Published by :
http://www.ijert.org

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 12 Issue 05, May-2023

Containers and Supply Chain Vulnerabilities Container Vulnerabilities in Different Layers

Alok Kumar
Application Security
5Sec Cyberpwn Technologies Pvt. Ltd
Bangalore, India

***Abstract*:- There are many OS-Level virtualization products but this research is focused on Docker and Linux host operating system as these in combination are most widely used products across the industry. This research focuses on the possible vulnerabilities in whole supply chain and categorizing the vulnerabilities on basis of layers so that it is easy to detect, prioritize the remediation process and remediate identified vulnerabilities. The research is done to identify the security gaps in the supply chain environment and also possible ways to remediate the vulnerabilities. Also, Linux Security Modules (LSMs) is also introduced as an added security to enforce policies on the host operating system so that risk and abuse on host operating system can be reduced making the supply chain secure.**

Keywords: Docker, container, docker images, docker registry, docker network, supply chain, orchestrator, LSMs

INTRODUCTION

Docker containers are being widely used to deploy standalone application code or microservices in a CI/CD deployment environment because it provides multiple benefits over traditional virtual machines. The research is done by reading already published research papers and other available content on internet. It is observed that securing only docker images and host operating system by following different benchmark is not enough to secure the whole Supply chain environment so, the supply chain is first categorized in different layers as Host Operating System (1) where the docker daemon runs, Container Runtime (2), Docker Registry (3), Docker Images (4), Docker Network (5) and finally the Orchestration framework (6) which is responsible for scaling and managing different running containers in an environment. After categorizing the layers, the research is done on the possible vulnerabilities on each layer and their possible remediations.

An introduction is done on containers [I], traditional virtual machines [II]. A comparison is done between containers and virtual machines [III]. Section A contains the possible attack surfaces in a typical supply chain environment followed by conclusion where LSM [Table 10] is also introduced as a security feature which can be used to enforce extra policies to make the environment more secure from possible threats.

I. CONTAINERS

Docker is a set of platform as a service products that use OS-level virtualization to deliver software in packages called containers. Docker, LXC, and RKT are examples of container managers. This study is solely focused on Docker and linux host Operating system as it is the most common use case scenario.

Containers take comparatively much lesser time to start the service as it is kept minimalistic by design. As Containers share resources and kernel with host Operating system, it doesn't need any Operating system to be installed to run any particular service.

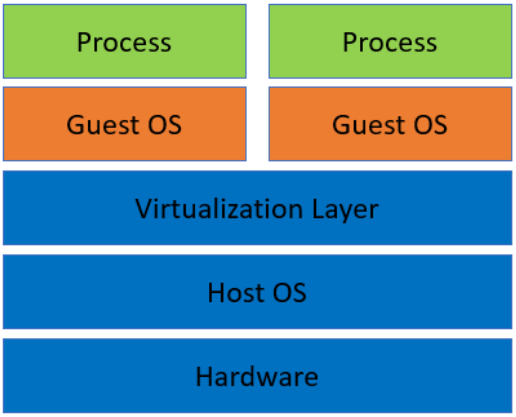
II. VIRTUAL MACHINES

A virtual machine (VM) is a digital version of a physical computer. Virtual machine software can run programs and operating systems, store data, connect to networks, and do other computing functions, and requires maintenance such as updates and system monitoring.

Virtual machine needs an Operating system to be installed and mimic the behavior of physical machine. Hence it needs an Operating system and kernel to be installed which is time consuming and takes much more effort than compared to Containers.

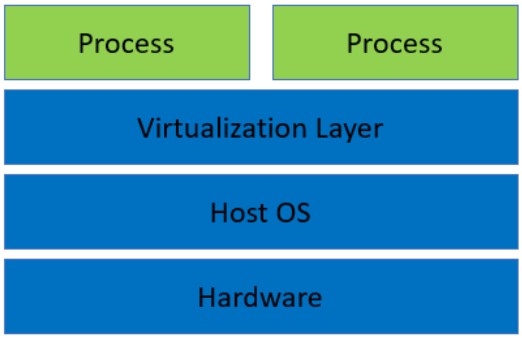
III. CONTAINERS VS VIRTUAL MACHINES

Hardware, Host Operating system, Virtualization Layer are the common layers across any virtual machine or any containers. The major difference can be observed in image (i) below as the virtual machines needs a separate Operating system to be installed to function properly whereas we can see in image (ii) that there is no Guest Operating system involved.



i. Virtual Machines

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 12 Issue 05, May-2023



ii. Containers

Factors	Containers	Virtual Machines
Operating system Kernel	Share kernel with host Operating system	Needs separate Operating system and kernel
Resources	Can share with host	Needs separate resources
Load Time	Can be started instantly	Needs its own time to start
Applications	Ideal for micro services	Ideal for servers hosting applications

Table 1: Difference between Containers and Virtual Machines

A. Possible Attack Surfaces

1. Host Operating System

Operating System sits in middle of hardware and virtualization layer. Host Operating system is responsible for running Docker engine. Any Linux Operating System can be used to run Docker engine as recommended by Docker and like any other OS, it should be made sure that the Host Operating system is not vulnerable and unnecessary services should not be exposed.

Attack vectors on Host Operating System is listed in Table 2 where we can see **Improper access rights** [Table 2 (i)]. All users having access rights for docker means unintended users can start/stop or interfere with docker daemon or in case of any user getting compromised creates risk to the docker daemon and the supply chain as well.

Host Operating System components [Table 2 (ii)] can be vulnerable to attacks leading to compromised supply chain for example CVE-2018-10900 can lead to privilege escalation attacks.

Overly permissive SSH [Table 2 (iii)] can allow users to connect remotely leaving docker engine accessible to users from anywhere. SSH access should be allowed to limited users and interaction to docker engine should be done by docker APIs only.

Kernel Vulnerabilities [Table 2 (iv)] running under Operating System can also be vulnerable to attacks like CVE-2016-5195 which can lead to Privilege escalation or Remote Code Execution vulnerabilities. If an attacker gets root privilege on the host Operating system then the whole supply chain can be considered as compromised.

Docker daemon always runs as root user because it needs to create a unix socket. **Running docker daemon as root** [Table 2 (v)] can also create security issues which can bring risk to daemon and even container runtime. The user privileges are propagated inside containers as well. Running containers with root privilege can cause container escape vulnerabilities.

Running stale services or exposing **Unnecessary services** [Table 2 (vi)] on host Operating system increases the attack surface area. Vulnerable or misconfigured services exposed on host operating system can lead to Denial of Service (DoS) or even Remote Code execution vulnerabilities. For example CVE-2018-14009 can allow authenticated users to trigger remote code execution which gives access to the server running docker daemon.

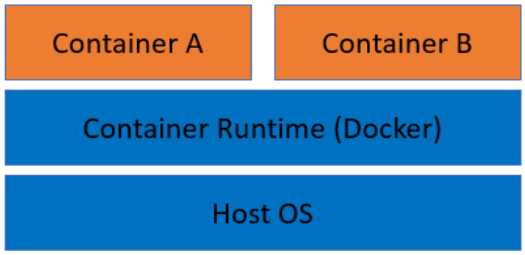
International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 12 Issue 05, May-2023

S. No.	Attack Vector	Scenario	Recommendations
i	Improper access rights	An employee/user having rights to access docker engine gets his account compromised.	Separate user should be created and added to docker group to manage docker daemon.
ii	Host Operating System components	Running outdated or components with known vulnerabilities.	Host Operating system and its components should be kept up to date. System Hardening can be done by following CIS benchmarks.
iii	Overly permissive SSH	An employee/user having rights to access docker engine gets his account compromised. An attacker logs in to server via SSH	SSH access can be limited to selected users (if required).
iv	Kernel Vulnerabilities	An attacker uses exploits to achieve remote code execution or privilege escalation on server.	Latest stable kernel should always be used.
v	Running docker daemon as root	User inside container having root access can escape containers and interact with host operating system	Separate user should be created and added to docker group to manage docker daemon.
vi	Unnecessary services	An attacker if compromises any running service, can get access to docker engine	All running services should be updated and configured properly. Services should be stopped if not in use.

Table 2: Attack vectors on Host Operating System

2. CONTAINER RUNTIME

Container runtime [image iii] is a software which is responsible for running containers on host operating system. Container runtime for example Docker, sits on top of host Operating system and can run applications inside containers as required.



iii. Container Runtime

Common attack vectors, scenarios and possible recommendations are mentioned in Table 3 below. As in [Table 3 (i)], **Containers should not be run as root user** because in case an attacker exploits the application to achieve remote code execution then the host operating system can also get compromised. Or if any existing container has malicious code running inside container can have code execution on whole host operating system.

Write access for host root file system [Table 3 (ii)] can also help attackers or malicious containers to achieve code execution by modifying host root files. An attacker can edit "/etc/passwd" file and add decoy user, expose ports and access the containers remotely.

Linux **Capabilities** [Table 3 (iii)] are group of kernel calls, these groups can be assigned to per-process. By using linux capabilities a standard user can execute programs which can make kernel calls as assigned. A compromised or malicious container having capabilities such as CHOWN can change ownership of the files and achieve write access to the disk.

Containers running in privileged mode [Table 3 (iv)] means that the running container will have complete root access on host machine. Which if compromised or is malicious can allow attackers to achieve code execution on host operating system.

Unbound network access from containers [Table 3 (v)] means all containers can interact with each other, which is by default. A container running malicious code can help attackers to sniff the traffic or craft DoS attacks from inside the network. An attacker can also use the same to send malicious requests other running application containers.

Containers having sensitive mounts on host [Table 3 (vi)] can also make security risks. Having sensitive mounts on host can give attacker/malicious user on host to modify the services or complete write access to container mounts. That can be abused by an attacker to compromise containers giving ability to interact with other containers in same network.

Containers with known vulnerabilities [Table 3 (vii)], the applications running inside container can have known vulnerabilities or exploits available. Or even the image being used for creating containers can have publicly known vulnerabilities which can give an opportunity to attacker to exploit known vulnerabilities and compromise the containers. It is always advisable to use latest stable images to create containers.

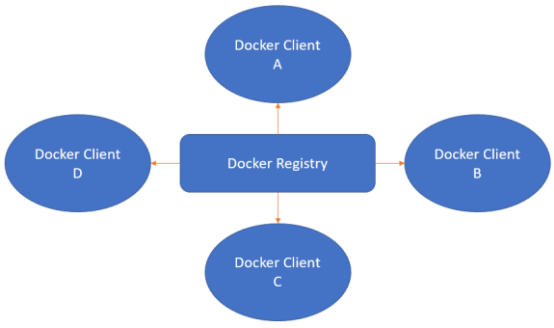
International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 12 Issue 05, May-2023

S No.	Attack Vector	Scenario	Recommendations
i	Containers running as root user	Attacker compromises application inside container and gets root access.	Containers should always be ran using low privilege user or separate user can be created with required privilege to manage containers.
ii	Write access for host root file system	Compromised/Malicious containers can modify the root file system	Root file system should not be writable from inside containers.
iii	Capabilities	Compromised/Malicious containers running with CHOWN capability can file ownership or group.	Excessive capabilities should be dropped from containers.
iv	Containers running in Privileged mode	Compromised/Malicious containers running in privileged mode has all root access on the host.	Privileged mode should never be used on production.
v	Unbound network access from containers	Compromised/malicious container can send malicious requests to other containers or services in network.	Containers should be kept in separate network if multiple containers need to interact with each other. Continuous monitoring of network should be done.
vi	Containers having sensitive mounts on host	Host operating system gets compromised and now attacker has full write access to container's sensitive mounts.	Directories with sensitive data should never be mounted on host.
vii	Containers with Known Vulnerabilities	Container was created using old image which has publicly known vulnerabilities. An attacker can compromise whole container by exploiting the vulnerabilities.	Image scan should be done for finding existing vulnerabilities. Tools like Trivy can be used to scan for known vulnerabilities.

Table 3: Attack vectors on Container Runtime

3. DOCKER REGISTRY

Docker registries [image iv] are a setup for distribution of Docker images. Registries are public and private in nature. Organization can host their own docker registries to distribute docker images and also can keep version control by methods like adding specific tags or versions to the images.



iv. Docker Registry

There can be multiple scenarios and vulnerabilities on a docker registry as well. For example, **Docker registries left unauthenticated and exposed to public** [Table 4 (i)] can cause severe damage to the organization. It can open door of opportunities to attacker and help him getting internal applications and in worse case scenario even he can modify the images with malicious code in it. Corporate docker registries should never be left exposed to public and proper authentication mechanism should be made mandatory in place to remediate such attacks.

Second attack vector can be the **Misconfigured Docker Registry applications** [Table 4 (ii)]. Misconfigurations can give birth to vulnerabilities in any case and docker registries are no different. Taking a scenario for example as a docker registry which doesn't has any access management in place and allows user to pull and push images to the registry. This can even be worse if there is no authentication in place. An attacker can pull any image, modify it with malicious codes and push the image back to registry. This will lead users pulling malicious images and running vulnerabilities in supply chain. Access Control should be managed using IAM if using GCP or AWS. Registry access management can be used to maintain access control for the user.

Application owners or owners docker images often try to resolve existing bugs/vulnerabilities and update the images with version number in tags. **Stale images** [Table 4 (iii)] can have older version of running base image or outdated application running inside which can pose as an attack vector. Outdated base images or application can have known vulnerabilities and available public exploits as well. It is always advised to update the base image to latest stable version. Note that updating of base images to latest stable version should always be done by taking approval from the organization as updating images might break other dependencies required by the application.

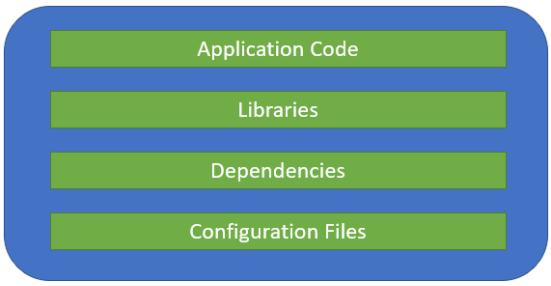
International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 12 Issue 05, May-2023

S No.	Attack Vector	Scenario	Recommendations
i	Unauthenticated public registry APIs	Attacker identifies public registries using search engines like Shodan and can interact with the registry.	Docker registries should always have authentication in place as mandatory rule. Private docker registries should never be left exposed to internet.
ii	Misconfigured Applications	Attacker identified misconfiguration in registry where he can pull or push docker images.	Access control with IAM can be used in case of GCP or AWS. Registry Access Management can be used. Registry Access Management is available to Docker Business customers only.
iii	Stale Images	Attacker identifies stale images having known vulnerabilities due to no update.	Base images and applications should be updated to latest stable versions. Remove the images from the registry which are no longer in use.

Table 4: Attack vectors on Docker Registry

4. DOCKER IMAGES

Docker images [image v] are basically templates which are used to start containers. Docker images contain a Application code, Libraries, Dependencies and configuration files. They provide capability for developers and operational team to setup a light weight environment to run the application. For example, if we need to host any web application, we can select a base image of web server like nginx, create required configuration files. Volume mounts, environment variables can also be defined in docker images as well so that container runs with all required configurations for the application.



v. Docker Image

Possible attack vectors on docker images are listed in below [Table 5]. Selecting incorrect or older **Base images** [Table 5 (i)] can bring outdated software packages to the environment which might already have known vulnerabilities leading to risks for any organization. Latest stable version of base images should always be preferred to create any docker

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 12 Issue 05, May-2023

image so that we can consider that there are no publicly known vulnerabilities. If the base image has vulnerabilities, then it can ruin all the efforts of writing secure code and following best practices.

[Table 5 (ii)] describes the vulnerabilities or risks which might come with the images such as **embedded malwares.** Again, selecting docker images without verification might bring embedded malwares with the images. Malicious actors can create and host docker images containing embedded malwares in it which if used can cause security issues like Data exfiltration, crypto mining, sniffing of network traffic. To prevent from this type of vulnerabilities, images should be scanned for vulnerabilities and misconfigurations. Docker image hash can be compared for known malicious hashes of docker images.

Developers often **hardcode secrets** [Table 5 (iii)] in docker images as its easy to deploy. Hardcoding of secrets is considered as bad practice in any ways and docker images are no different. For example, the application has payment gateway integrated. In this case the developers might want to hardcode the tokens in the image itself so the application will run without errors. Any one having docker image can see the configuration of it leading to credential leak. Secrets or tokens should always be provided to containers when needed.

Image trust [Table 5 (iv)] can also be considered a cause of vulnerabilities. Using untrusted images in production environment can bring known vulnerabilities or malware into the network. Malicious actors can embed code to perform activities like crypto mining, monitoring the network or escaping containers to gain access on host machine. Images from trusted source should only be used to lower the risk of vulnerabilities and getting exploited.

Docker images are templates to start containers which also contains configuration. Using images from trusted source, checking for embedded malwares, removing hardcoded secrets and checking for existing known vulnerabilities takes effort which can all go to vain if not checking for **poor configurations** [Table 5 (v)]. [Table 6] describes possible misconfiguration which can be avoided to make the running containers secure.

S No.	Attack Vector	Scenario	Recommendations
i	Base Images	Team selects outdated base images or images with existing known vulnerabilities.	Base images should be selected properly so that they are of latest stable version and having no existing known vulnerabilities.
ii	Embedded Malwares	Malicious actors inject malware codes into the image for example crypto mining.	Image should be scanned for vulnerabilities and should be downloaded from trusted sources.
iii	Hardcoded Secrets	Team decides to hardcode API keys to the image because it's convenient and easy.	Secrets should always be provided to containers when needed. Many orchestrators provide feature to manage secrets which can be used.
iv	Image Trust	Team decides to use an image from an unknown or untrusted source where an attacker has already placed malicious code into the image.	Images should always be downloaded from trusted sources.
v	Poor Image Configurations	Team decides to use a base image which has root user login enabled.	Images should always be scanned for misconfigurations and configuration audit can be done following the standards like CIS Benchmark.

Table 5: Attack vectors on Docker Images

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 12 Issue 05, May-2023

S. No.	Misconfiguration	Remediation
i	Use of root user account for containers	Root user account should always be avoided for containers. Many base images like Alpine come with support of root user which should be considered properly and root user should always be avoided for running containers
ii	Unwanted users as part of docker group	Unwanted users should never be used/added to docker group and proper authorized person should be only able to execute docker commands.
iii	Mounting docker.sock on containers	Mounting docker.sock should be always not to consider for misconfiguration
iv	Using "-privileged" flag to run containers	Using "-priviledged" flag should always be avoided as it gives root access to containers on host operating system.
v	Exposed docker daemon over HTTP	Docker daemon remote APIs should never be exposed to the network. Although it's not exposed by default.

Table 6: Poor Docker Image Configurations

5. DOCKER NETWORK

Docker containers are used to run applications or most often micro-services which need to interact with other containers. Docker provides different network drivers [Table 7] which can be selected as per requirement to solve the issue. Docker networks are used to connect containers to other containers and the internet. Hence, Docker networks should also be given priority in securing a container environment. [Table 8] describes the possible vulnerabilities which can affect the network where containers interact with each other or the internet. And network monitoring should be done to identify possible threats or anomalies in the network.

S. No.	Network Driver	Description
i	Bridge	The default network driver. If you don't specify a driver, this is the type of network you are creating. Bridge networks are usually used when your applications run in standalone containers that need to communicate.
ii	Host	For standalone containers, remove network isolation between the container and the Docker host, and use the host's networking directly
iii	Overlay	Overlay networks connect multiple Docker daemons together and enable swarm services to communicate with each other. You can also use overlay networks to facilitate communication between a swarm service and a standalone container, or between two standalone containers on different Docker daemons. This strategy removes the need to do OS-level routing between these containers.
iv	ipvlan	IPvlan networks give users total control over both IPv4 and IPv6 addressing. The VLAN driver builds on top of that in giving operators complete control of layer 2 VLAN tagging and even IPvlan L3 routing for users interested in underlay network integration.
v	macvlan	Macvlan networks allow you to assign a MAC address to a container, making it appear as a physical device on your network. The Docker daemon routes traffic to containers by their MAC addresses. Using the macvlan driver is sometimes the best choice when dealing with legacy applications that expect to be directly connected to the physical network, rather than routed through the Docker host's network stack.
vi	none	For this container, disable all networking. Usually used in conjunction with a custom network driver. none is not available for swarm services.
v	Network plugins	For this container, disable all networking. Usually used in conjunction with a custom network driver. none is not available for swarm services.

Table 7: Docker Network Drivers

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 12 Issue 05, May-2023

Unencrypted traffic [Table 8 (i)] is considered as a security issue in a network and docker networks are no different. No plain text traffic should be used to transmit data across the network as any compromised container or malicious actor can intercept the data leading to data disclosure issues. This can be avoided by using encrypted traffic and legacy protocols like FTP should be avoided to lower the risk.

Insufficient Cryptography [Table 8 (ii)] is also a general network issue which should be taken care in docker networks. For example, using HTTPS in place of HTTP can make the connection secure but using weak cipher suites to encrypt the traffic for the web application can give an opportunity to attackers to decrypt the traffic leading to data disclosure. Weak Cipher suites should always be avoided while encrypting the data.

Unorganized docker networks [Table 8 (iii)] can also lead to security issues. Containers should be joined to networks which are organized or categorized properly. Containers having high sensitivity of the application or data should be grouped in separate network and containers having low sensitivity should be grouped separately. If containers are not categorized properly and a container having low sensitivity gets compromised the attacker will get access to containers having high sensitivity data. Grouping of containers also help in monitoring the network and prioritising the remediation of any identified issues.

S No.	Attack Vector	Scenario	Recommendations
i	Unencrypted Traffic	Container is running a web application on port 80 using plain text traffic. An attacker or a container with malicious intent running in same network sniffs the data in motion.	All the data in motion from containers to other containers or to the internet should always be encrypted.
ii	Insufficient Cryptography	Container is hosting a web application running with HTTPS but using weak cipher suites. A n attacker or a container with malicious intent running in same network decrypts the data in motion.	Use of weak cipher suites should always be avoided.
iii	Unorganized Docker Networks	Critical and non-critical containers are running in same network. A compromised container in same network can interact across the network	Containers should be grouped as per the criticality of the application and data in rest or motion.

Table 8: Attack Vectors on Docker Network

6. ORCHESTRATORS

Container orchestration tools gives features to manage container life cycle. There are multiple orchestration tools available but the most popular among all of them are Kubernetes, Docker Swarm and Apache Mesos.

They all provide similar features like container deployment, rollouts, exposing services to other containers or internet and load balancing. The tools can also self-restart or replace any container that doesn't meet the defined requirements. [Table 9] describes the possible vulnerabilities on an orchestration tool.

Unauthorized Access [Table 9 (i)] to the orchestration framework should be avoided and login should be protected via MFA which gives added security to strong passwords. In case, an attacker gets the credentials of any authorized personnel can replay the credentials and get access to the orchestration environment. AWS role and policies can be created to maintain unauthorized access and maintain MFA.

Poorly separated inter-container traffic [Table 9 (ii)] can bring risks to the environment like DoS or data disclosure. A malicious or compromised container can interact with other containers in network, monitor the traffic or craft DoS attacks to other containers running sensitive applications. To mitigate the risk all containers should be grouped according to application sensitivity level and the data present in the containers.

Giving **Administrative access** [Table 9 (iii)] to everyone or the person who is not intended to perform admin activities is always considered as a bad practice. This is not a

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 12 Issue 05, May-2023

vulnerability in itself but can be the root cause of security risks to the environment. Least privilege model should be followed and administrative access should always be given to people who are responsible to perform administrative activities.

Orchestrator node trust [Table 9 (iv)] should be maintained throughout the lifecycle of any node. If not taken in to account, then unintended hosts can join the cluster and run malicious containers. The orchestration frameworks provide feature to maintain the trust across the clusters so that no unauthorized host can join the cluster and making the environment secure from such risks.

S No.	Attack Vector	Scenario	Recommendations
i	Unauthorized access	No MFA(Multi-Factor Authentication) is present for orchestrator login. Leaked credentials of any user can give orchestrator access to attackers.	MFA should be introduced and made mandatory to login to the orchestrator. AWS role and policies can be created to maintain authorized access.
ii	Poorly separated inter-container traffic	There is no isolation of network and containers can interact to other containers which they are not supposed to. A compromised container in network can craft request to other container or monitor the traffic.	Containers should be separated into groups as per the sensitivity level of the applications running inside any containers.
iii	Administrative access to everyone	The Test team has admin access to production nodes. A malicious actor or the test team gets admin access to production nodes.	Least privilege model should be followed and the permission should always be given as per requirement of the team.
iv	Orchestrator node trust	No trust management is maintained in orchestration tool. Unauthorized host can join the cluster and run malicious containers.	Orchestration tool should provide feature to maintain trust. All nodes should be securely introduced into the infrastructure and identity should be persistent throughout the lifecycle of each node.

Table 9: Attack Vectors on Orchestration Framework

CONCLUSION

Docker provide a very better way to manage multiple applications or micro-services where containers can be started, paused, scaled and managed to maintain the supply chain that is the main reason of containers getting popular across the teams. Docker gives many benefits over traditional virtual machines such as running containers with very les start-stop time, light weight in structure. But as other technologies, vulnerabilities can also be present in docker or containers which should be taken care of.

As per my research basically containers have six layers in architecture where vulnerabilities can appear as mentioned below.

1. Host Operating System
2. Container runtime
3. Docker registry
4. Docker image
5. Docker network
6. Orchestration framework

Considering above layers, if proper security posture and configurations are maintained then the supply chain can be considered as fairly secure to host any application in production environment. And of course, security testing of the application code should always be done according to the application lifecycle because securing whole supply chain environment and hosting a vulnerable application inside can ruin whole effort of securing docker environment.

In addition to above mentioned layers of docker environment and making them secure, Linux Security Modules (LSMs) like SELinux or AppArmor can be introduce on the system running docker daemon to make the environment more secure and robust. [Table 10] describes few Linux Security Modules which can be introduced to the environment as per the requirements. Linux Security Modules gives features of hooking into check points for each operations and policies can be enforced which will be

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 12 Issue 05, May-2023

checked and can deny tasks, process to run which are not supposed to run as per the policies.

ACKNOWLEDGEMENT

First of all I would like to thank Olivier FLAUZAC for his great work in "A review of native container security for running applications" which inspired me as well to dive deep and look for possible threats on each layer. I would also like to thank SARI SULTAN for his work "Containers' Security: Issues, Challenges, and Road Ahead" which also helped me while doing the research.

At last, I would like to thank whole community for contributing and sharing their experience.

Name	Description
AppArmor	AppArmor is a Mandatory Access Control framework. When enabled, AppArmor confines programs according to a set of rules that specify what files a given program can access.
LoadPin	LoadPin is a Linux Security Module that ensures all kernel-loaded files (modules, firmware, etc) all originate from the same filesystem.
SELinux	Security-Enhanced Linux (SELinux) is a security architecture for Linux systems that allows administrators to have more control over who can access the system.
Smack	Smack is a kernel based implementation of mandatory access control that includes simplicity in its primary design goals.
TOMOYO	TOMOYO is a name-based MAC extension (LSM module) for the Linux kernel. It allows processes to declare resources and behaviours required to perform activities.
YAMA	Yama is a Linux Security Module that collects system-wide DAC security protections that are not handled by the core kernel itself. This is selectable at build-time with CONFIG_SECURITY_YAMA, and can be controlled at run-time through sysctls in /proc/sys/kernel/yama

Table 10: Linux Security Modules (LSMs)

REFERENCES

[1] Olivier FLAUZAC, Fabien MAUHOURAT, Florent NOLOTA "A review of native container security for running applications" (2020)
[2] SARI SULTAN, "Containers' Security: Issues, Challenges, and Road Ahead" (2019)
[3] Kelly Brady, Seung Moon, Tuan Nguyen, Joel Coffman, "Docker Container Security in Cloud Computing" (2020)
[4] Xin Lin, Lingguang Lei, Yuewu Wang, Jiwu Jing, Kun Sun, Quan Zhou, "A Measurement Study on Linux Container Security: A acks and Countermeasures" (2018)
[5] https://github.com/OWASP/Docker-Security
[6] https://www.alertlogic.com/blog/top-docker-security-vulnerabilities-best-practices-insights/
[7] https://madhuakula.com/content/attacking-and-auditing-docker-containers-and-kubernetes-clusters/attacking-private-registry/scenario.html
[8] https://docs.docker.com/config/containers/container-networking/
[9] https://en.wikipedia.org/wiki/Linux_Security_Modules
[10] https://www.sdxcentral.com/cloud/containers/definitions/how-does-container-networking-work/
[11] https://blog.container-solutions.com/linux-capabilities-why-they-exist-and-how-they-work
[12] https://0xn3va.gitbook.io/cheat-sheets/container/escaping/sensitive-mounts
[13] https://resources.infosecinstitute.com/topic/common-container-misconfigurations-and-how-to-prevent-them/
[14] https://docs.docker.com/network/
[15] https://www.optiv.com/insights/discover/blog/orchestrator-risks
[16] https://www.kernel.org/doc/html/v4.14/admin-guide/LSM/LoadPin.html
[17] https://apparmor.net/
[18] https://www.redhat.com/en/topics/linux/what-is-selinux
[19] https://www.kernel.org/doc/html/v4.18/admin-guide/LSM/Smack.html
[20] https://www.kernel.org/doc/html/v4.16/admin-guide/LSM/tomoyo.html