

## Report on container security

#Introduction:- 1.

What is container?

A container is a standard unit of software that packages up code and all its dependencies so the application runs quickly and reliably from one computing environment to another.

Containers and virtual machines have similar resource isolation and allocation benefits, but function differently because containers virtualize the operating system instead of hardware. Containers are more portable and efficient.

OS-level virtualization is an operating system (OS) paradigm in which the kernel permits many separate user space instances, known as containers, to exist.

### 2. Types of container

There are two main types of containers:

- a. Operating system containers: These containers provide a complete operating system environment in which applications can be run. Operating system containers include all the necessary libraries and dependencies, as well as a full copy of the operating system.
- b. Application containers: These containers include only the necessary libraries and dependencies for a specific application, and do not include a full copy of the operating system. Application containers are generally smaller and more lightweight than operating system containers, as they do not include the overhead of a full operating system.

### 3. What is container security?

As security threats and opportunities to tamper with organizations escalates, it's increasingly important for organization to assess their system's attack surface to identify all possible points of vulnerability.

Container Security is a critical part of a comprehensive security assessment. It is the practice of protecting containerized applications from potential risk using a combination of security tools and policies.

Container Security manages risks throughout the environment, including all aspects of the software supply chain or CI/CD pipeline, infrastructure, and container runtime, and lifecycle management applications that run on containers. When implementing solutions for container network security.

### #Container security Risks:-

Containers can introduce security risks, just like any other software. Here are a few potential security issues to consider when using containers:

**Vulnerabilities in the container image:** If the container image contains known vulnerabilities, attackers may be able to exploit them to gain access to the container or the host system. It is important to keep container images up to date and to use images from trusted sources.

**Shared host operating system:** Containers share the host operating system, so vulnerabilities in the host operating system can affect all containers running on the host. It is important to keep the host operating system up to date and to apply security patches as needed.

**Networking:** Containers use networking to communicate with other containers and with external services. It is important to properly configure networking to ensure that only authorized traffic is allowed in and out of the container.

**Resource constraints:** Containers may be limited in the resources (such as CPU and memory) that they are allowed to use. Attackers may try to exploit these limits to launch denial of service attacks or to gain access to the container.

### #Essentials of Container Security:-

**Configuration:** Many container, orchestration, and cloud platforms offer robust security capabilities and controls. However, they must be set up correctly and re-tuned over time to be fully optimized. This configuration includes critical settings and hardening in areas such as access/privilege, isolation, and networking.

**Automation:** Because of the highly dynamic and distributed nature of most containerized applications and their underlying infrastructure, security needs such as vulnerability scanning and anomaly detection can become virtually insurmountable when done manually. This is why automation is a key feature of many container security tools—much like how container orchestration helps automate a lot of the operational overhead involved in running containers at scale.

**Container security solutions:** Some teams will add new purpose-built security tools and support to the mix that are specific to containerized environments. Such tools are sometimes focused on different aspects of the cloud-native ecosystem, such as CI tools, container runtime security, and Kubernetes. Automating as many manual processes as possible through Kubernetes and similar open source technologies will help to detect issues in real-time and keep your organization more secure.

Cloud & Network Security: Network and container security are often discussed in tandem since containers use networks to communicate with each other. But cloud security extends further, including networks, containers, servers, apps, and the broader environment—all of which are interconnected, and thus dependent on one another to remain protected. Addressing cloud vulnerabilities must be a priority for every organization.

### #Security Hardening (Process to secure containers)

Hardening is the process of strengthening the security of a system or application by reducing its vulnerabilities and minimizing the attack surface. Here are a few best practices for hardening container security:

#### 1. Keep Host and Docker up to date:-

To prevent from known, container escapes vulnerabilities, which typically end in escalating to root/administrator privileges, patching Docker Engine and Docker Machine is crucial.

In addition, containers (unlike in virtual machines) share the kernel with the host, therefore kernel exploits executed inside the container will directly hit host kernel. For example, kernel privilege escalation exploit (like Dirty COW) executed inside a well-insulated container will result in root access in a host.

Command: - docker ps -a

#### 2. Set a user:-

Configuring the container to use an unprivileged user is the best way to prevent privilege escalation attacks. This can be accomplished in three different ways as follows:

```
FROM ubuntu

LABEL maintainer="jay"

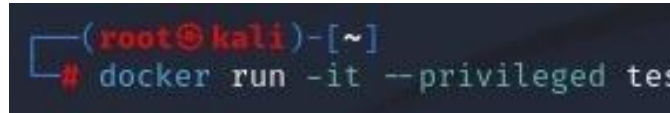
RUN groupadd -r jay && useradd -r -g jay
RUN echo "root:root" | chpasswd

#Environment Variables
ENV HOME /home/jay
ENV DEBIAN_FRONTEND=noninteractive
```

#### 3. Limit capabilities

Linux kernel capabilities are a set of privileges that can be used by privileged. Docker, by default, runs with only a subset of capabilities. You can change it and drop some capabilities (using --cap-drop) to harden your docker containers, or add some capabilities (using --

capadd) if needed. Remember not to run containers with the --privileged flag - this will add ALL Linux kernel capabilities to the container.



```
(root@kali)-[~]  
# docker run -it --privileged tes
```

#### 4. Add --no-new-privileges flag:-

Always run your docker images with --security-opt=no-new-privileges in order to prevent escalate privileges using setuid or setgid binaries.

Command: docker container\_name --security-opt=no-new-privileges

#### 5. Disable inter-container communication (--icc=false):-

By default inter-container communication (icc) is enabled - it means that all containers can talk with each other (using docker0 bridged network). This can be disabled by running docker daemon with flag. If icc is disabled (icc=false) it is required to tell which containers can communicate using --link=CONTAINER\_NAME\_or\_ID:ALIAS option. See more in Docker documentation - container communication

Command:- run with this command --icc=false

#### 6. Set filesystem and volumes to read-only

Run containers with a read-only filesystem using --read-only flag. For example:

Command: docker run --read-only alpine sh -c 'echo "whatever" > /tmp'

#### 7. - Use static analysis tools

To detect containers with known vulnerabilities - scan images using static analysis tools.

##### -Free Tools

Clair

ThreatMapper

Trivy

##### -Commercial

Snyk (open source and free option available) anchore

(open source and free option available)

Aqua Security's MicroScanner (free option available for rate-limited number of scans)

JFrog XRay

Qualys

#### 8. Set the logging level to at least INFO

By default, the Docker daemon is configured to have a base logging level of 'info', and if this is not the case: set the Docker daemon log level to 'info'. Rationale: Setting up an appropriate log level, configures the Docker daemon to log events that you would want to review later. A base log level of 'info' and above would capture all logs except the debug logs. Until and unless required, you should not run docker daemon at the 'debug' log level.

To configure the log level in docker-compose:

Command: `docker-compose --log-level info up`