Container Security Checklist: From the image to the workload

Table Of Contents

Cloud Native challenges

Legacy apps	Cloud Native apps	Cloud Native Security
Discrete, infrequent releases	frequent releases, using CI/CD	Shifting left with automated testing
Very little open source	Open source everywhere	SCA - Software composition analysis
Proprietary software	Proprietary code, Open source, Third-party software	Software supply chain risk
Persistent workloads	Ephemeral workloads. Ensure that your containers are stateless and immutable	Runtime controls that follow the workload
Hypervisor or hardware isolation	Shared kernel, obscured OS	Enforce least privilege on each

		workload
Permanent address	Orchestrated containers. Kubernetes creates DNS records for services and pods	Identity-based segmentation
Vertical control of the stack	multi-cloud	Detecting cloud services misconfigurations (CSPM)
Networking monitoring and threat detection tools were based on auditd, syslog, dead-disk forensics, and it used to get the full contents of network packets to disk "packet captures". Capturing packets sotes every packet in a network to disk and runs custom pattern matching on each packet to identify an attack.	Cloud native apps the traffic is encrypted. Packet captures are too costly and ineffective for cloud native environments.	Using eBPF programs, you collect the events in real time without disruption to the app.

> Table by Aqua Cloud Native Security Platform, more details download here

Container Threat Model

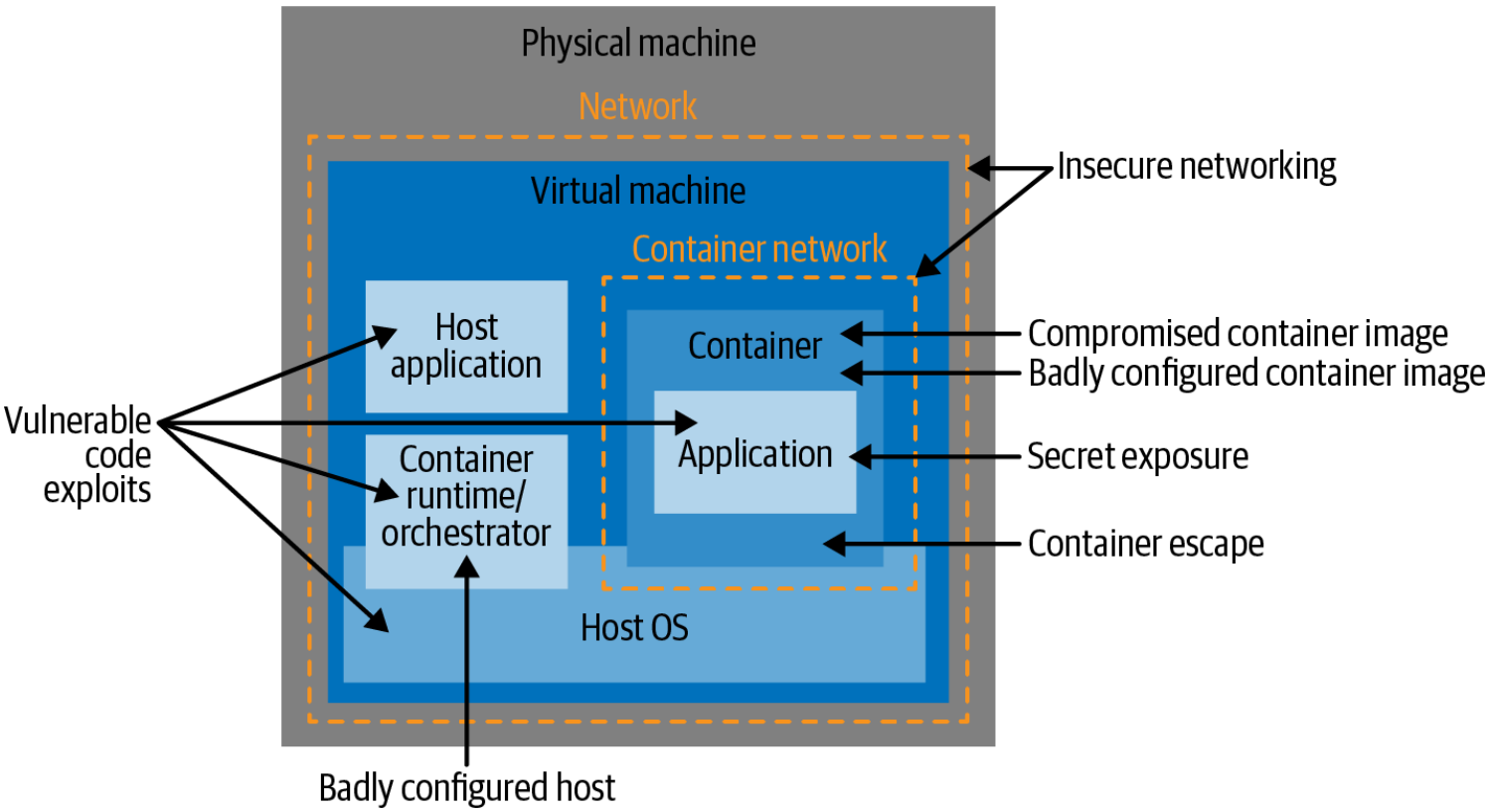

Figure by Container Security by Liz Rice

- Insecure Host
- Misconfiguration container
- Vulnerable application
- Supply chain attacks
- Expose secrets
- Insecure networking
- Integrity and confidentiality of OS images
- Container escape vulnerabilities

Container Security Checklist

Checklist to build and secure the images across the following phases:

- Secure the Build
- Secure the Container Registry
- Secure the Container Runtime
- Secure the Infrastructure
- Secure the Data
- Secure the Workloads

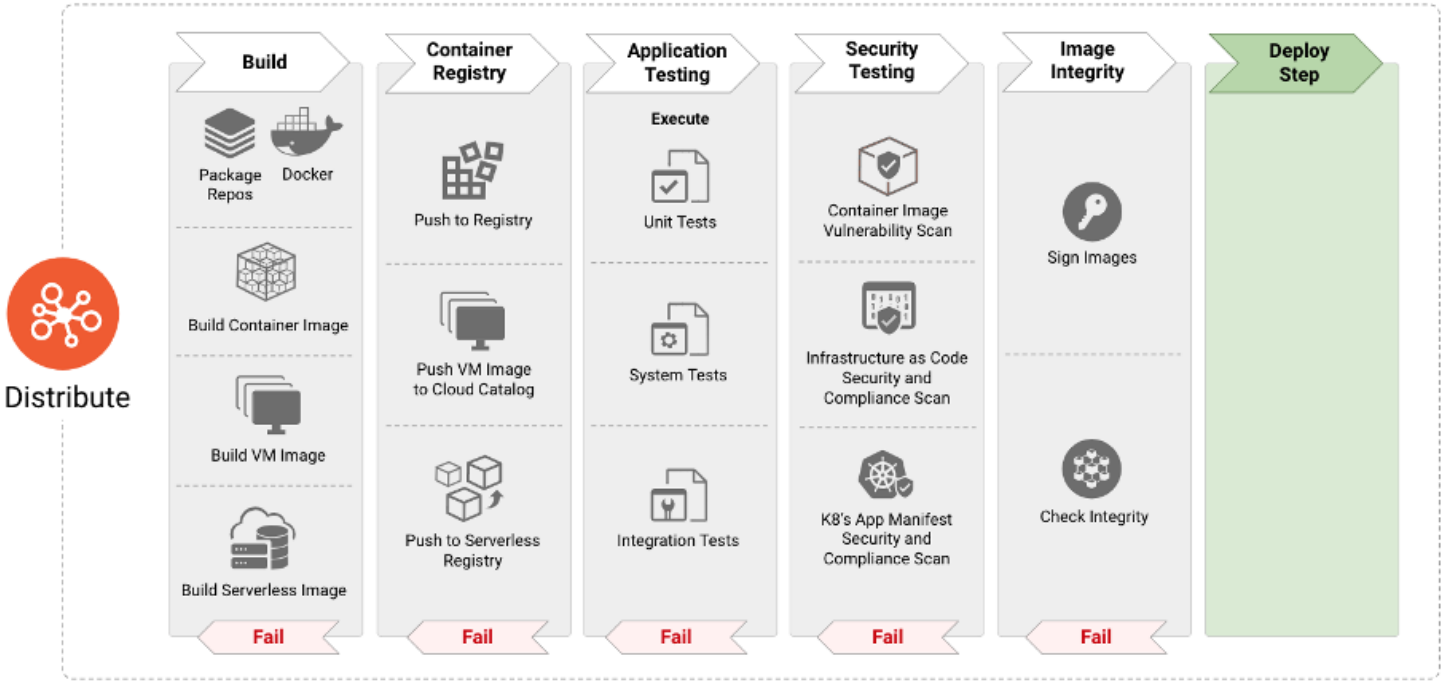

Figure by cncf/tag-security

Supply Chain Security

- Enforce image trust with Image signing: Container Signing, Verification and Storage in an OCI registry.

Image Signing

Sign and verify images to mitigate a man in the middle (MITM) attack. Docker offers a Content Trust mechanism that allows you to cryptographically sign images using a private key. This guarantees the image, and its tags, have not been modified.

- Notary. Implementation of TUF specification.
- sigstore/Cosign
- Sigstore: A Solution to Software Supply Chain Security
- Zero-Trust supply chains with Sigstore and SPIFFE/SPIRE

Secure the Build

Hardening Code - Secure SDLC (Software Development Life Cycle)

- [x] Do a static analysis of the code and libraries used by the code to surface any vulnerabilities in the code and its dependencies.

    - Improve the security and quality of their code. OWASP Open Source Application Security tools: SAST, IAST.

Secure the Image - Hardening

You can build the container images using Docker, Kaniko.

- *Reduce the attack surface*

> Package a single application per container. Small container images. Minimize the number of layers.

- Use the minimal OS image:
    - Alpine images
    - Scratch images
    - Distroless images
- Use OS optimized for running containers:
    - Flatcar images
    - CodeOS by Fedora replaced the Project Atomic.
    - Bottlerocket by Aws
    - k3os by Rancher
    - Container-Optimized OS - COS by Google, based on Chromium-os used by Google

> - Do you use Alpine, distroless or vanilla images? ...
> - 7 Google best practices for building containers

- Multi-staged builds.

> A well-designed multi-stage build contains only the minimal binary files and dependencies required for the final image, with no build tools or intermediate files. Optimize cache.

- [x] Use official base images.
    - Avoid unknown public images.
- [x] Rootless. Run as a non-root user. Least privileged user
- [x] Create a dedicated user and group on the image.

> Do not use a UID below 10,000. For best security, always run your processes as a UID above 10,000. Remove setuid and setgid permissions from the images

- [x] Avoid privileged containers, which lets a container run as root on the local machine.
- [x] Use only the necessary Privileged Capabilities.
    - Drop kernel modules, system time, trace processes (CAP_SYS_MODULE,

CAP_SYS_TIME, CAP_SYS_PTRACE).

- [x] Enable the `--read-only` mode in docker, if it's possible.
- [x] Don't leave sensitive information (secrets, tokens, keys, etc) in the image.
- [x] Not mounting Host Path.
- [x] Use Metadata Labels for Images, such as licensing information, sources, names of authors, and relation of containers to projects or components.
- [x] Used fixed image tag for inmutability.
  - Tagging using semantic versioning.
  - Not use mutable tags(latest,staging,etc). Use Inmutable tags(SHA-256, commit).
  - The challengue of uniquely identifying your images

```
Pulling images by digest
docker images --digests
docker pull alpine@sha256:b7233dafbed64e3738630b69382a8b231726aa1014ccaabc1947c53
```

- [x] Enabled Security profiles: SELinux, AppArmor, Seccomp.
- [x] Static code analysis tool for Dockerfile like a linter. **Detect misconfigurations**
  - Hadolint
  - Packers (including encrypters), and downloaders are all able to evade static scanning by, for example, encrypting binary code that is only executed in memory, making the malware active only in runtime.
  - Trivy detect missconfigurations

Image Scanning

- [x] Scan image for Common Vulnerabilities and Exposures (CVE)
- [x] Check image for secrets
- [x] Prevent attacks using the Supply Chain Attack
- [x] Used dynamic analysis techniques for containers to prevent runtime bad behaviours.

Container Security Scanners

- Trivy by AquaSecurity
- Clair by Quay
- Anchore
- Dagda
- GitGuardian Shield

Comparing the container scanners results:

- Container Vulnerability Scanning Fun by Rory
- Comparison – Anchore Engine vs Clair vs Trivy by Alfredo Pardo

More Material about build containers

- Azure best practices for build containers
- Docker best practices for build containers
- Google best practices for build containers

Secure the Container Registry

Best configurations with ECR, ACR, Harbor, etc. Best practices.

- [x] Lock down access to the image registry (who can push/pull) to restrict which users can upload and download images from it. Uses Role Based Access Control (RBAC)

> There is no guarantee that the image you are pulling from the registry is trusted. It may unintentionally contain security vulnerabilities, or may have intentionally been replaced with an image compromised by attackers.

- [x] Use a private registry deployed behind firewall, to reduce the risk of tampering.

Registry Resources

- Azure ACR
- Azure best practices for Azure Container Registry
- Amazon ECR
- Google Artifact Registry
- Harbor

Secure the Container Runtime

See the following container runtimes, there are three main types of container runtimes—low-level runtimes, high-level runtimes, and virtualized runtimes or sandboxed.

1. Low-Level Container Runtimes:

- runC
- crun

- containerd

2. High-Level Container Runtimes

- Docker Engine
- Podman
- CRI-O - OCI-based implementation of Kubernetes Container Runtime Interface
- Mirantes Container Runtime

3. Sandboxed and Virtualized Container Runtimes

- gVisor
- nabla-containers
- kata-containers

Why is important Runtime Security?

- Detection of IOC (Indicator Of Compromise)
- Detect Zero Days attack
- Compliance requirement
- Recommended in highly dynamic environments

Constraints

- Event context
- Safety
- Low overhead
- Wide support of kernels

Enable detection of anomalous behaviour in applications.

- [x] Applied the secure configurations in the container runtime. By default is insecure.
- [x] Restrict access to container runtime daemon/APIs
- [x] Use if it's possible in Rootless Mode.

Docker Security

- [x] Avoid misconfigured exposed Docker API Ports, attackers used the misconfigured port to deploy and run a malicious image that contained malware that was specifically designed to evade static scanning.
- [x] TLS encryption between the Docker client and daemon. Do not expose the Docker

engine using Unix socket or remotely using http.

> Never make the daemon socket available for remote connections, unless you are using Docker's encrypted HTTPS socket, which supports authentication.

- [x] Limit the usage of mount Docker socket in a container in an untrusted environment.
- [x] Do not run Docker images with an option that exposes the socket in the container.

```
-v /var/run/docker.sock://var/run/docker.sock
```

> The Docker daemon socket is a Unix network socket that facilitates communication with the Docker API. By default, this socket is owned by the root user. If anyone else obtains access to the socket, they will have permissions equivalent to root access to the host.

- [x] Run Docker in Rootless Mode. `docker context use rootless`
- [x] Enable the user namespaces.
- [x] Enable Docker Content Trust. Docker. `DOCKER_CONTENT_TRUST=1` . Docker Content Trust implements The Update Framework (TUF) . Powered by Notary, an open-source TUF-client and server that can operate over arbitrary trusted collections of data.
- [x] Do not run Docker without the default **seccomp profile**: `seccomp=unconfined`

> - Seccomp enabled by default. See the Docker profile here
> - Hardening Docker and Kubernetes with seccomp

More Material about Docker Security

- Docker Security Labs
- CIS Docker Bench.
- Content trust in Docker
- Docker Security Playground - DSP - Network Security and Penetration Test techniques

Secure the Infrastructure

Risk:

- If host is compromised, the container will be too.
- Kernel exploits

Best practices:

- [x] Keep the host kernel patched to prevent a range of known vulnerabilities, many of which can result in container escape. Since the kernel is shared by the container and the host, the kernel exploits when an attacker manages to run on a container can directly affect the host.
- [x] Use CIS-Benchmark for the operating system.
- [x] Use secure computing (seccomp) to restrict host system call (syscall) access from containers.
- [x] Use Security-Enhanced Linux (SELinux) to further isolate containers.

Secure the Data

- [x] Don't leak sensitive info in the images, avoid using environment variables for your sensitive information.

> Secrets are Digital credentials:
>
> - passwords
> - API keys & Tokens
> - SSH keys
> - Private certificates for secure communication, transmitting and receiving data (TLS, SSL, and so on)
> - Private encryption keys for systems like PGP
> - Database names or connection strings.
> - Sensitive configuration settings (email address, usernames, debug flags, etc.)

- [x] Use a proper filesystem encryption technology for container storage
- [x] Use volume mounts to pass secrets to a container at runtime
- [x] Provide write/execute access only to the containers that need to modify the data in a specific host filesystem path
- [x] OPA to write controls like only allowing Read-only Root Filesystem access, listing allowed host filesystem paths to mount, and listing allowed Flex volume drivers.
- [x] Automatically scan container images for sensitive data such as credentials, tokens, SSH keys, TLS certificates, database names or connection strings and so on, before pushing them to a container registry (can be done locally and in CI).
- [x] Limit storage related syscalls and capabilities to prevent runtime privilege escalation.

- [x] Implement RBAC, or role-based access control. Every human or application only needs the minimum secrets required to operate, nothing more. **Principle of Least Privilege**.
- [x] Run audits regularly. Centralized audit trails are the key to knowing all the key security events.
- [x] Rotate secrets, a standard security practice.
- [x] Automatically create and store secrets

Secrets Management Tools

Open source tools:

- detect-secrets by Yelp: detecting and preventing secrets in code.
- git-secrets by awslabs: Prevents you from committing secrets and credentials into git repositories

Cloud Provider Key Management

- AWS Secrets Manager
- Azure Key Vault
- Google Secret Manager

Enterprise secrets vault:

- HashiCorp Vault
- CyberArk Conjur

Secure the Workloads... Running the containers

- [x] Avoid privileged containers

  • Root access to all devices • Ability to tamper with Linux security modules like AppArmor and SELinux • Ability to install a new instance of the Docker platform, using the host's kernel capabilities, and run Docker within Docker.

> To check if the container is running in privileged mode `docker inspect --format ='{{.HostConfig.Privileged}}'[container_id]`

- [x] Limit container resources.

> When a container is compromised, attackers may try to make use of the underlying host resources to perform malicious activity. Set memory and CPU usage limits to minimize the

> impact of breaches for resource-intensive containers.

```
docker run -d --name container-1 --cpuset-cpus 0 --cpu-shares 768 cpu-stress
```

- [x] Preventing a fork bomb. `docker run --rm -it --pids-limit 200 debian:jessie`
- [x] Segregate container networks.
    - The default bridge network exists on all Docker hosts—if you do not specify a different network, new containers automatically connect to it.
    - Use custom bridge networks to control which containers can communicate between them, and to enable automatic DNS resolution from container name to IP address.
    - Ensure that containers can connect to each other only if absolutely necessary, and avoid connecting sensitive containers to public-facing networks.
    - Docker provides network drivers that let you create your own bridge network, overlay network, or macvlan network. If you need more control, you can create a Docker network plugin.
- [x] Improve container isolation.

> Protecting a container is exactly the same as protecting any process running on Linux. Ideally, the operating system on a container host should protect the host kernel from container escapes, and prevent mutual influence between containers.

- [x] Set filesystem and volumes to Read only.

> This can prevent malicious activity such as deploying malware on the container or modifying configuration. `docker run --read-only alpine`

- [x] Complete lifecycle management restrict system calls from Within Containers
- [x] Monitor Container Activity. Analyze collected events to detect suspicious behaviourial patterns.
- [x] Create an incident response process to ensure rapid response in the case of an attack.
- [x] Apply automated patching
- [x] Confirms Inmutability. Implement drift prevention to ensure container inmutability.
- [x] Ensure you have robust auditing and forensics for quick troubleshooting and compliance reporting.

Common Containers Attacks

- Unsecured Docker daemons
  - Cetus: Cryptojacking Worm Targeting Docker Daemons
  - Black-T: New Cryptojacking Variant from TeamTNT

Container Security Guides

- SP 800-190 - Application Container Security Guide by NIST

Further reading:

- Linux Capabilities: making them work, published in hernel.org 2008.
- Using seccomp to limit the kernel attack surface
- Docker Security Best Practices by Rani Osnat - AquaSecurity
- Applying devsecops in a Golang app with trivy-github-actions by Daniel Pacak - AquaSecurity

Collaborate

If you find any typos, errors, outdated resources; or if you have a different point of view. Please open a pull request or contact me.

Pull requests and stars are always welcome 🙌