Secure-by-Design Docker Containers

Weijun Huang
Tandon School Of Engineering
New York, USA
wh2531@nyu.edu

Abstract*—This study proposes a three-layer container security workflow that integrates Dockerfile linting, image vulnerability scanning, and subnet-based network isolation to reduce vulnerabilities across the container lifecycle. The workflow was evaluated by comparing unprotected Dockerfiles and images with their hardened counterparts after corrections guided by Hadolint and Docker Scout. Simulating attacker-assessed network security scans in both a baseline bridge network and a hardened isolated subnet, demonstrating the effect of runtime isolation. Experimental results show that the workflow reduced significant vulnerabilities, and prevents attacker containers from discovering internal services, which enhances container security.

Keywords—docker container,vulnerability, security layer

I. INTRODUCTION

As cloud technology gets better, containers have become more popular because of their portability and lightweight nature. However, unlike virtual machines, which isolate the kernel by hypervisors, containers share the same host operating system kernel. This shared environment introduces significant security risks. For example, a container running as root can escalate privileges, and ARP spoofing can compromise the host or neighboring containers when they share the same default network bridge. These vulnerabilities make container security a growing concern.

To mitigate the above problem, previous studies have proposed various solutions for specific aspects of the container. The earlier works focus on utilizing Linux primitives, such as cgroups and namespaces, for isolation. Recently, studies proposed a model to prevent Dockerfile misconfigurations before image creation or applying image scanning tools, such as Clair. However, these solutions are more focused on enterprise-level container or Kubernetes deployments, which makes them less friendly for small-scale use cases.

In this paper, we propose a complete container security framework that integrates three defense layers: configuration analysis, vulnerability scanning, and network isolation. We use Hadolint for detecting misconfigurations in Dockerfiles. Then, we scan the image using Docker Scout to detect known CVEs (Common Vulnerabilities and Exposures). Finally, we isolate the network and implement a Firewall container (FWC) to build a gateway between external network and internal network. We hypothesize that combining these techniques and tools into a single workflow will significantly reduce high-severity vulnerabilities and minimize the risk of host compromise.

The paper is structured as follows: Section II reviews related research for building secure container. Section III presents design and expectations for the results. Section IV evaluates the empirical evidence. Section V concludes the findings and outlining future work.

II. RELATED RESEARCH

Mahajan's and Mane's study analyzes the problem of container deployment misconfigurations and proposes a solution: building a configuration check model that prevents Dockerfiles with unsafe configurations from being used to build images. Their approach emphasizes securing the container at the earliest stage of deployment. Building on their idea, we are using Hadolint, an open-source Dockerfile linter, to catch the most common misconfigurations based on up-to-date best practices. [3]

Nan Yang et.al.'s study analyzes the security of container images, particularly the issue of container escape, where a process running inside the container can access resources outside of the container without permission. The authors propose a four-layer defense model which not only can scan the image for vulnerabilites and measure the integrity of image but also monitor the process and provide network restriction. Based on their design idea, instead of using Clair as a mirror scanner, we are using Docker Scout, a lightweight CVE scanner introduced in 2023, designed for individual users who focus on single-container security, while referring to their network isolation concepts.[1]

Asem et.al.'s study analyzes the problem of containers and hosts sharing the same default network bridge. If an attacker compromises one container, they can easily access other containers or even the host through MITM and ARP spoofing attacks. The authors propose a security design that isolates the network between the host and related containers by utilizing a FWC, which acts as a gateway between the external network and internal containers network through iptables and NAT rules. Based on their design idea, we add simple FWC as the final layer of defense in our design to enhance network security. [2].

III. HYPOTHESIS

Previous studies lack a comprehensive design workflow from Dockerfile to running a container, focusing instead on a single aspect of container security, such as configuration checking (Dockerfile), vulnerability scanning (image), or network isolation (running container). Moreover, the current solutions are more focused on enterprise-level containers or Kubernetes, and the tools they use are complicated to use and set up. Our proposed solution integrates three layers of defense across the container lifecycle, utilizing lightweight tools and providing easy setup. We hypothesize that by combining these techniques into a single workflow, the container's vulnerabilities will decrease significantly compared to one deployed without any protection. To validate this, we compare a deliberately insecure container with a hardened

one that applies each layer of our defense. The measurement is based on the number of vulnerabilities found at each stage using Hadolint, Docker Scout, and network isolation tests.

IV. EMPIRICAL EVIDENCE

The objective of this study is to validate our hypothesis that a multi-layer container security workflow—combining Dockerfile linting, image vulnerability scanning, and runtime network isolation—can significantly reduce container vulnerabilities compared to an unprotected deployment. Our evaluation covers the entire container lifecycle, beginning with static Dockerfile analysis (layer 1), followed by image vulnerability scanning (layer 2), and finally runtime network test with FWC (layer 3).

Layer 1 Setup: Three Docker files were prepared

- H1 – Original, insecure Dockerfile (such as using outdated packages, root user privileges, installing unnecessary packages, and risky configuration)
- H2 – The Dockerfile corrected after Hadolint recommendations (Install required packages only, switch to non-root user, modify the configuration)
- H3 – The Dockerfile was further improved by Docker Scout recommendations (using the correct version of packages)

Layer 2 Setup: Using above Dockerfile to build corresponding Docker image (M1, M2, M3) for CVE detection using Docker Scout.

Layer 3 Setup: Creating two containers' network scenarios to compare:

1. Baseline: Building a web container and an attacker container. Then deploy them on the default bridge network (172.17.0.2).

2. Hardened: Building a web container that was deployed on an isolated internal network (172.18.0.2) behind an FWC, while the attacker remained on the default bridge network.

In this case, we pull the FWC from Docker Hub and use it as a gateway to connect the internal network and the external network. And the attacker container will run Nmap to detect all reachable IP addresses.

	Error	Warning	Info	Ignore	Style	Unknow
H1	1	2	2	0	0	0
H2	0	1	0	0	0	0
H3	0	1	0	0	0	0

Table 1. Layer 1 - Hadolint analysis of Dockerfile vulnerabilities (H1-H3)

	Critical	High	Medium	Lower	unknow
M1	3	20	10	213	5
M2	2	13	10	210	5
M3	0	1	1	44	0

Table 2. Layer 2- Docker Scout detect Image's vulnerabilities

	IP address	Number of IP Discovered
Attacker 1 (Baseline)	172.17.0.3	4
Attacker 2 (Hardened)	172.17.0.4	2

Table 3. Layer 3- Network Scan Results: Baseline vs Hardened

From Table 1, we find that the number of misconfigurations reduced after we corrected the Dockerfile configuration based on the feedback from Hadolint.

From Table 2, the number of vulnerabilities reduces significantly after we make corrections to the Dockerfile based on the Docker Scout feedback. Most vulnerabilities could be eliminated by updating the package version and base image. Then we can follow the CVE detail step by step to eliminate the vulnerabilities if necessary.

From Table 3. The first baseline scenario shows that the attacker successfully discovered the IP address of the host bridge, and other containers within that subnet (such as web1, attacker1, and attack2), indicating a risk of lateral movement. In the second scenario, the attacker can only detect the host bridge and its IP, while the target container in the isolated network remains invisible, indicating isolate the subnet could mitigates lateral attacks (without FWC).

In summary, the three-layer workflow reduces misconfigurations from Dockerfile, eliminates critical CVEs from the image, and prevents runtime container discovery, providing support for the hypothesis. Although we did not demonstrate how FWC protects our network security, it acts as a central traffic gateway, enabling NAT and traffic monitoring, which would be valuable for larger deployments or multi-host environments where additional network control is required.

V. CONCLUSION

In conclusion, this study demonstrates that a three-layer container security workflow can significantly reduce vulnerabilities across the container lifecycle. By combining Dockerfile linting, image vulnerability scanning, and network isolation via subnet separation we achieved measurable security improvements. Although effective, all three layers currently require manual testing and configuration, which can be time-consuming for individual developers.

Future work will focus on automating this workflow. Currently, our methods require manual testing and manual corrections to the Dockerfile. A possible approach is to integrate Layer 1 (Dockerfile linting) and Layer 2 (image vulnerability scanning) into a pipeline powered by a machine learning model that can automatically detect and correct misconfigurations. This would reduce developer effort, speed up the hardening process, and ensure consistent application of security best practices.

REFERENCES

[1] Nan Yan, C. Chen, T. Yuan, Y. Wang, X. Gu, D. Yang, "Security hardening solution for docker container," *2022* International Conference on Cyber-enabled Distributed Computing and Knowledge Discovery (CyberC), doi: 10.1109/CyberC55534.2022.00049.

[2] A. Mousa, W. Tuffaha, M. Abdulhaq, M. Qadry, Othman Othman M.M., "In-depth network security for Docker containers," 2023 14th International Conference on Computing Communication and Networking Technologies (ICCCNT), doi: 10.1109/ICCCNT56998.2023.10307493.

[3] V. B Mahajan, Dr.Sunil B Mane, “Detection, Analysis and Countermeasures for Container based Misconfiguration using Docker and Kubernetes,” 2022 International Conference on Computing, Communication, Security and Intelligent Systems (IC3SIS), doi: 10.1109/IC3SIS54991.2022.9885293