Container Security

Presented by:

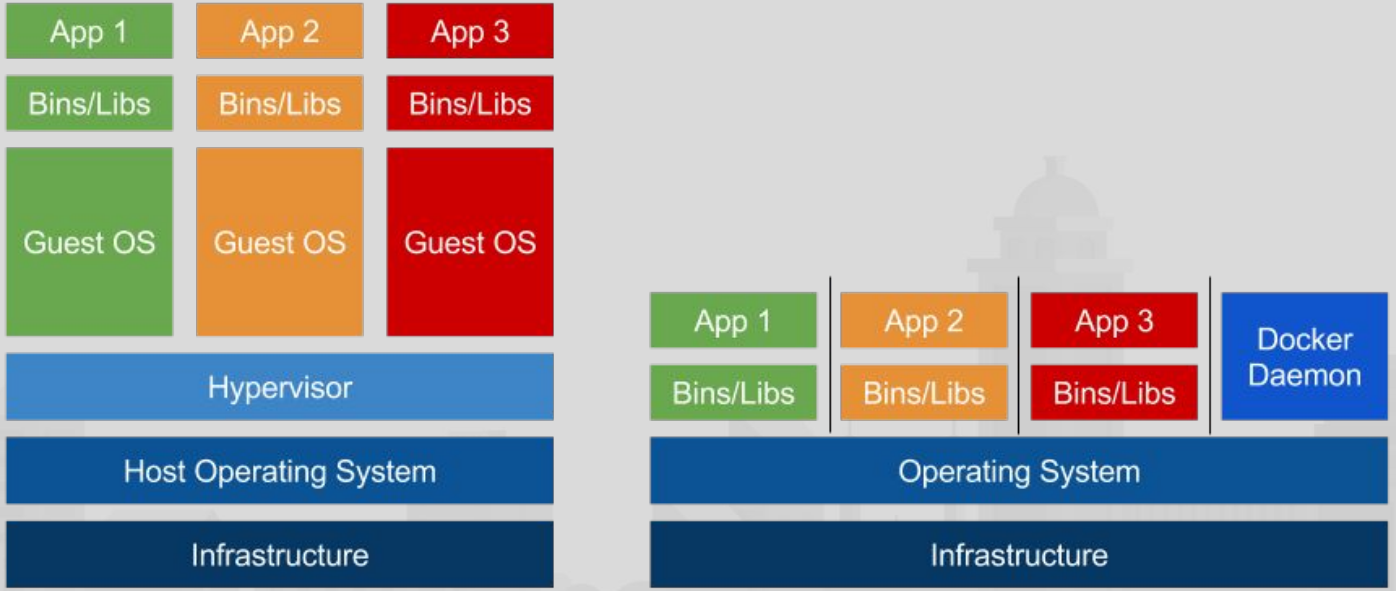
Kasra Motamedi (Security Engineer)
Amir Moghaddam (Security Engineer)

Spring 1401

What Are Containers?

- Resolving Dependency Nightmare
- Build once, run anywhere
- Simple, fast, and transparent way to set up runtime environments
- Easy for developers and operations to understand and use
- Smaller runtime footprint than VMs, fast deploy and startup
- Ideal for microservices

Containers vs. VMs



Container Advantages (Security Perspective)

- Runtime Separation of applications **on a host**
- Only deploy what you need: limited dependencies and smaller attack surface
- Easier to set up and tear down instances makes it easier to patch
- Increases consistency
  - making hardening, auditing, and testing easier
- Docker runtime has “sane OS defaults”: limited capabilities, etc.

Container Security Issues

- Docker Daemon presents its own attack surface
- Managing secrets with Docker
- Lightweight isolation
- Process running as root in container has root in underlying host
- Untrusted content
  - compromised and vulnerable images
- Container sprawl, especially at scale, and ephemeral runtimes are difficult to track and manage using classic network defenses (WAFs, IDS/IPS…)

Docker Components

- Host/Docker Daemon
- Registry
- Dockerfile
- Image
- Container

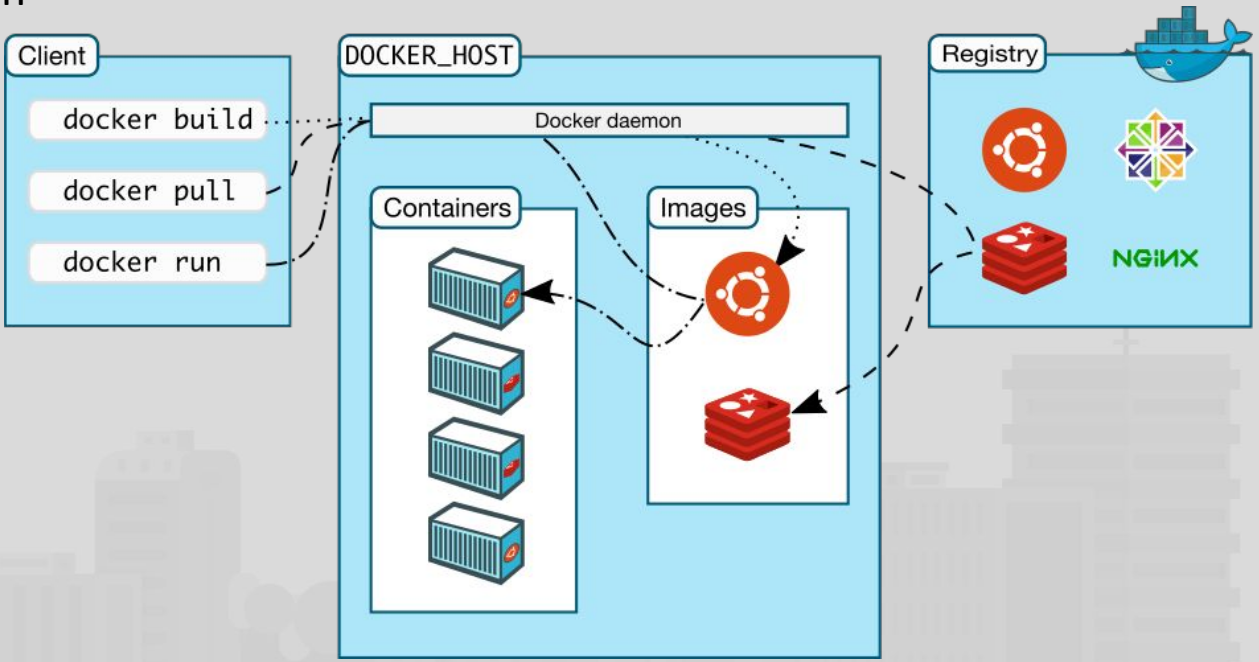
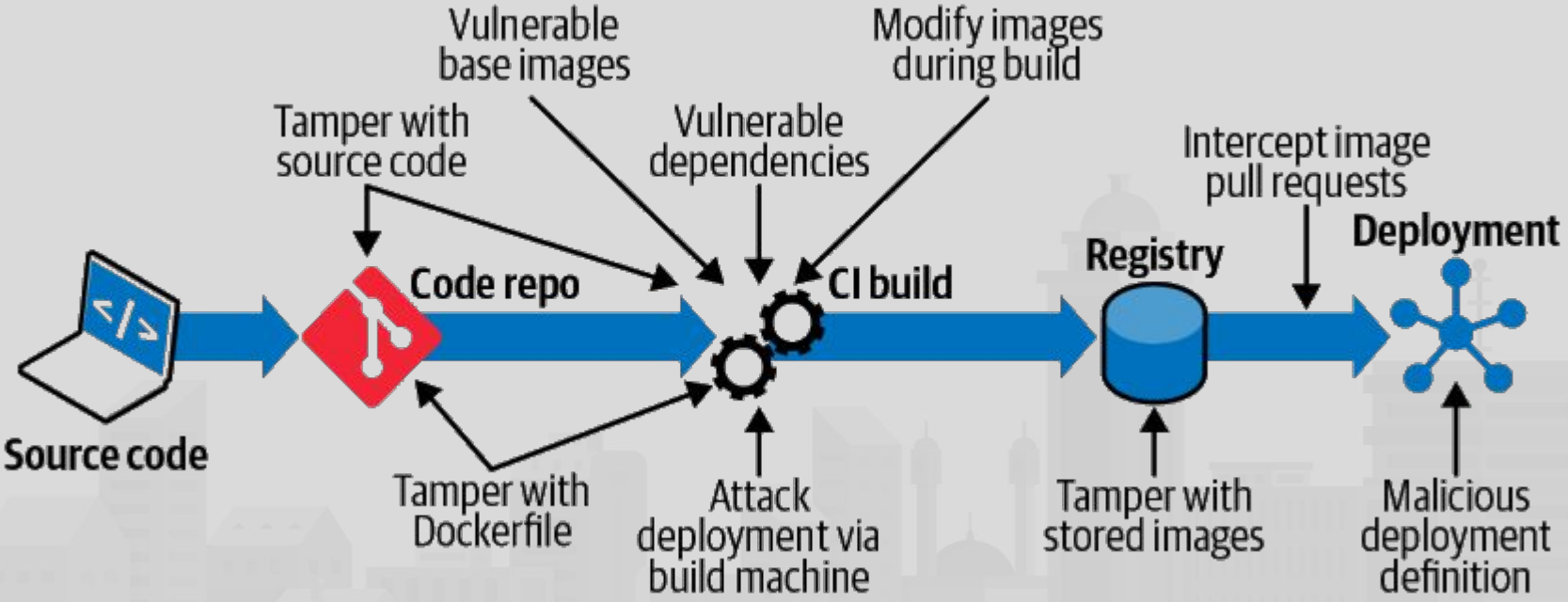
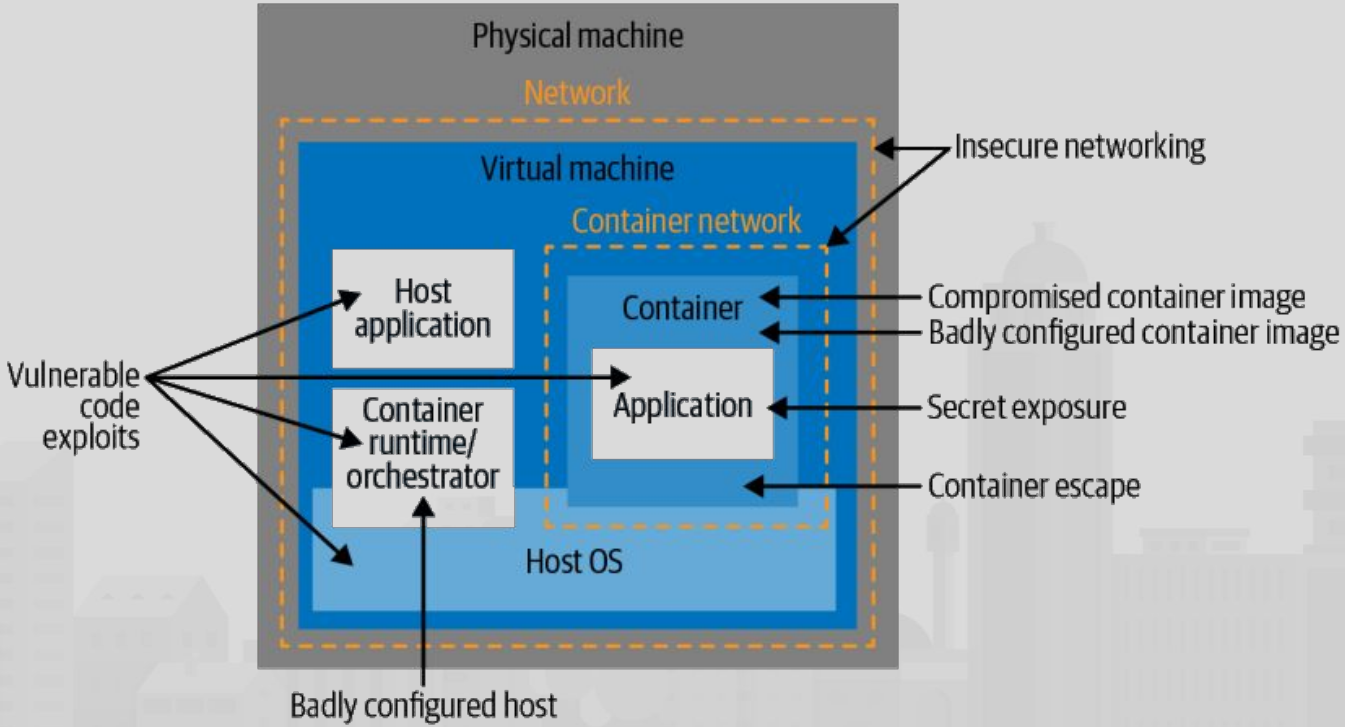


Image Attack Vectors



Container Attack Vectors



Host/Docker Daemon Security

- Monitoring host vulnerabilities
- Checking members of docker group
- Set correct permissions for docker files
- Track and patch vulnerabilities in Docker engine
- Enabling docker content trust (DCT)
  - Export DOCKER_CONTENT_TRUST=1
  - Image tags are digitally signed before pushing to a registry
  - Enforce using digitally signed images (push and pull)
- Enable auditing docker files
- Tool → Docker Bench Security [demo-link]
  - Scanning docker daemon + images + containers for best practices

Host/Docker Daemon Security (Authentication/Authorization)

- Do not expose the daemon API/socket unless you enable mutual TLS and user authentication
- If you have socket access to Docker Daemon, you have host root [demo-link]
- Authorization plugin framework implemented in Docker 1.10 so that you can write your own checks/rules
- Tool: Twistlock AuthZ Broker

Registry Security

- Using private docker registry
  - Preventing access to app codes
  - Part of CI/CD process
  - Faster pull/push
- Changing default registry port
- Enable TLS support and authentication

Image Security (1)

- Images from public registries can contain vulnerabilities or malware
- Use official images from public repositories
- Keeping images up to date
  - Protecting against vulnerabilities
  - Microsoft fixed vulnerabilities 2 months before WannaCry attack
- Scanning images regularly
  - Detecting vulnerabilities

Image Security (2)

- Signing images
  - Integrity check
- Don’t use setuid
  - Set the least capabilities
- Tool → Trivy [demo-link]
  - Auditing images for best practices
- Tool → Dive [demo-link]
  - Analyzing each image layer contents
  - Searching files in each layer for forensic purposes

Dockerfile Best Practices (1)

- Use base image from trusted sources
- Use scratch images
  - Reduce the attack surface
- Copy only what is needed
- Use non root user account
- Manage secrets with Swarm or Kubernetes if you are running in clusters
- Docker build secrets (--secrets) switch released in 18.09
- Best practice is to use a general-purpose secret keeper like Vault
- Don’t use secrets in Dockerfile [demo-link]

Dockerfile Best Practices (2)

- Set least privileges for non root users
  - If it doesn't need, only set read and (or) execute for non root users
- Don't run unnecessary services inside the Dockerfile
  - Reduce the attack surface
- Use multi stage builds
  - Reduce the attack surface
- Tool → Trivy [demo-link]
  - Auditing Dockerfiles for best practices

Container Security (Isolation+Limitation)

- Isolation through (primarily) kernel namespaces
- Vulnerabilities in host OS kernel or Docker Engine can break isolation
- Config mistakes can allow processes in containers to interact with other containers and with the host
- Use selective sharing of namespaces between containers
- Don’t use host Namespaces (e.x., --uid host)
- Use Cgroups
  - Resource limitation (cpu and memory)
  - Prevent DOS attacks

```
root@ubuntu:~# lsns -p 1
        NS TYPE   NPROCS PID USER COMMAND
4026531835 cgroup    350   1 root /sbin/init splash
4026531836 pid       339   1 root /sbin/init splash
4026531837 user      333   1 root /sbin/init splash
4026531838 uts       347   1 root /sbin/init splash
4026531839 ipc       345   1 root /sbin/init splash
4026531840 mnt       327   1 root /sbin/init splash
4026532008 net       334   1 root /sbin/init splash
root@ubuntu:~#
```

Container Security (User Namespace)

- Prevent using root user inside the containers
- Users are not fully namespaced in Docker
- User namespace remapping at runtime added in Docker 1.10
  - --userns-remap=[USERNAME]
  - NOT enabled by default

Container Security (Capabilities)

- Default limited capabilities in Docker → reducing the container attack service
- Dropping additional capabilities
  - Note: --cap-add can be used to increase privileges if required
  - Best practice is to drop all capabilities not required by the application
  - [demo-link]
- Watch out: --privileged argument allows a Docker container to access all devices on the host and overrides AppArmor/SELinux restrictions! This should never be allowed
  - [demo-link]
  - [demo-link]

Container Security (Capabilities)

- Capabilities
  - CAP_SYS_PTRACE
    - process injection → reverse shell
  - CAP_SYS_MODULE
    - kernel module injection → reverse shell
  - CAP_DAC_READ_SEARCH
    - read host files
  - CAP_DAC_OVERRIDE
    - read and write host files
  - CAP_SYS_TIME
    - set system clock

Container Security (Docker Hosts)

- Some host OS builds are designed specifically to support Docker or the containers, especially in the cloud
  - CoreOS
  - RancherOS
  - Snappy Ubuntu Core
  - VMware Photon OS

Container Security (Security Profiles/Policies)

- Seccomp
  - Secure Computing Mode (seccomp)
  - syscall firewall between user-level processes and the Linux Kernel
  - Support and a default profile is implemented as of Docker Engine 1.10
  - Whitelist or blacklist filters define that system calls and arguments are allowed
  - Default filter blocks over 50 syscalls when a container runs as non-privileged user
- AppArmor
  - Used for debian based distros
  - Blocking capabilities for programs
- SELinux
  - Used for redhat based distros
  - Defining access controls for apps, processes and files

Container Security (Other)

- Least privileges
  - Set the least capabilities if needed
- Do NOT pass secrets in runtime environment variables
- Mounting the container filesystem as read-only
- Not mounting sensitive host directories inside the containers
- Restricting privileges
  - --security-opt=no-new-privileges (disable setuid)
- Collect log files
  - /var/lib/docker/containers/*/*.log
  - /etc/falco/events.txt
- Tool → Falco [demo-link]
  - Detecting anomalous activity

Summary

- Containers **can** improve security over bare metal/VM runtimes
- Don’t depend on Docker defaults
  - Docker provides some built-in architectural sandboxing and other runtime protection, but is NOT secure by default
- Make appropriate trade-offs between developer flexibility and security
- Treat images as sensitive data
  - control and restrict access to images, review and scan image contents
- Major investments are being made in container security
  - expect improvements from Docker, the community, and vendors

Reference

- https://www.sans.org/cyber-security-courses/cloud-security-devsecops-automation/
- https://info.aquasec.com/container-security-book
- https://bootcamps.pentesteracademy.com/course/container-security-on-demand
- https://www.udemy.com/course/docker-security
- https://www.youtube.com/watch?v=b_euX_M82uI