University of Sindh Journal of Information and Communication Technology (USJICT)
Volume 8, Issue 1, 2024
ISSN-E: 2523-1235, ISSN-P: 2521-5582
Website: http://sujo.usindh.edu.pk/index.php/USJICT/

Strategies for Malware Defense in Containerized Environments

Sara Hameed[1], Usra Sami[2], Sara Naeem Aslam[3], Meerab Tahir[4]

[1,3,4] Software Engineering Department, Ned University of Engineering and Technology Karachi, Pakistan
[2] Computer Engineering Department, Bahria University Karachi, Pakistan
hameed4402051@cloud.neduet.edu.pk[1],usrasami.bukc@bahria.edu.pk[2],aslam4406044@cloud.neduet.edu.pk[3],
tahir4405501@cloud.neduet.edu.pk[4]

Abstract— With advancements in the domain of technology, different environments are used. With the increasing use of different deployment environments, there are rising problems of portability, compatibility, and scalability. Containers are an isolated, portable, and efficient way of running modern applications. Their usefulness as tools has also led to increased cases of insecurity. Conventional container protection methods included the Ubuntu, Clair, Grape, STRIDE and DREAD frameworks, which mainly relied on vulnerability assessment and threat modeling. However, these measures did not include changes in the evolving features of container attacks aimed at protecting image interfaces and APIs. This paper provides a detailed analysis of malware attacks and the restriction of privileges to non-root users after malicious identification. This is done to minimize the attack vectors, including the preference of non- root users and only using up-to-date lightweight base images and multiple build formations. This paper implements the Trivy tool to protect against malware attacks. Scanner Trivy is an open source vulnerability scanner that can identify threats and evaluate risks to avoid misconfigurations and vulnerabilities. This prevents probable attacks by providing a secure and reliable host environment. Therefore, it helps future generations mitigate the risks associated with working in containerized environments and overcome possible future threats.

Keywords: Container, Docker, Trivy, Vulnerabilities, Malware.

I. INTRODUCTION

With the advancement of technology and the Internet, issues such as compatibility, deployment, portability, and scalability are prevalent. As a result, containers came into place and overcome the scalability and compatibility issues by providing a simple, lightweight environment where all applications can be easily isolated and deployed [1]. However, security has become a concern in containerization and brings new threats, which need efficient protection solutions. Docker has traditionally been available with others, such as Ubuntu, Clair, and Grape, in the provision of a vulnerability scan and images. The STRIDE and DREAD models assisted in threat assessment in container environments. However, these methods primarily included main channels or general threats not much concerning the newer and more complex forms of container-based threats, like that of unprotected image interfaces as well as APIs [2]. This research study is focused on protecting against these types of attacks that violate the root user privileges of containers and making sure that the base program images are safe and compact [3].

To address this research gap, this study emphasizes various strategies, including operating between root and normal users and updating base images frequently with lighter container images. Also, multiple builds for containerization split the code, enhancing security and performance at the same time. Therefore, these are some of the effective strategies that afford a broader solution in defending against the advanced malware within the container's environments [4].

However, Containers are isolated, but they have the same OS kernel across all environments, which is the root cause of security threats. Once the attackers gain access to the OS kernel, they can gain access to the host systems by breaking isolated environments. Different malware attacks can be done on containers. Some most common examples are unauthorized access to the data in isolated environments and creation of destructive files, folders, or programs that provide access to data, and the installation of harmful programs.

However, most of these malware attacks are done on containers with more privileges and configured to run with root permission. The containers with root permission are configured for full admin rights and privileges.

Therefore, containers with root privileges are subjected to most malware attacks and can gain access to complete admin rights and can attack the entire host systems [7].

With the increasing number of malware attacks and concerns regarding containerization security, it is advised to

assign user permission to the containers to save them from potential threats and attacks. Furthermore, there should be a standard that even if the container is compromised in a security attack then after gaining access it should also have user permissions. As a result, comparatively lesser damage can be done to the system because the attackers won't have access to data and folders which prevents them from adding or deleting files in working directories or installing new software within a container or on a host OS [8].

Therefore, with the rising concerns, it is important to mitigate such risks of malware attacks. This research paper overcomes the risks of potential malware attacks by providing a solution to containerization security. These security threats can be overcome by the use of a Trivy tool which is an opensource tool specifically designed to detect vulnerabilities in the system.

Trivy tool is used to scan vulnerabilities, misconfigurations, and security risks. Furthermore, it scans container images of the container and is given root privileges. However, this tool is very beneficial when a root user is identified as malicious and then this tool can immediately convert it into a non-root user by restricting its privileges to normal users only. This prevents the system from any possible malicious attacks and installation of harmful software.

In this research study, a malicious attack has been done on the host systems with root privileges and then the Trivy tool to mitigate the possible risks and threats. As a result, when an attack is done Trivy tool scans the images of the container and restricts its privileges to non-root users only. This shift in converting users drastically thins down the possibilities of an attacker's action, even if they manage to compromise the container [9].

However, as a tool, Trivy can be used to scan for these vulnerabilities, as well as to be incorporated into the CI/CD pipeline for containers, before going live. This makes it possible for organizations to automate security measures to allow containers with non-root privileges. A simulated privilege escalation and Trojan-based malware attack is carried out on the container by enabling root access, which exposes the system to vulnerabilities. This allows unauthorized access to the container and facilitates the creation of multiple malicious directories within the container environment. This implementation shows the importance of the least privilege principle as it saves from potential malware attacks by restricting privileges. Therefore, special attention is drawn towards security measures in container environments, efficient practices to mitigate malware attacks, and a comparison between the root and non- root user privileges, causes, and effects. Moreover, it will help ensure the security of container environments and ensure that the people working with business environments are aware of such malware attacks and security concerns. Figure 1 shows the area of study followed in this research.

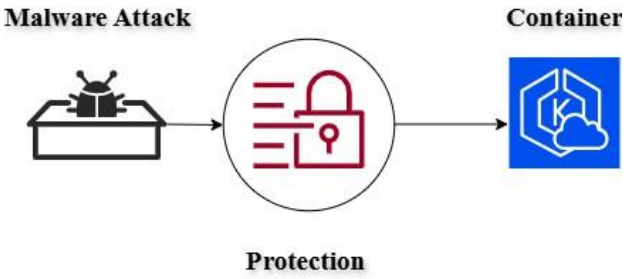

Figure 1. Secure Container Environments from malware attacks

II. Literature Review

This section reviews various malware attacks and their prevention strategies in container environments, derived from multiple research papers.

Lee et al. [2] conducted experiments to demonstrate security vulnerabilities in the Docker environment using Ubuntu 20.04.3 and Docker 20.10.12. During the experiment on ARP spoofing attack, a malicious container intercepted communication between two legitimate containers by mimicking an IP address. For the DDoS attack, 20 malicious Ubuntu containers launched a SYN flood against a phpMyAdmin container overwhelming it with traffic. The Wireshark tool was used to monitor the attack. Lastly, the privilege escalation scenario involved a malicious host uploading a tampered Docker image to a registry, which was then downloaded and executed on the target host. This provided the attacker unrestricted access through a reverse shell. The study highlights that poor isolation between containers, insufficient security protocols, and exposed Docker APIs are primary reasons for these security threats.

Haq et al. [7] provided an overview of various attack vectors threatening container security such as remote code execution, privilege escalation, and DoS attacks. According to the study, existing defense mechanisms such as static tools like Clair, Trivy, Snyk, and Grype showed detection rates below 50% which highlights their inability to discover new vulnerabilities. Dynamic anomaly detection techniques have gained attention as a solution to these constraints because of their ability to track container behavior in real time and identify deviations from normal patterns. Although dynamic defenses can detect zero-day vulnerabilities, their practical application is hindered by the complexity of attacks and the lack of comprehensive datasets. Additionally, current anomaly detection systems face challenges such as high false-positive rates and the need for well-labeled datasets for training. The authors recommend integrating static and dynamic security approaches to improve protection against evolving threats.

Although containers are effective and resource-efficient, Mullinix et al. [8] pointed out that they share the host operating system, which can cause security issues. One

major problem is the widespread use of outdated or malicious images, which may contain vulnerabilities that attackers might exploit. Despite various security protections from the Linux kernel, threats from compromised third-party images remain. The study emphasizes the necessity of using tools such as Docker Notary for content trust, static and dynamic analysis to un- cover vulnerabilities, and strong monitoring systems to protect against anomalies in container behavior. To reduce these risks, the authors suggested enhancing anomaly detection, modifying CVSS metrics for container vulnerabilities, and providing developers with security training.

Huang et al. [3] presented security vulnerabilities in Docker container technology, discussing the threats of the immaturity of auditing processes within the release of Docker images. The authors pointed out that, despite existing security mechanisms such as Linux namespaces for resource isolation and control groups (Cgroups) for resource management, vulnerabilities still exist. They discussed the threats of container escape attacks and even leakage of sensitive information, and how malicious containers can eventually compromise host systems. To improve this, the authors proposed a detection framework that utilizes scanning for known vulnerabilities and employing machine learning algorithms to predict unknown threats, thus proving the need for better auditing and monitoring practices to support security in containerized environments.

Chelladhurai et al. [1] emphasized the weaknesses of Docker containers, specifically their susceptibility to DoS at- tacks due to poor isolation when compared to virtual machines. Docker containers interact directly with the host kernel, which raises the possibility of host-targeted attacks if exploited. The authors identified weaknesses such as exposure to ARP spoofing, MAC flooding, and security risks from the Docker daemon's root privileges. They recommended strengthening its security using AppArmor, SELinux, and Mandatory Access Control (MAC) configuration controls along with resource allocation controls like memory and CPU limits to mitigate DoS threats. Additionally, using minimal, optimized Docker images and read-only file systems can reduce the attack surface. Their findings emphasized the need for multi-layered defenses to secure container environments against malware attacks.

Wong et al. [4] used the STRIDE framework to analyze threats in the container ecosystem. While containerization offers advantages such as lightweight deployment and shorter startup times, it also poses security threats. The study emphasized that containers are vulnerable to a variety of threats, including privilege escalation, DoS, man-in-the-middle attacks, and data breaches caused by misconfigured Docker images or compromised credentials. Existing mitigating measures include multi-factor authentication (MFA), image security practices such as reducing attack surfaces and vulnerability screening, security patching, limiting administrative access, and ensuring correct isolation via Linux cgroups. While promising, these methods also have some drawbacks, such as the risk of stolen private keys in image signing and the variable effectiveness of certain approaches in vulnerability detection.

Qamar et al. [5] suggested a machine learning-based solution to detecting Distributed Denial of Service (DDoS) attacks that employs Recurrent Neural Networks (RNNs) and trains and evaluates on the KDD Cup 1999 dataset. The researchers evaluated three optimisation algorithms—Gradient Descent with Momentum, Scaled Conjugate Gradient, and Variable Learning Rate Gradient Descent—to determine the most effective technique for accurate and efficient DDoS detection. The neural network was trained and tested in MATLAB, and the Variable Learning Rate Gradient Descent algorithm produced the highest accuracy of 99.9% with the shortest training time
of 2 minutes and 29 seconds.

Madiha Amjad Hussain et al. [6] suggested a neural network-based approach for detecting fraudulent Android APKs by statically analysing permission requests and API calls. The model performed well, with an accuracy of 98.42%, a precision of 98.86%, and a low false positive rate of 0.0121. The neural network outperformed traditional classifiers like Random Forest and Na¨ıve Bayes in terms of detection accuracy. Precision-recall and separability tests validated the model's efficacy even with unbalanced datasets. Despite its high accuracy, the study noted issues such as false positives, data labelling, and the requirement for dependable datasets for real-world deployment.

Noor Mohd et al. [9] investigated the security vulnerabilities of Docker containers, which have direct access to the host kernel, as opposed to virtual machines, which must bypass the hypervisor. Docker containers are vulnerable to SQL injection and privilege escalation attacks due to their direct access. The research identifies different ways through which these attacks occur and stresses the need for good security practices such as running containers as non-root users and keeping images clean so as to reduce risks.

Devi Priya et al. [10] proposed a threat modeling approach that prioritizes container-related threats using the DREAD (Damage, Reproducibility, Exploitability, Affected Users, and Discoverability) criteria. It identifies vulnerabilities across different stages of the container lifecycle, from image development to runtime, and explores real-world exploits like resource exhaustion, privilege escalation, and API attacks. The authors presented strategies such as scanning for images, configurations for containers, secure communication over the network, and strict access control as some mitigation techniques

Jin et al. [11] proposed a container-based backup mechanism for the prevention against ransomware attacks. This will isolate the whole backup process within a container; hence, ransomware installed on the host cannot access vital

data. It ensures backup both locally and remotely. The mechanism includes automated backup operations, reducing reliance on users, thus reducing the possibility of users neglecting back- ups. This provides recovery of data when the container or host is attacked.

Yasrab [12] discussed security risks associated with Docker containers. Unlike typical virtual machines, Docker containers share the same host operating system kernel, increasing the risk of numerous attacks such as privilege escalation, insider threats, and kernel exploits. He discussed ways for mitigating these vulnerabilities, such as leveraging security frameworks like SELinux, AppArmor, and Docker-specific configurations. It emphasizes a proactive approach, recommending secure deployment guidelines to prevent attacks and limit damage in the case of breaches. Table 1 illustrates different techniques used to mitigate risk related to malware attacks in a containerized environment.

TABLE I

COMPARISON OF CONTAINER SECURITY TOOLS

Attribute	**Trivy**	**Clair**	**Aqua Sec.**
Malware Coverage	CVEs, Malware	CVEs only	CVEs + Runtime
Container Types Supported	Docker, Kubernetes	Docker	Docker, K8s, OpenShift
Integration Effort	Easy	Moderate	Complex
Scan Time	Fast	Medium	Fast
Cost	Free	Free	Paid
False Positive	Low	Medium	Low

TABLE II

STUDIES SHOW PROTECTION AGAINST MALWARE

References	**Contributions**
[13]	Suganthi Subramanian et al. propose a core approach for enhancing container security through two key policies: (a) static analysis to assess execution metrics of container images, and (b) dynamic analysis, which involves continuous monitoring of container activity during runtime.
[14]	The primary defense strategy involves isolating the container from the host system and other containers. Various techniques, such as lowering user privileges by running the container as a non-root user and restricting access to host resources like CPU and memory, can reduce risk. Additionally, implementing Role-Based Access Control (RBAC) limits container access, effectively minimizing the attack surface.
[15]	Banyai, E., and Gherasim, T. highlight that various scanning tools, such as Trivy, Aqua Security, and Clair, aid in automating the detection of vulnerabilities, outdated libraries, and harmful dependencies within container images.
[16]	Falco, A., et al. emphasize that monitoring resource patterns, including memory, CPU, and network usage within containers, helps identify unusual behaviors, such as sudden spikes in network traffic, which may indi- cate malicious activity.
[17]	According to Cito et al., analyzing and segmenting network traffic can help prevent unauthorized access and restrict inter- container communication. Adopting a zero- trust model is recommended as a best practice for containerized environments.
[18]	Turnbull suggests using lightweight base images, such as Alpine-based images, and adopting multi-stage build to minimize the attack surface within containers.
Our Contribution	A malware attack was conducted on a container built for our e-commerce application, and mitigation strategies were implemented, including the use of lightweight base images, multi-stage builds, and tools like Trivy to scan for vulnerabilities within the container.

III. MALICIOUS ATTACK AND MITIGATION APPROACHES

A. Malicious Attacks in Containerized Environments

Malware refers to various types of malicious software, including spyware, Trojan horses, worms, adware, and viruses, which can damage, steal from, and jeopardize the security of containers. Since 2015, there have been approximately 5 to 10 billion malicious attacks globally each year [19].

Some of the malicious attacks in containerized environments are listed below:

1) Unprotected Interfaces: APIs (Application Programming Interfaces) play a critical role in facilitating communication within containerized applications. If these APIs contain vulnerabilities, attackers can exploit them to compromise the entire system [20].

2) Image Vulnerabilities: Containers are built using pre-defined base images. If these images contain vulnerabilities or security loopholes, the security of the entire container can be compromised, leading to potential exploitation [20].

3) Container Privileges*:* Access to containers should be strictly controlled through proper containerization practices. If access permissions are not adequately set, unauthorized

users may gain access to the system, increasing the risk of exploitation [20].

*4) **Data Breaches**:* Containers can store sensitive and confidential data. Without proper security measures in place, attackers can exploit vulnerabilities to access this data easily, leading to potential data breaches [20].

B. Strategies to Mitigate Malware Attacks

The following approaches can be implemented to enhance container security against malicious attacks:

1) ***Use Of Lightweight Base Images**:* Alpine-based images, which are 30 times smaller than Debian-based images, can reduce the attack surface area.

```
FROM node:18-alpine
```

2) ***Update Base Image**:* Regularly updating images pro- vides patch fixes that can improve the security of your applications.

```
RUN apk update
```

3) ***Run Container as a Non-Root User**:* Run the container as a non-root user to restrict permissions.

```
RUN addgroup -S appgroup adduser -
    S appuser -G appgroup
    USER appuser
```

4) ***Using Multi-Stage Builds**:* In CI/CD pipelines, when you push code to the repository, the pipelines automatically perform various tasks such as downloading code, installing dependencies, compiling, building, and testing. These processes create additional layers, which can increase the size of your Docker images. To address this, we will implement a multi- stage build. This approach involves two stages:

1) Copying the source code and compiling it or creating the build.
2) Copying only the final build to a new base image results in a smaller final image containing only the build and the base image.

Large images can heighten security vulnerabilities and expand the attack surface.

5) ***Resource Quota:*** By default, a container can utilize the entire resources of the host machine. We need to impose restrictions on this to prevent potential security lapses. It's important to specify the CPU and memory our container requires.

```
Docker stat
```

6) ***Security Tools**:* Various security tools offer mechanisms for identifying, scanning, and analyzing vulnerabilities, as well as providing runtime protection and compliance checks within containers.

Examples include Aqua Security, Falco, Clair, Trivy, and Kube-Bench etc.

IV. Experimental Setup and Methodology

Containerized environments offer significant advantages, including deployment efficiency and scalability. However, they also introduce a distinct set of security challenges that must be addressed to mitigate the risks associated with containerization effectively. Understanding these challenges is crucial for maintaining a secure containerized ecosystem [21].

A. Executing a Malicious Attack in the Container

A Privilege Escalation and Trojan-based malicious attack is carried out by creating unauthorized folders and installing harmful software within the container.
The following steps were carried out to execute a malicious attack on the container:

*1) **Setting Up a Dockerfile in the Application Directory**:* First, a Dockerfile is created within the application directory to build its container. The Dockerfile contains the following commands, as shown in Figure 2:

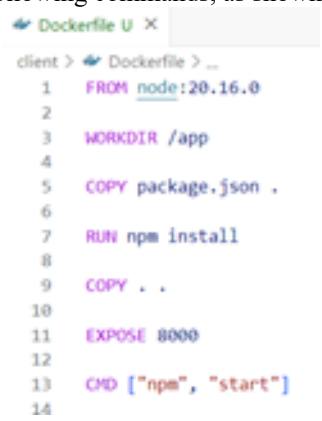

Figure 2. Docker File

1)

*2) **Creating a Docker Container**:* The application container is configured using Docker by initially building a Docker image of the application.
*3) **Running a Docker Container**:* A container is launched using the image of the application built previously.

*4) **Malicious Attack**:* A simulated privilege escalation and Trojan-based malware attack is carried out on the container by enabling root access, which exposes the system to vulnerabilities. This allows unauthorized access to the container and facilitates the creation of multiple malicious directories within the container

environment. The following steps were undertaken to simulate a malware attack on the container:

- **Run Container as Root User:** Initially, root access is granted to the container, making it vulnerable to malicious attacks. With root access, users can create harmful directories within the container, compromising its security and potentially leading to data and file corruption. The command shown in Figure 3 is executed to access the container and verify if it has root access:

```
PS C:\Users\Tesla laptop\Documents\Ecommerce-website> docker exec -it 3bfc712fbb31 sh
>>
# whoami
root
```

Figure 3. Run container as root user

- **Creating Malicious Folders:** A controlled simulation of a malicious attack was conducted within the Docker environment by programmatically creating unauthorized files and folders within the container.
Figure 4 represents the commands for creating the malicious folders inside the container.

```
# mkdir test
# ls
Dockerfile  node_modules  package-lock.json  package.json  public  src  test
# mkdir maliciousfile
# ls
Dockerfile  maliciousfile  node_modules  package-lock.json  package.json  public  src  test
```

Figure 4. Creating malicious folders

- **Installing unauthorized packages:** To further simulate a malicious attack, various software packages were installed within the container. In this scenario, the 'nmap' package was installed to represent unauthorized tool deployment by an attacker.
Figure 5 represents the command for installing of nmap package within the container.

```
# apt-get install nmap
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  dbus dbus-bin dbus-daemon dbus-session-bus-common dbus-system-bus-common libapparmor1 libblas3 libdbus-1-3
  liblinear4 liblua5.3-0 libpcap0.8 libpcre3 lua-lpeg nmap-common
Suggested packages:
  default-dbus-session-bus | dbus-session-bus liblinear-tools liblinear-dev ncat ndiff zenmap
The following NEW packages will be installed:
  dbus dbus-bin dbus-daemon dbus-session-bus-common dbus-system-bus-common libapparmor1 libblas3 libdbus-1-3
  liblinear4 liblua5.3-0 libpcap0.8 libpcre3 lua-lpeg nmap nmap-common
```

Figure 5. installation of nmap package

B. *Securing Docker Containers Against Malware Attack*

Although the container ecosystem faces various vulnerabilities and threats, several mitigation strategies have been developed to counter them.
The following steps can be implemented to mitigate security risks in a containerized environment:

1) ***Setting Up a Dockerfile in the Application Directory****:* A Dockerfile is created within the application directory, based on the mitigation approaches outlined in Section 3, to build the container. This Dockerfile includes the following commands as shown in Figure 6:

Dockerfile U

client > Dockerfile > ...

```
# Stage 1: Build the application
FROM node:18-alpine AS build

# Set the working directory inside the container
WORKDIR /app

#update the image
RUN apk update

# Copy the package.json and install dependencies
COPY package.json .
RUN npm install

# Copy the entire source code
COPY . .

# Stage 2: Create the runtime image
FROM node:18-alpine

# Set the working directory inside the container
WORKDIR /app

# Copy only the necessary files from the build stage
COPY --from=build /app /app

# Create a non-root user and group
RUN addgroup -S appgroup && adduser -S appuser -G appgroup

# Switch to the non-root user
USER appuser

# Expose the application port
EXPOSE 8000

# Start the application
CMD ["npm", "start"]
```

Figure 6. Commands in Docker File

2)

2) ***Creating a Docker Container****:* The application container is configured using Docker by initially building a Docker image of the application.

3) ***Running a Docker Container****:* A container is launched using the previously built image of the application.

4) ***Mitigation:***

- **Run Container as Non-Root User:** The access to the container is restricted by running it as a non-root user. This prevents users from creating directories or installing software, thereby enhancing the container's security. We grant access to a user named 'appuser'.
- **Use Alpine-based Image:** By using an Alpine-based image, the Docker image size was reduced from 2.55 GB to 837 MB, resulting in a more lightweight and efficient container as seen in figure 7.

```
PS C:\Users\Tesla laptop\Documents\Ecommerce-website> docker images
REPOSITORY                  TAG       IMAGE ID       CREATED         SIZE
build-frontend              latest    3380e6e0ab54   20 hours ago    837MB
new-frontend                latest    271c99328085   20 hours ago    1.67GB
mern-frontend               latest    37dfe769ef77   10 days ago     2.55GB
```

Figure 7. Using alpine based images

- **Verify Non-root access:** The command shown in Figure 8 is executed to access the container and verify if it has non-root access:

```
PS C:\Users\Tesla laptop\Documents\Ecommerce-website\client> docker exec -it b8b6facafea3 sh
/app $ whoami
appuser
```

Figure 8. Verify non root user access

- **Preventing Attack:** Subsequently, a privilege escalation and Trojan-based malware attack is simulated by attempting to create files and install software within the container. However, in Figure 9, it is observed that the user lacks the necessary permissions to perform these actions, indicating that the implemented security measures are effectively restricting unauthorized modifications.

```
/app $ mkdir maliciousfolder
mkdir: can't create directory 'maliciousfolder': Permission denied
/app $ apk update
ERROR: Unable to lock database: Permission denied
ERROR: Failed to open apk database: Permission denied
/app $ apk add map
ERROR: Unable to lock database: Permission denied
ERROR: Failed to open apk database: Permission denied
```

Figure 9. Preventing malware attack

- **Kernel Privileges:** Executed the following command to verify whether the user has kernel-level privileges.

```
/app $ more /proc/1/status | grep CapEff
CapEff: 0000000000000000
```

Figure 10. Kernel Privileges

Figure 10 illustrates that the output displays all zeros, indicating that the container has no elevated privileges. This minimizes the risk of unauthorized access and potential manipulation of the container.

*5) **Vulnerability Scanner**:* Various vulnerability scanners such as Trivy, Clair, Docker Bench for Security, Grype, and Falco can be employed to detect security threats and vulnerabilities within containers. In this research, the Trivy tool is utilized to enhance container security by identifying vulnerabilities within the application image.

- **Trivy:** Trivy is an open source vulnerability scanner that plays a crucial role in enhancing container security through automated vulnerability assessment and remedi- ation [22].

Trivy has been used to identify exploitable vulnerabilities within the container. It categorizes risks as low, medium, high, or critical as shown in Figure 11.

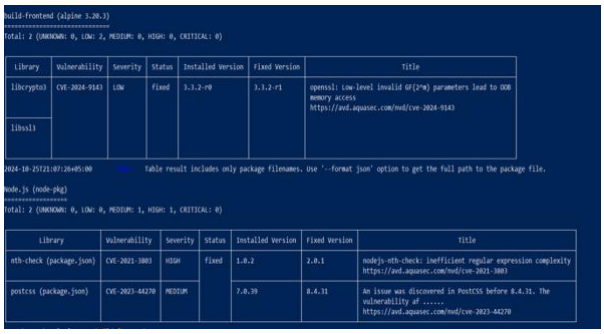

Figure 11. Categories of Risk in Trivy Tool

Trivy detected two low-severity OpenSSL vulnerabilities in the base Alpine image, along with a high-severity issue in the Node.js dependency nth-check and a medium- severity vulnerability in the postcss package. These vulnerabilities could potentially lead to security breaches within the container environment. All identified vulnerabilities were mitigated by upgrading the affected packages to their corresponding fixed versions. Trivy facilitated the identification and resolution of these issues, thereby enhancing the overall security of the container.

*6) **Security Validation of Trivy**:* Trivy's security capabilities were evaluated using a dedicated security validation process. Publicly available susceptible photos were utilized as test cases to evaluate Trivy's detection ability. The tool successfully found a wide range of vulnerabilities in these images, including multiple CVEs of high and critical severity. The evaluation parameters comprised detection count, severity categorization, false positives, and scan length. Trivy displayed great detection accuracy and few false positives. However, certain low-severity concerns were either overlooked or incorrectly recognized. Trivy's lightweight command-line interface and native support for CI/CD pipelines make integration simple. Nevertheless, issues were encountered during scans of huge multi-layer pictures, when scan times rose and certain runtime-specific threats went unnoticed. These findings highlight Trivy's strengths as a static vulnerability detector.

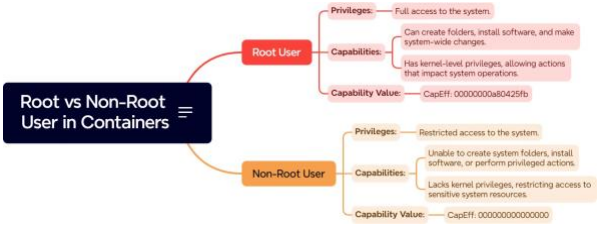

Figure 12. Comparison between Root and Non-Root User

Figure 12 demonstrates the behavior of a container when running as a root user compared to running as a non-root user.

Table 3 presents the overall methodology implemented in this research which includes the malware attack on the container and the mechanism to safeguard the container from malware attacks.

TABLE III

COMPARISON OF MALWARE ATTACKS AND MITIGATION STRATEGIES

Malware Attack on Container	Attack Mitigation in Container
A container built with root user privileges is susceptible to malware attacks.	To mitigate this risk, we switch from the root user to a non-root user by adding a specific command in the Dockerfile.
Large container images, especially those around 2.55 GB in size as in our case, have an increased risk of vulnerabilities.	We minimize the image size by opting for an Alpine-based image and implementing a multi-stage build which reduces the size to 837 MB.

Containers with elevated access privileges face a heightened risk of exploitation.	Limit user access privileges strictly to essential permissions only.
Without thorough scanning and analysis using a security tool, containers are vulnerable to attacks.	Use security tools like Aqua Security, Trivy, Falco, and Clair to scan containers for potential vulnerabilities.

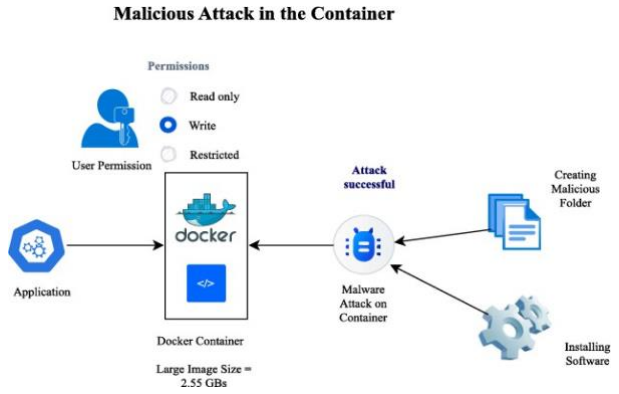

Figure 13. Malware Attack

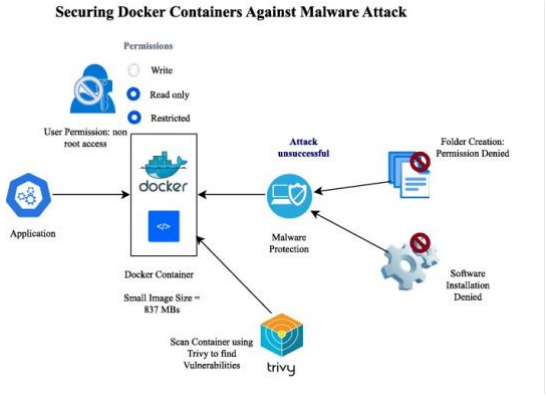

Figure 14. Attack Mitigation

Figures 13 and 14 illustrate the overall methodology adopted in this research for simulating the malware attack and implementing the corresponding mitigation process.

V. Result

The findings indicate that preventing root access is essential to securing containers against privilege escalation and Trojan- based malware attacks. Running containers with root privileges introduces significant security vulnerabilities, as it allows attackers to create malicious directories or install unauthorized software within the container.

To mitigate these risks, this research applies a non-root user configuration, which effectively limits user permissions and prevents unauthorized file creation, directory modification, or software installation. This enhances the overall security posture of the container. Furthermore, the use of an Alpine- based image combined with multi-stage builds significantly reduced the image size from 2.55 GB to 837 MB thereby improving efficiency and lowering the attack surface.

Lastly, the Trivy tool was utilized to scan the container image, which initially identified two low-severity, one high-severity, and one medium-severity vulnerability in the Alpine- based image and associated Node packages. All detected vulnerabilities were addressed by upgrading the affected packages to their respective fixed versions. These combined mitigation practices significantly enhance the container's security and reduce the risk of successful malicious activities in the deployment environment.

VI. Conclusion

With the advancements in container use, many users are subjected to malware attacks and security breaches. As a result, protecting containers is finally a long-term problem because of new emerging complex threats and issues. Earlier, solutions like Ubuntu with Clair, and Grape offered only a limited level of protection, leaving a variety of open doors, including unprotected APIs and root user privileges, uncovered. However, this research study provides a solution to restrict the privileges of root users to non-root users, updating the base images frequently, and using lightweight builds which will further contribute to an extra layer of protection upon identifying malware and threats. Therefore, for this purpose, an open-source malware protection tool called Trivy was used to scan images of the containers of the root users and upon identifying malicious activity it restricts the privileges to non-root users preventing security breaches and installation of viruses and updation of files and directories. This set of mechanisms greatly decreases the probability of attacks in conditions of containerization and closes the security gap between the prior approaches. These techniques will be important as containers persist to protect cloud-native applications and infrastructure.

References

[1] P. R. C. a. S. A. K. J. Chelladhurai, "Securing Docker Containers from Denial of Service (DoS) Attacks," in 2016 *IEEE International Conference on Services Computing (SCC)*, San Francisco, 2016.

[2] S. K. a. J.-H. L. H. Lee, "Experimental Analysis of Security Attacks for Docker Container Communications," Electronics, vol. 12, no. 4, p. 940, 2023.

[3] H. C. S. W. a. C. H. D. Huang, "Security Analysis and Threats Detection Techniques on Docker Container," in 2019 IEEE 5th International Conference on Computer and Communications (ICCC), Chengdu, 2019.

[4] E. C. M. O. a. J. Z. A. Wong, "Threat Modeling and Security Analysis of Containers: A Survey," 2021.

[5] B. A. Z. A. A. A. F. H. K. a. F. A. J. Roheen Qamar, "Detecting Distributed Denial of Service attacks," University of Sindh Journal of Information and Communication Technology (USJICT), vol. 5, no. 2, 2021.

[6] S. M. K. S. M. a. S. R. H. Madiha Amjad Hussain, "An Efficient Malware Detection Approach for Malicious Android Application," University of Sindh Journal of Information and Communication Technology (USJICT) , vol. 5, no. 3, 2021.
[7] T. D. N. A. S¸ . T. F. V. T. K. a. A. -R. S. M. S. Haq, "A Comprehensive Analysis and Evaluation of Docker Container Attack and Defense Mechanisms," in 2024 IEEE Symposium on Security and Privacy (SP), San Francisco, 2024.
[8] E. K. R. T. a. R. P. S. Mullinix, "On Security Measures for Containerized Applications Imaged with Docker," arXiv, 2020.
[9] D. U. N. Mohd and V. Kuriyal, "Security Implications in Docker Based Virtual Environment," Webology, vol. 18, no. 4, p. 2373, 2021.
[10] S. C. S. M. K. K. D. P. V. S. Devi Priya, "Container security: Precaution levels, mitigation strategies, and research perspectives," Computers & Security, vol. 135, 2023.
[11] M. T. S. M. Y. K. Y. Jin, "A Secure Container-based Backup Mechanism to Survive Destructive Ransomware Attacks," *International Conference on Computing, Networking and Communications (ICNC)*, Maui, HI, USA, 2018.
[12] R. Yasrab, "Mitigating Docker Security Issues," April 2018.
[13] S. S., P.B. H. Shylaja,"Container Security: An Extensive Roadmap", *3rd International Conference on Integrated Intelligent Computing Communication and Security (ICIIC 2021)*, pp. 427-436, 2021
[14] Xu, X., et al. "A Survey on Security Issues in Container-Based Virtualization." *IEEE Transactions on Network and Service Management*, vol. 16, no. 1, 2019, pp. 70–85.
[15] Banyai, E., and Gherasim, T. "Enhancing Security for Docker Contain- ers." *Journal of Cloud Computing: Advances, Systems and Applications*, vol. 9, no. 1, 2020.
[16] Falco, A., et al. "Resource Monitoring for Containerized Applications." *International Journal of Network Management*, vol. 30, no. 2, 2020, e2074.
[17] Cito, J., et al. "Network Isolation in Docker: Improving Security through Proper Network Segmentation." *IEEE Communications Surveys and Tutorials*, vol. 21, no. 4, 2019, pp. 2854-2870.
[18] Turnbull, J. "The Docker Book: Containerization Is the New Virtualization." *Turnbull Press*, 2021.
[19] Mitchell, Brian S and Chandnani, Ansh and Carter, John and Roumelioti, Danai and Mancoridis, Spiros, "Malware Detection in Cloud Native Environments", *AICCC*,pp. 14-16, Dec. 2024.
[20] "Container Security Vulnerabilities: Types, Assessment, and Mitigation", *Checkpoint*, accessed October 27, 2024.
[21] Budi Pranoto, "Threat Mitigation in Containerized Environments", *ARAIC*, vol. 6, no. 8, pp. 22–38, Aug. 2023.
[22] "Enhancing Container Security Through Automated Vulnerability Scanning and Remediation with Trivy" *Insights2Techinfo A Platform for Researchers and Technology Enthusiasts*, accessed October 27, 2024.