment and the safety consequences of using containers on traditional applications. Docker can help in all solutions for container security. Instead of being a package solution, it is a complete manufacturing and delivery network. It contributes several times to conducting a detailed survey on the relevant tasks in the field, organizing them into categories guided by protection, and then evaluating the security environment in containers [16]. Specifically, the vulnerability found in various components of the Docker environment, whether it be developing or implementing any original use cases using a top-down method. We also identify real-life situations in which specific vulnerabilities can be abused, recommend potential solutions, and address Docker-provider PaaS implementation issues [17]. However, the study described the security issues of containers and the problems associated with containers being lightweight and using the same kernel as the Host operating system. The study presents four cases and solutions obtained, considering all the security requirements for the hosted container.

It includes. (I) securing the container from the applications inside it, (II) inter-container protection, (III) protecting the host from the containers, and (IV) securing the containers from a malicious or semi-honest host. First, three use cases are handled by software, and the fourth is handled by hardware [18].

The use of virtualization technology is increasing day by day because of versatile services in containerization. There are two kinds of virtualization first one is container-based, and the second one is hypervisor-based. Container-based virtualization is faster and more lightweight in both virtualizations, but there are some security concerns. For container-based virtualization, virtualization is done by using Docker. An open-source platform. Hence, the analysis has been done for the internal security of Docker and the integration of Docker with the security feature of the Linux kernel, such as SELinux and AppArmor. It has been described that the security of Docker could be increased if it runs as non-privileged and enabled an additional layer of security by using AppArmor or SELinux [19]. A study described Docker and its performance analysis and took the stance that Docker has a protected layer on the container. Docker used a tool known as "Docker Engine" to execute the applications. It provided a Docker hub for sharing applications it worked the same as virtual machines. The lightweight approach reduced the computational cost and gave a better performance than virtual machines. It also described the advantages of Docker containers over virtual machines like speed, portability, scalability, and rapid delivery [20].

Container technologies are becoming popular and are used in many cloud applications. Recent research has shown various methods for detecting vulnerabilities in containers. Two types of detection schemes have been defined, one is static, and the other is dynamic. Twenty-eight vulnerabilities were tested; out of 28, only 03 vulnerabilities were discovered according to the static scheme, and 22-according to the dynamic scheme. It has been observed that dynamic detection is much faster than static detection [21]. A light rendering technique called containerization is currently popular. It is tightly integrated with the host operating system, which causes many security concerns. Some current work has been done in security, which deals with the relationship between host and container. The container is now automated and uses a third-party element that works across platforms and causes a vulnerability. In addition, the study showed Docker for three main reasons. (1) The popularity of containers in the market and the DevOps ecosystem. The survey found that 92 percent of people plan to use Docker to find a solution in container technology. (2) There is a security issue for container technology, and Docker is much safer. (3) Docker is already running in a different environment [22].

A study described different methods to solve high latency, network bottleneck, and network congestion. It could be achieved by moving from centralized to decentralized paradigm and edge computing to reduce application response time. Edge computing has enabled Docker, a platform of container-based technology with more advantages over VM-based Edge computing. Meanwhile, a study that evaluates the fundamental requirement [23] for EC, i.e., (1) deployment and termination, mainly describes the platform that provides an easy way to manage, install and configure services to deploy the low-end devices. (2) Resource and Service Management allows users to use the services even when the resources are out of the limit. (3) Fault Tolerance, which relies on high availability and reliability to the user. (4) Caching allows the user to experience better performance where Docker images can cache at the Edge. One such enables the Docker concept applied on Hadoop Streaming, which reduces the setup time and configuration errors. Overall, there are areas of improvement yet, providing elasticity and good performance [24].

Data centres' (DCs) efficiency has a room for improvement, and Docker presents that possibility. However, the resource allocation methods cannot be utilized effectively for Docker container-based applications. To reduce the cost of application deployment in DCs and to facilitate automated scaling when the workload of cloud apps fluctuates, we develop a unique application-oriented Docker container (AODC)-based resource allocation system. Then, taking into account Docker features, the needs of different applications, and the resources currently available in cloud data centers, we model the AODC resource allocation problem and propose a scalable algorithm for DCs with a variety of dynamic applications and substantial physical resources [25].

Numerous vulnerabilities brought on by the Internet's quick expansion have put a lot of strain on network security. Network security experts are, however, hard to come by. One of the key causes is that vulnerability testing is difficult for novices to master. As a result, a graphical operation platform is created and put into use. The virtual environment is created using Docker, while the graphical user interface is conceived and built using the Web system. The Metasploit framework is encapsulated to implement the attack technique. It can manage virtual environments, handle port scanning data, choose an attack mode, and report the results of an attack, among other things. With the help of this platform, novice programmers may learn less code and get a general grasp of network security vulnerability testing [26].

Due to the flexibility, portability, and scalability of containers, Docker is well-liked among software developers. However, as the security of the pictures that act as the foundational elements of apps has risen in importance, worries regarding vulnerabilities have increased. Validating the security of images that are retrieved from diverse sources is crucial as more development activities move to the cloud. We outline a continuous integration and continuous deployment (CI/CD) system in this article that verifies the security of Docker images throughout the software development life cycle. We provide photos with vulnerabilities and evaluate how well our method for finding the flaws works. Additionally, we demonstrate how dynamic analysis complements the static studies generally used for security assessments by evaluating the security of Docker containers based on their activity [27].

3. Proposed Docker-Security Design

This section discusses some important characteristics of today's leading container-based technologies that may affect performance. Docker is the leader in all container solutions. It is not an end-to-end solution but a complete production and delivery network. However, containers communicate directly with the host core, and attackers can place containers on the host system faster than virtual machines. This poses serious security concerns, while current hardening systems aim to enforce universal access control rules that only make it difficult to assess the required procedure. Thus, these mechanisms involve an explicit understanding of the purpose and actions of the container and entail manual interaction and customization [28]. Docker-sec is the most common Linux container implementation for Docker-based security solutions, but it can be easily extended to any standard OCI container. In short, Docker uses the Docker web and the Docker host client service architecture, as shown in Figure 2. The Docker engine uses a small and lightweight daemon to handle concurrent requests from multiple containers in its lifecycle. The consumer is the Docker user line, and through the Docker Engine, it connects to the server, the daemon that creates and distributes the boxes in the main framework. In turn, containers depend on a method named RunC, CLI, to control the container's operations at a low level. The shim container typically runs RunC as the mechanism used to create headerless containers [29].

In order to push the image to the docker repository docker cloud. Docker security scanning is passed through the Scanning API that will trigger the scan process and explore the vulnerabilities from main CVE databases. For vulnerability scanning, a scan is triggered. The scanner is connected with a plugin that sends data or iso images to CVE Scanning Validation Service after CVE system identification notifications are pushed to clients and isolation components for further classification in Processes, Network, and File System. User/Client push their customized images to Docker repository or cloud repository. The CVE is a list of information security vulnerabilities and exposures. It is the list of all types of vulnerabilities and its subtypes. CVE's mission is to facilitate data sharing across various vulnerability capabilities (tools, repositories, and services) via this common enumeration [30].

In order to execute this exploit, an attacker needs local access to this device, or it may be executed without any admin access. If you find this vulnerability critical and others may consider it low vulnerable. So there should be some standardized framework that may be needed to calculate the numerical value of the vulnerability. Based on the characteristics, its number can be high. Based on the common characteristics, Framework Common Vulnerability Scoring System (CVSS) [31].

Linux namespaces are the main framework for Docker sandboxing that virtualize; it separates different device components from each other. However, namespaces are typically connected to host device resources that cannot be virtualized to provide the necessary features. Even if the mount namespace provides various hierarchical file System views for the container, different critical file systems are typically shared with the host (for instance, cgroups, and sysfs). A container can access private information and configurations through them. It has been defined that AppArmor can reach, identify critical containers, and secure them. The mechanisms by which containers are distributed these services also need to be secured to allow valid access only to containers [32].

Docker-sec provides an extra protection layer in addition to safety by building AppArmor container profiles automatically. The device has secured against malicious or damaged containers seeking to take care or control the host, as it does not connect with containers via a signal, ptrace, or D-Bus to other processes. Docker-sec also increases the protection of containers by creating complex security profiles, provided an application is running. In this way, container rights (e.g., capacity, network connectivity, etc.) are restricted to the least required for the particular capacity. Docker-sec users will benefit from a MAC device without the complexity of its upkeep.

Docker-Sec creates stable AppArmor schemas for all Docker modules that need to be protected to make things more secure. Second, Docker-sec installs and applies the AppArmor profile to containers, which act as the attacker's entry point, since users of the virtualized applications execute arbitrary code and are entitled to it. Second, Docker-sec creates RunC security profiles for AppArmor because it is the only process that can directly interact with containers using signaling commands [33]. The proposed work aims to develop a different configuration profile for each container that can be minimized in a certain sense of security.

3.1. Creating Container Profile.

3.1. Creating Container Profile. Container profiles are developed with rules extracted from the configuration of each container and modified with appropriate rules. Docker-sec utilizes two strategies for this function:

(a) a static analysis that generates the original Docker profiles

(b) dynamic testing improves the Docker's profiles utilizing the controlled user-defined testing period.

The mechanism for static analysis gathers valuable static information and accesses the container. This data is either supplied by the user as evidence, or Docker generates this information and extracts the initial security rules by Docker directives to construct the profiles required for starting a new container. When a container image is examined, a report containing the following information is generated:

(i) The name, version, package manager, and description of the fundamental component

(ii) There are known vulnerabilities with the component.

The process of building image and checking the container's health during development, as shown in Figure 3. In particular, Docker-sec collected valuable details such as container volumes from the command line argument when a user executed a Docker created or a Docker executed commands for which the Docker Engine creates requested

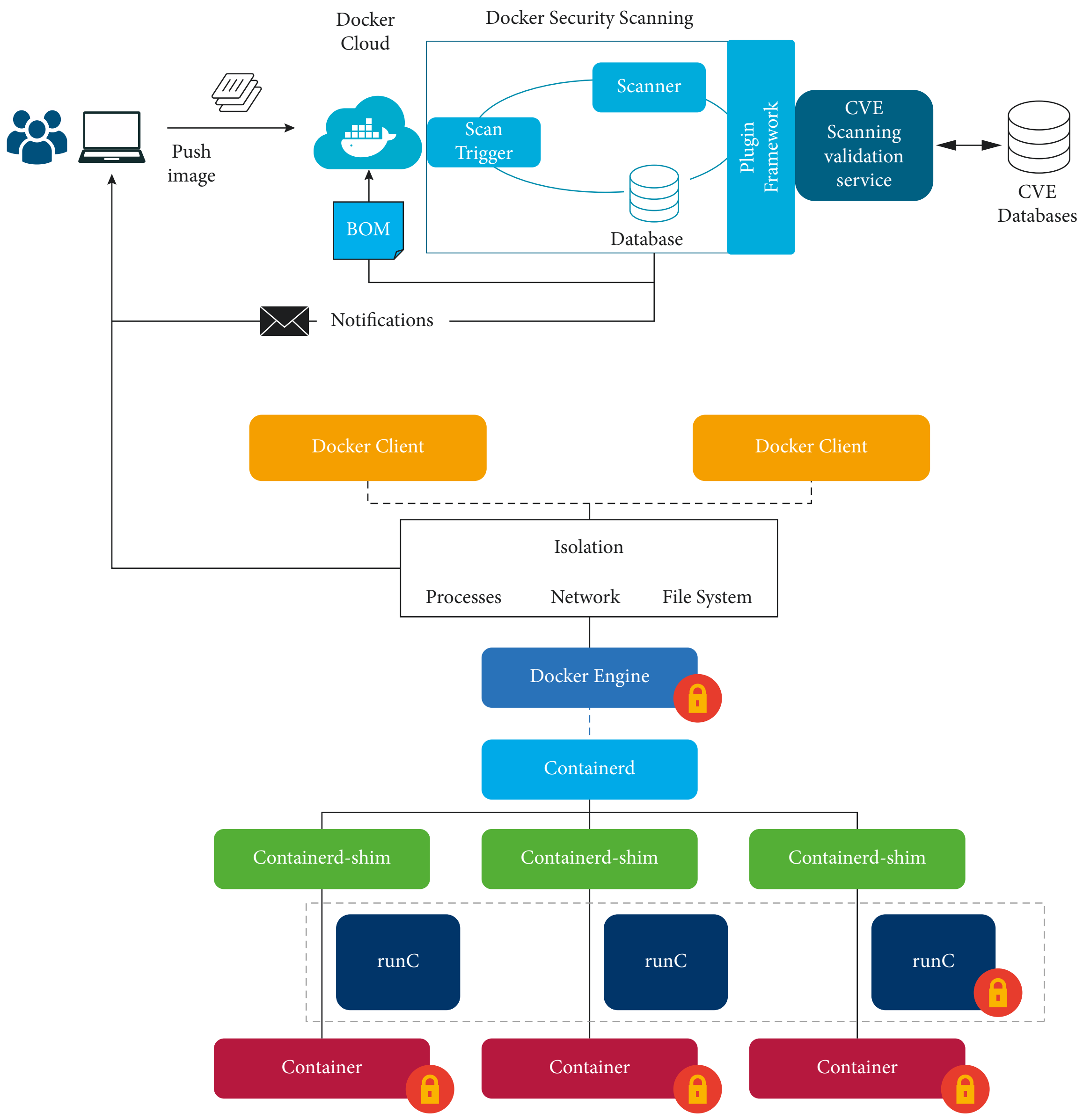


Figure 2: Proposed system architecture.

containers, i.e., the container-mapped files and directories of the file system host, as well as the user, root, or non-root container, and the related preferences [34]. Also, the Docker-Sec system provides information, such as the container ID, and SHA 256 checksum, and the container root-file systems mount point, from the Docker-info, which displays system-wide information concerning the Docker installation. When Docker-Sec learns these facts, it implements RunC on a brief AppArmor profile to initialize the individual container. Only then can RunC transformations to the AppArmor profile (and likely increased by the Dynamic Monitoring mechanism) be used in the container runtime before handing over the power to the container process [35].

Dynamic monitoring helps cloud users set a clear container training cycle during which Docker-sec collects data regarding the actions of containers. After beginning training, the user will take advantage of the application component involved in using all the application features needed to decide the necessary rights for the operation of the container (e.g., file system access, network access, and capabilities). Docker-Sec produced an audit log registration that tracks legitimate container access at the end of the training cycle and uses the applicable rules to extract disproportionate privileges originated by the static analysis profile from the container runtime profile. Training will be repeated if required before the necessary functions in the container profile have been recorded and printed. It is critical that only enabled and confident consumers have an entry point to the container system during container runtime profile preparation. It should be recalled that the other containers continue to be guarded through the container is in drill mode.

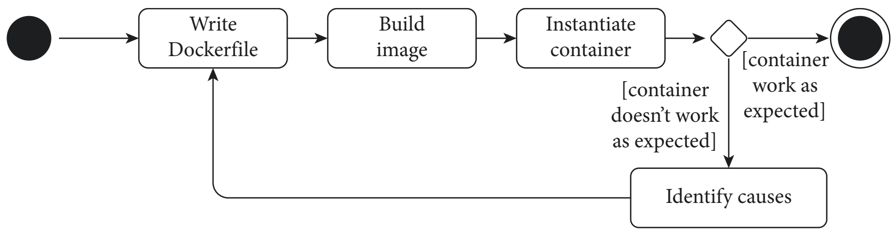

Write Dockerfile
Build image
Instantiate container
[container work as expected]
[container doesn't work as expected]
Identify causes

Figure 3: workflow when developing a Dockerfile.

Docker-sec uses AppArmor capability to inspect those accesses required by a mechanism to provide the above features, as shown in Figure 3. In either enforcement mode, which enforces all the profile systems and does not enable infringements, a Profile may be specified by AppArmor or a complaint mode that records infringements but enables system calls to succeed. Furthermore, these two modes can be combined for more versatility by reasonable rules. To track record the collection of accesses controlled by the rule, in specific, by keeping a profile in administer mode, the proposed system will continue to enforce the resulting profile rules protecting the system. After training, docker-sec can control and access the container to particular services using this capability [23].

3.2. Run-C Profile. The host's valid processes only permit connection to containers using signals or ptraces, and containers cannot, most notably, access or handle processes of the host utilizing these protocols, reduce the attack surface and defend them against several assaults [11]. With RunC working via commands like Docker execution, Docker exec, or Docker stats directly with container processes, the proposed system is agreed to have a different AppArmor profile [12]. The RunC profile includes rules that allow RunC to set the mount point of each container by calling the device pivot root and assigning it a different time profile. This profile has the corresponding rules. This temporary profile, used while the specific container has initialized, shields the container from the end profile of the container used for container service, as mentioned above before it passes (by aa_change_onexec, aa_change profile, or functions) to the run-c profile. Docker-sec covered the entire container lifecycle from RunC, the temporary profile, and configuration of containers, and finished with the concluding container profile used at runtime.

3.3. Profile of Docker Daemon. Docker-sector uses a modified version of the Docker GitHub repository of AppArmor, limiting admittance to the tools and services that the Docker engine needs for service to secure the Docker daemon.

3.4. Process Isolation. Docker ensures that operations are containerized in namespaces, that permissions are restricted, and that applications are accessed in other containers and on the appropriate host. Process isolation primarily aims to prevent corrupted containers from using process control interfaces that interfere with other containers. PID (process identification) namespaces isolate the container method ID from the host ID for this mechanism. PID namespaces are hierarchical, so only other processes can be used in their namespace or in their "child" namespace class. Therefore, the host will manage and manipulate processes in the current folder of the PID namespace; when a new namespace is created and delegated to a container, the container processes will either not know or do nothing with other processes in the host or container that are running independently. They are more difficult to attack if the attacker is not watching such processes. PID names often force each container to have a PID of 1, which allows all processes in the namespace to terminate. This helps the administrator close the container when suspicious activity is detected.

3.5. Filesystem Isolation. The file structures of the host and containers must be secured from unauthorized access and manipulation to receive filesystem isolation. To separate the filesystem hierarchy shared with multiple containers, Docker uses mount namespaces, which are often called filesystem namespaces. Mount namespaces offer a separate view of the filesystem tree in the processes of each container and limit mounting events that only influence the container boundary. Any kernel file systems are not namespace; however, i.e., /sys, /proc/sys, /proc/SysRq – cause, /proc/irq, and /proc/bus, and they need to be installed on a Docker container. This poses the dilemma that a container can directly access these file systems from the server. Docker restricts threats an infected container might make to the host using the two file system security mechanisms using these filesystems: (1) the removal from containers of written authorization from specific file systems and (2) the removal of a container method from a file system within the container. The second mechanism is to delete capability from containers from the CAP_SYS_ADMIN framework.

3.6. Network Isolation. To isolate the network, you must avoid network attacks such as Man-in-the-Middle attacks and ARP network spoofing. Containers should be designed so that neither network traffic nor the server can eavesdrop. Docker creates a standalone networking stack for each container using network namespaces. Therefore, they use their IP addresses, IP routing tables, and network equipment in network isolation. It prompts containers, similar to their contact with external hosts, to communicate with each other through the appropriate network interfaces. By default, the virtual Ethernet Bridge connects between the containers and the host computer, as shown in Figure 4. This approach allows Docker to build a virtual Ethernet bridge on a host device known as docker0 that automatically exchanges packets over its network interfaces. Docker will also create a new named VIE-Ethernet interface that links this structure to the bridge when a new container is created. The port is also linked to the container's eth0 interface for forwarding packets to the connection. The regular Docker networking model is vulnerable to ARP spoofing and Mac flooding, as all incoming unfiltered packets are sent over the network.

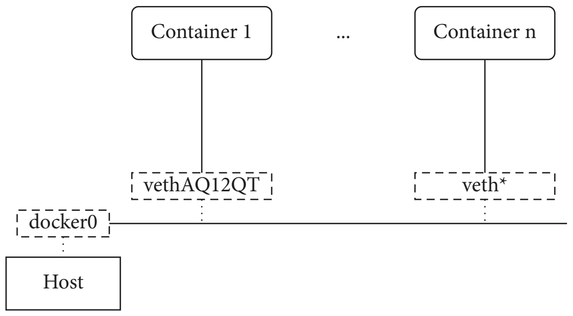


Figure 4: Docker's networking model.

3.7. Docker Vulnerability Scanning. A central component of the Docker protection plan should be Docker image security scanning. Although image scanning cannot shield you from all potential malware vulnerabilities, it is the primary means of protecting your images against security flaws or insecure code. Docker's local images are susceptible to scanning on the Snyk engine, enabling the users to obtain insight through their local Docker and local images' protection positions. Users trigger (command-line interface) CLI vulnerability scans and use the CLI to view scan results. The scan feature includes a list of Typical Vulnerabilities and Exposures and proposes CVE remedies. Figure 5 shows the image scanning through Clair. During the scan results, 37 vulnerabilities have been detected, CVE severity type is low, and buffer over-read is detected.

In Figure 6, the scanning is performed over different CVE databases and found 13 vulnerabilities, most of which are payload, signedness, and overflow type.

4. Results

The security service measured the Docker-sec overhead performance of starting the container and executing the nested program by adding an AppArmor profile in the container. The proposed assessment is carried out in two directions. First, it performs various workloads using a primary operating system testing tool to load your computer system and use an Ubuntu mirror taken from the approved Docker Center repository. There are two types of Docker containers: secure Docker-sec (called enabled) and others without a security profile (known as "disabled"). Selected workloads like computer primaries, file transfer rate, random read/write access, and different file operations like read, write, and delete. In the different stress workload over network assessment like socket, keep different network settings like some ports open and closed through different evaluations, as shown in Figure 7. Performance time has been calculated by booting the container up and calculated using different Docker images with different settings. Their boot-up time dictated their preference for specific case scenarios: For the worst-case scenario, choose images with fast initialization times to test for the highest relative overhead our engines can deliver. In the case scenario, stress performance is low in ubuntu and overhead is 2.8% compared to other images, as shown in Figure 8.

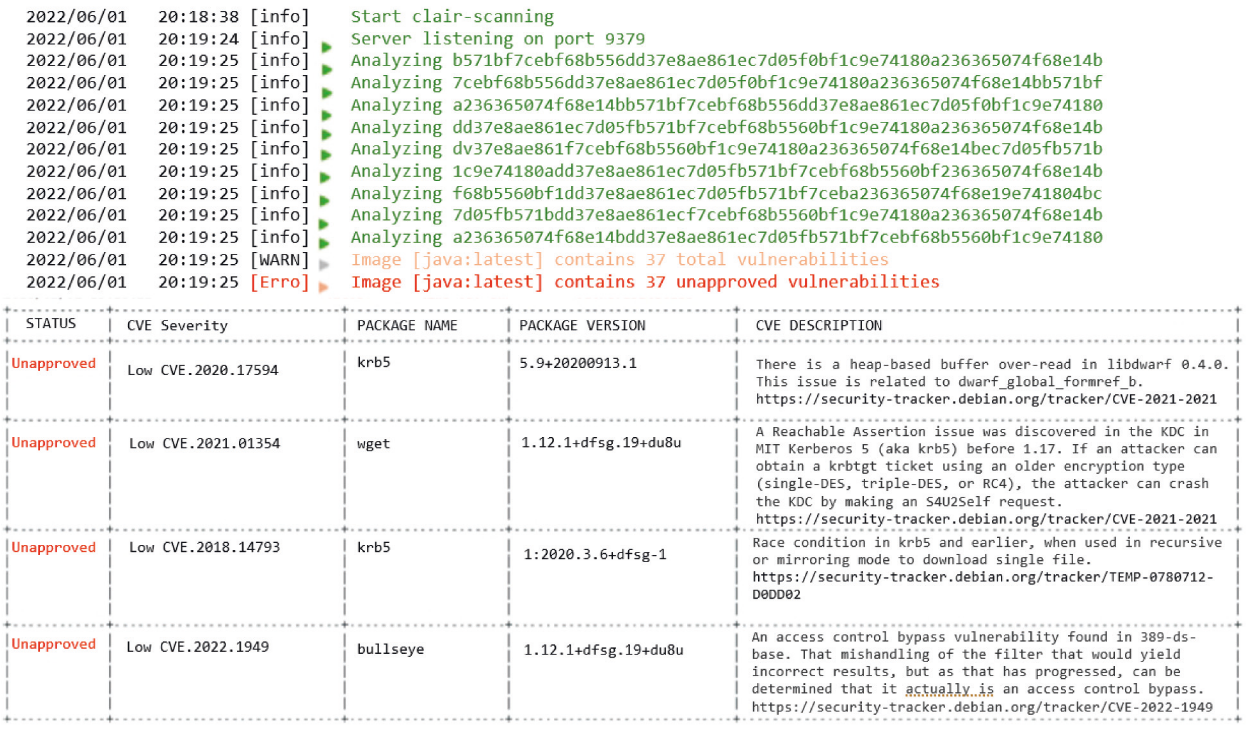
With the same network setting, performance is calculated over different images and monitored the stress load is 3.8% whereas the overall is 3.45%.

Our overview shows that a Docker-sec application imposes low overhead on both the lifetime and bootstrap time of the container, as shown in Figures 7 and 8. The overhead observed in CPU-bound applications has little effect, while performance tests have marginally increased overhead for the capital investment of file system deformation that does not exceed 2%. In the latter case, the observed overhead does not exceed 3.45%. Interestingly, the highest overhead is computed for the socket test. This is because it takes longer to enforce AppArmor laws when creating sockets than in all other cases. Finally, Docker-sec imposes a relatively constant overhead (from 3 to 4%) regardless of image types, as we calculate container load delays for various Dockerfiles.

5. Discussion

Docker-sec uses CMD like Docker GUI that adds the -sec suffix to the new Docker commands. This proposed programmed framework has built on AppArmor and a bash-wrapping efficacy, which helps build AppArmor profiles customized to particular container instances and connect to execute them. The key aspects of interaction include creating containers, creating an AppArmor profile for an image container, well-known exploits, and training of random images of containers of various workloads. Two situations are covered in our demonstration section. The first scenario would allow participants to check the efficiency of Docker-sec through the design and utilization of an improved security profile suited to a particular container case. In the second scenario, the participant's images created a new safety profile using a container with an arbitrary performance workload.

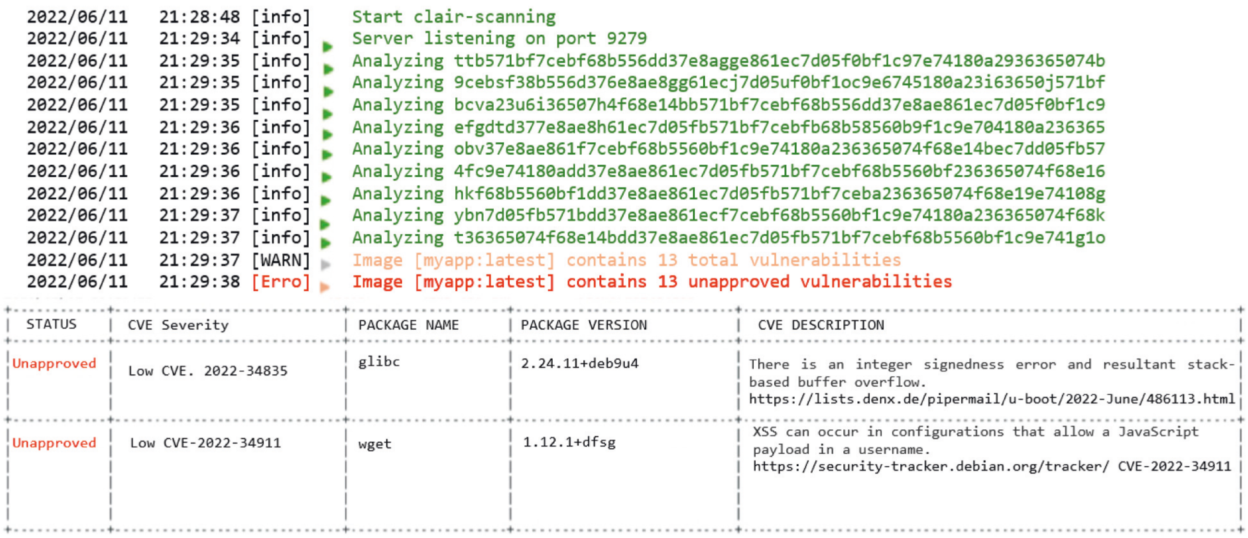
After accessing the kernel of a container, users can "act maliciously" by performing several simulated attacks such as modifying the Secure Sockets Host (SSH) daemon, adding new services to the container, and manipulating a container engine vulnerability (such as CVE-2022-1949). In the first scenario, a new WordPress container can be launched using the Docker-sec CLI, which is launched using a profile generated by a static testing device. After configuring the



```
2022/06/01   20:18:38 [info]   Start clair-scanning
2022/06/01   20:19:24 [info]   Server listening on port 9379
2022/06/01   20:19:25 [info]   Analyzing b571bf7cebf68b556dd37e8ae861ec7d05f0bf1c9e74180a236365074f68e14b
2022/06/01   20:19:25 [info]   Analyzing 7cebf68b556dd37e8ae861ec7d05f0bf1c9e74180a236365074f68e14bb571bf
2022/06/01   20:19:25 [info]   Analyzing a236365074f68e14bb571bf7cebf68b556dd37e8ae861ec7d05f0bf1c9e74180
2022/06/01   20:19:25 [info]   Analyzing dd37e8ae861ec7d05fb571bf7cebf68b5560bf1c9e74180a236365074f68e14b
2022/06/01   20:19:25 [info]   Analyzing dv37e8ae861f7cebf68b5560bf1c9e74180a236365074f68e14bec7d05fb571b
2022/06/01   20:19:25 [info]   Analyzing 1c9e74180add37e8ae861ec7d05fb571bf7cebf68b5560bf236365074f68e14b
2022/06/01   20:19:25 [info]   Analyzing f68b5560bf1dd37e8ae861ec7d05fb571bf7ceba236365074f68e19e741804bc
2022/06/01   20:19:25 [info]   Analyzing 7d05fb571bdd37e8ae861ecf7cebf68b5560bf1c9e74180a236365074f68e14b
2022/06/01   20:19:25 [info]   Analyzing a236365074f68e14bdd37e8ae861ec7d05fb571bf7cebf68b5560bf1c9e74180
2022/06/01   20:19:25 [WARN]   Image [java:latest] contains 37 total vulnerabilities
2022/06/01   20:19:25 [Erro]   Image [java:latest] contains 37 unapproved vulnerabilities
```

STATUS	CVE Severity	PACKAGE NAME	PACKAGE VERSION	CVE DESCRIPTION
Unapproved	Low CVE.2020.17594	krb5	5.9+20200913.1	There is a heap-based buffer over-read in libdwarf 0.4.0. This issue is related to dwarf_global_formref_b. https://security-tracker.debian.org/tracker/CVE-2021-2021
Unapproved	Low CVE.2021.01354	wget	1.12.1+dfsg.19+du8u	A Reachable Assertion issue was discovered in the KDC in MIT Kerberos 5 (aka krb5) before 1.17. If an attacker can obtain a krbtgt ticket using an older encryption type (single-DES, triple-DES, or RC4), the attacker can crash the KDC by making an S4U2Self request. https://security-tracker.debian.org/tracker/CVE-2021-2021
Unapproved	Low CVE.2018.14793	krb5	1:2020.3.6+dfsg-1	Race condition in krb5 and earlier, when used in recursive or mirroring mode to download single file. https://security-tracker.debian.org/tracker/TEMP-0780712-D0DD02
Unapproved	Low CVE.2022.1949	bullseye	1.12.1+dfsg.19+du8u	An access control bypass vulnerability found in 389-ds-base. That mishandling of the filter that would yield incorrect results, but as that has progressed, can be determined that it actually is an access control bypass. https://security-tracker.debian.org/tracker/CVE-2022-1949

Figure 5: Docker image scanning with file system isolation.



```
2022/06/11   21:28:48 [info]   Start clair-scanning
2022/06/11   21:29:34 [info]   Server listening on port 9279
2022/06/11   21:29:35 [info]   Analyzing ttb571bf7cebf68b556dd37e8agge861ec7d05f0bf1c97e74180a2936365074b
2022/06/11   21:29:35 [info]   Analyzing 9cebsf38b556d376e8ae8gg61ecj7d05uf0bf1oc9e6745180a23i63650j571bf
2022/06/11   21:29:35 [info]   Analyzing bcva23u6i36507h4f68e14bb571bf7cebf68b556dd37e8ae861ec7d05f0bf1c9
2022/06/11   21:29:36 [info]   Analyzing efgdtd377e8ae8h61ec7d05fb571bf7cebfb68b58560b9f1c9e704180a236365
2022/06/11   21:29:36 [info]   Analyzing obv37e8ae861f7cebf68b5560bf1c9e74180a236365074f68e14bec7dd05fb57
2022/06/11   21:29:36 [info]   Analyzing 4fc9e74180add37e8ae861ec7d05fb571bf7cebf68b5560bf236365074f68e16
2022/06/11   21:29:36 [info]   Analyzing hkf68b5560bf1dd37e8ae861ec7d05fb571bf7ceba236365074f68e19e74108g
2022/06/11   21:29:37 [info]   Analyzing ybn7d05fb571bdd37e8ae861ecf7cebf68b5560bf1c9e74180a236365074f68k
2022/06/11   21:29:37 [info]   Analyzing t36365074f68e14bdd37e8ae861ec7d05fb571bf7cebf68b5560bf1c9e741g1o
2022/06/11   21:29:37 [WARN]   Image [myapp:latest] contains 13 total vulnerabilities
2022/06/11   21:29:38 [Erro]   Image [myapp:latest] contains 13 unapproved vulnerabilities
```

STATUS	CVE Severity	PACKAGE NAME	PACKAGE VERSION	CVE DESCRIPTION
Unapproved	Low CVE. 2022-34835	glibc	2.24.11+deb9u4	There is an integer signedness error and resultant stack-based buffer overflow. https://lists.denx.de/pipermail/u-boot/2022-June/486113.html
Unapproved	Low CVE-2022-34911	wget	1.12.1+dfsg	XSS can occur in configurations that allow a JavaScript payload in a username. https://security-tracker.debian.org/tracker/ CVE-2022-34911

Figure 6: Docker image with network isolation scanning.

container, the testing cycle and container used by the WordPress user interface will be defined. During this process, the dynamic control engine checks the resources of the individual devices and manages the residuals of the machine, while the static profile remains active, which can be shown to change the profile. After training, static with a dynamic profile determines the exact rights each application needs during the testing phase.

In another scenario, the participants will run Docker-sec for different images from Docker and capabilities in the enclosed container. All images have been associated with the generated profiles for containers with similar images and varying different workloads. After the execution of the process, the individual container rights can be defined and how Docker-sec familiarizes them. Different benchmarks include high application loads or computer stress loads, such as CPU and I/O. The flexibility of requirements and user requirement observations regarding overall costs levied on different applications by Docker-sec and AppArmor tested their performance in real-life and serious situations.

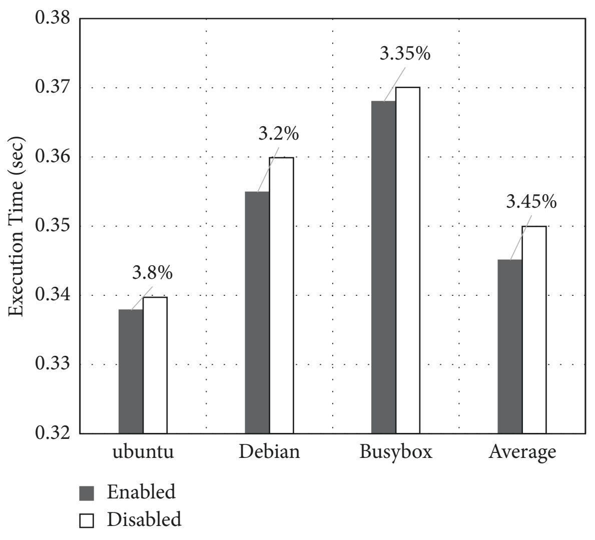


Figure 7: Docker-Sec's performance overhead.

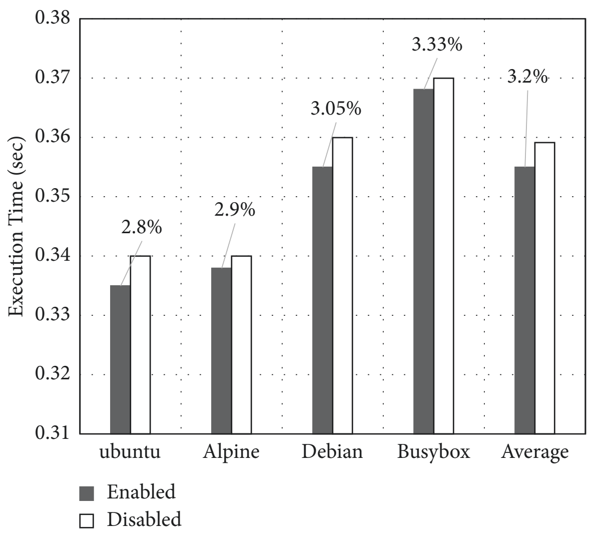


Figure 8: Docker-Sec's performance overhead with different images.

6. Conclusion

Container virtualization can provide higher virtual worlds and more critical functionality than hypervisor virtualization. This technology is considered better than the former virtual machine technology. Docker's performance and vulnerability analyzed the security of Docker, which is one of the most common container-based technologies, and how to protect against unauthorized access. Vulnerability management is more crucial now than it has ever been because of containers and cloud-native technology. Potential tasks are equating Docker container security with other containers or virtual machines. Docker containers are reasonably more stable and can perform better by modified default configuration. In contrast, benchmarks have a marginally high overhead for a strain file system no longer over 1%.

Data Availability

The data collected during the data collection phase are available from the corresponding authors upon request.

Conflicts of Interest

No potential conflicts of interest have been declared by the authors.

Acknowledgments

This work was supported by the Qatar National Library, Doha, Qatar, and in part by the QU Internal Grant, Qatar University Internal, under Grant IRCC-2021-010.

References

[1] 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.

[2] 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.

[3] A. Martin, S. Raponi, T. Combe, and R. Di Pietro, "Docker ecosystem–vulnerability analysis," *Computer Communications*, vol. 122, pp. 30–43, 2018.

[4] 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.

[5] N. Tabassum, T. Alyas, M. Hamid, M. Saleem, and S. Malik, "Hyper-convergence storage framework for ecocloud correlates," *Computers, Materials & Continua*, vol. 70, no. 1, pp. 1573–1584, 2022.

[6] 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.

[7] S. Kwon and J. H. Lee, "DIVDS: docker image vulnerability diagnostic system," *IEEE Access*, vol. 8, pp. 42666–42673, 2020.

[8] M. Rovnyagin, A. Hrapov, A. Guminskaia, and A. Orlov, "ML-based heterogeneous container orchestration architecture," in *Proceedings of the IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering*, pp. 477–481, Moscow, Russia, January 2020.

[9] P. Prajapati and P. Shah, "A review on secure data deduplication: cloud storage security issue," *Journal of King Saud University–Computer and Information Sciences*, vol. 20, pp. 1–12, 2020.

[10] B. Ismail, E. Goortani, M. Karim, M. Tat, S. Setapa, and J. Luke, "Evaluation of docker as edge computing platform," *IEEE Conference on Open Systems (ICOS)*, vol. 50, pp. 190–101, 2015.

[11] T. Alyas, N. Tabassum, M. Waseem Iqbal, A. S Alshahrani, A. Alghamdi, and S. Khuram Shahzad, "Resource based automatic calibration system (rbacs) using kubernetes framework," *Intelligent Automation & Soft Computing*, vol. 35, no. 1, pp. 1165–1179, 2023.

[12] Y. Jiang, W. Liu, X. Shi, and W. Qiang, "Optimizing the copy-on-write mechanism of docker by dynamic prefetching," *Tsinghua Science and Technology*, vol. 26, no. 3, pp. 266–274, 2021.

[13] N. Tabassum, T. Alyas, M. Hamid, M. Saleem, S. Malik, and S. Binish Zahra, "Qos based cloud security evaluation using neuro fuzzy model," *Computers, Materials & Continua*, vol. 70, no. 1, pp. 1127–1140, 2022.

[14] X. Cai, S. Geng, D. Wu, J. Cai, and J. Chen, "A multi-cloud model-based many-objective intelligent algorithm for efficient task scheduling in internet of things," *IEEE Internet of Things Journal*, vol. 8, no. 12, pp. 9645–9653, 2021.

[15] M. Mohamed, R. Engel, A. Warke, S. Berman, and H. Ludwig, "Extensible persistence as a service for containers," *Future Generation Computer Systems*, vol. 97, pp. 10–20, 2019.

[16] M. De Benedictis and A. Lioy, "Integrity verification of docker containers for a lightweight cloud environment," *Future Generation Computer Systems*, vol. 97, pp. 236–246, 2019.

[17] F. D'Urso, C. Santoro, and F. F. Santoro, "Wale: a solution to share libraries in Docker containers," *Future Generation Computer Systems*, vol. 100, pp. 513–522, 2019.

[18] S. Liu, Z. Xu, and Z. Tian, "Implementation of NRF in the Docker-based NFV platform," *Journal of Engineering*, no. 23, pp. 8884–8887, 2019.

[19] V. Giménez-Alventosa, G. Moltó, and M. Caballer, "A framework and a performance assessment for serverless mapreduce on AWS lambda," *Future Generation Computer Systems*, vol. 97, pp. 259–274, 2019.

[20] S. Sultan, I. Ahmad, and T. Dimitriou, "Container security: issues, challenges, and the road ahead," *IEEE Access*, vol. 7, pp. 52976–52996, 2019.

[21] 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.

[22] A. Pérez, G. Moltó, M. Caballer, and A. Calatrava, "Serverless computing for container-based architectures," *Future Generation Computer Systems*, vol. 82, 2018.

[23] R. Morabito, "Virtualization on internet of things edge devices with container technologies: a performance evaluation," *IEEE Access*, vol. 5, pp. 8835–8850, 2017.

[24] N. Tabassum, M. Khan, S. Abbas, T. Alyas, A. Athar, and A. Khan, "Intelligent reliability management in hyper- convergence cloud infrastructure using fuzzy inference system," *EAI Endorsed Transactions*, vol. 2020, 12 pages, 2020.

[25] T. Bhuddtham and P. Watanapongse, "Time-related vulnerability lookahead extension to the CVE," in *Proceedings of the 13th International Joint Conference on Computer Science and Software Engineering, JCSSE*, Khon Kaen, Thailand, July 2016.

[26] T. Alyas, I. Javed, A. Namoun, A. Tufail, S. Alshmrany, and N. Tabassum, "Live migration of virtual machines using a mamdani fuzzy inference system," *Computers, Materials & Continua*, vol. 71, no. 2, pp. 3019–3033, 2022.

[27] D. Reis, B. Piedade, F. F. Correia, J. P. Dias, and A. Aguiar, "Developing docker and docker-compose specifications: a developers' survey," *IEEE Access*, vol. 10, pp. 2318–2329, 2022.

[28] N. Tabassum, A. Ditta, T. Alyas et al., "Prediction of cloud ranking in a hyperconverged cloud ecosystem using machine learning," *Computers, Materials & Continua*, vol. 67, no. 3, pp. 3129–3141, 2021.

[29] L. Heilig, E. Lalla-Ruiz, and S. Vob, "Modeling and solving cloud service purchasing in multi-cloud environments," *Expert Systems with Applications*, vol. 147, pp. 113165–113198, 2020.

[30] R. Ushakov, E. Doynikova, E. Novikova, and I. Kotenko, "CPE and CVE based technique for software security risk assessment," in *Proceedings of the 11th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications, IDAACS*, vol. 1, pp. 353–356, Cracow, Poland, January 2021.

[31] X. Guan, X. Wan, B. Y. Choi, S. Song, and J. Zhu, "Application oriented dynamic resource allocation for data centers using docker containers," *IEEE Communications Letters*, vol. 21, no. 3, pp. 504–507, 2017.

[32] M. Alaluna, E. Vial, N. Neves, and F. M. Ramos, "Secure multi-cloud network virtualization," *Computer Networks*, vol. 161, pp. 45–60, 2019.

[33] X. Gao, Z. Gu, M. Kayaalp, D. Pendarakis, and H. Wang, "Container-Leaks: emerging security threats of information leakages in containerclouds," *International Conference on Dependable Systems and Networks (DSN)*, vol. 47, pp. 237–248, 2017.

[34] O. Onadele, J. He, T. Dai, and X. Gu, "A study on container vulnerability exploit detection," *IEEE International Conference on Consumer Electronics*, vol. 21, pp. 159–166, 2019.

[35] T. Combe, A. Martin, and R. Di Pietro, "To docker or not to docker: a security perspective," *IEEE Cloud Computing*, vol. 3, no. 5, pp. 54–62, 2016.