Hacking and Securing Docker containers

Srinivasa Rao Kotipalli

Abhijeth Dugginapeddi

About US

Srinivasa Rao Kotipalli
Red Team Engineer
Author
Speaker
Trainer

@srini0x00

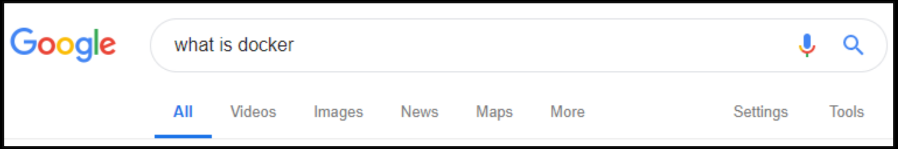
Abhijeth Dugginapeddi
Security lead
Mentor @wesecureapp
Adjunct Professor
Speaker

@abhijeth

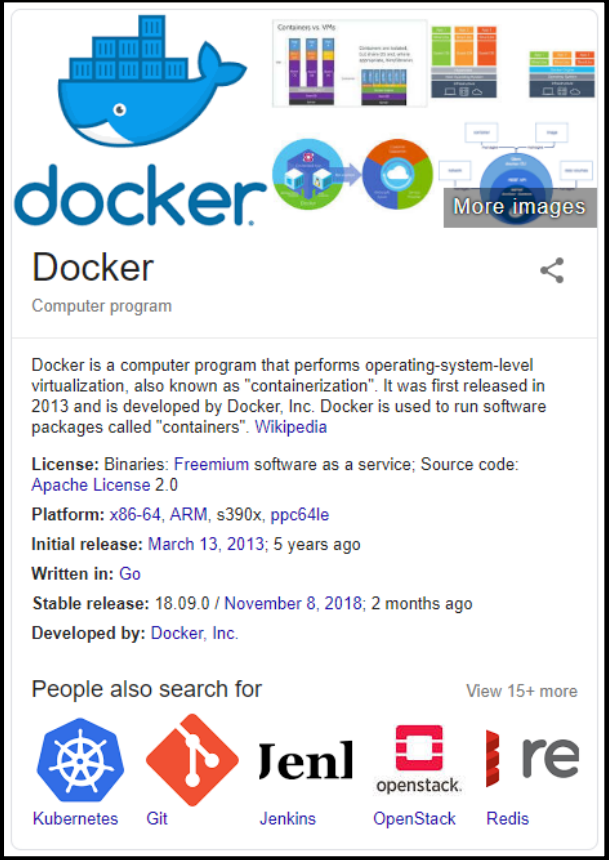
Agenda

- Introduction to the whys and hows of docker
- Hacking Docker Containers
- The magic of automation
- Defenses
- Questions

Docker Docker Docker?



what is docker



Docker

Computer program

Docker is a computer program that performs operating-system-level virtualization, also known as "containerization". It was first released in 2013 and is developed by Docker, Inc. Docker is used to run software packages called "containers". Wikipedia

License: Binaries: Freemium software as a service; Source code: Apache License 2.0

Platform: x86-64, ARM, s390x, ppc64le

Initial release: March 13, 2013; 5 years ago

Written in: Go

Stable release: 18.09.0 / November 8, 2018; 2 months ago

Developed by: Docker, Inc.

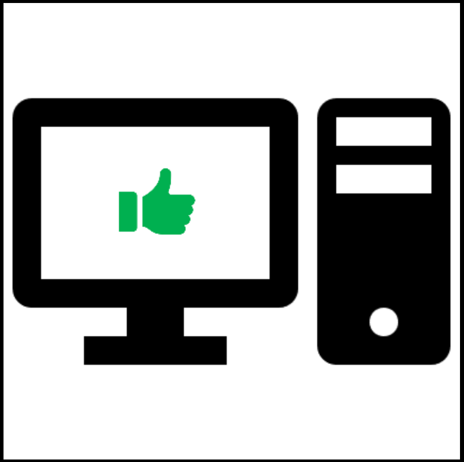
People also search for

View 15+ more

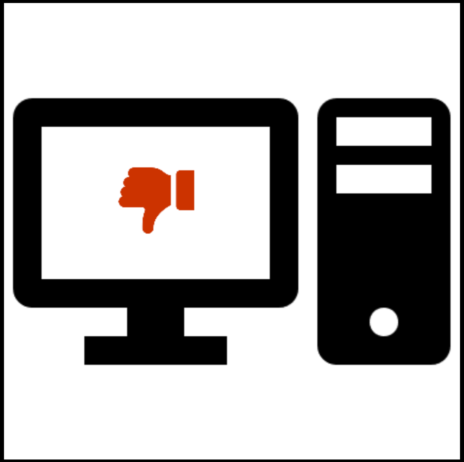
Kubernetes Git Jenkins OpenStack Redis

But why?

When same software is run on two different computers



DEVELOPER 1



DEVELOPER 2

Portability: Since a Docker container can easily be sent to another machine, you can set up everything on your own computer and then run the analyses on e.g. a more powerful machine.

Shareability: You can send the Docker container to anyone (who knows how to work with Docker).

Virtual Machines vs Containers

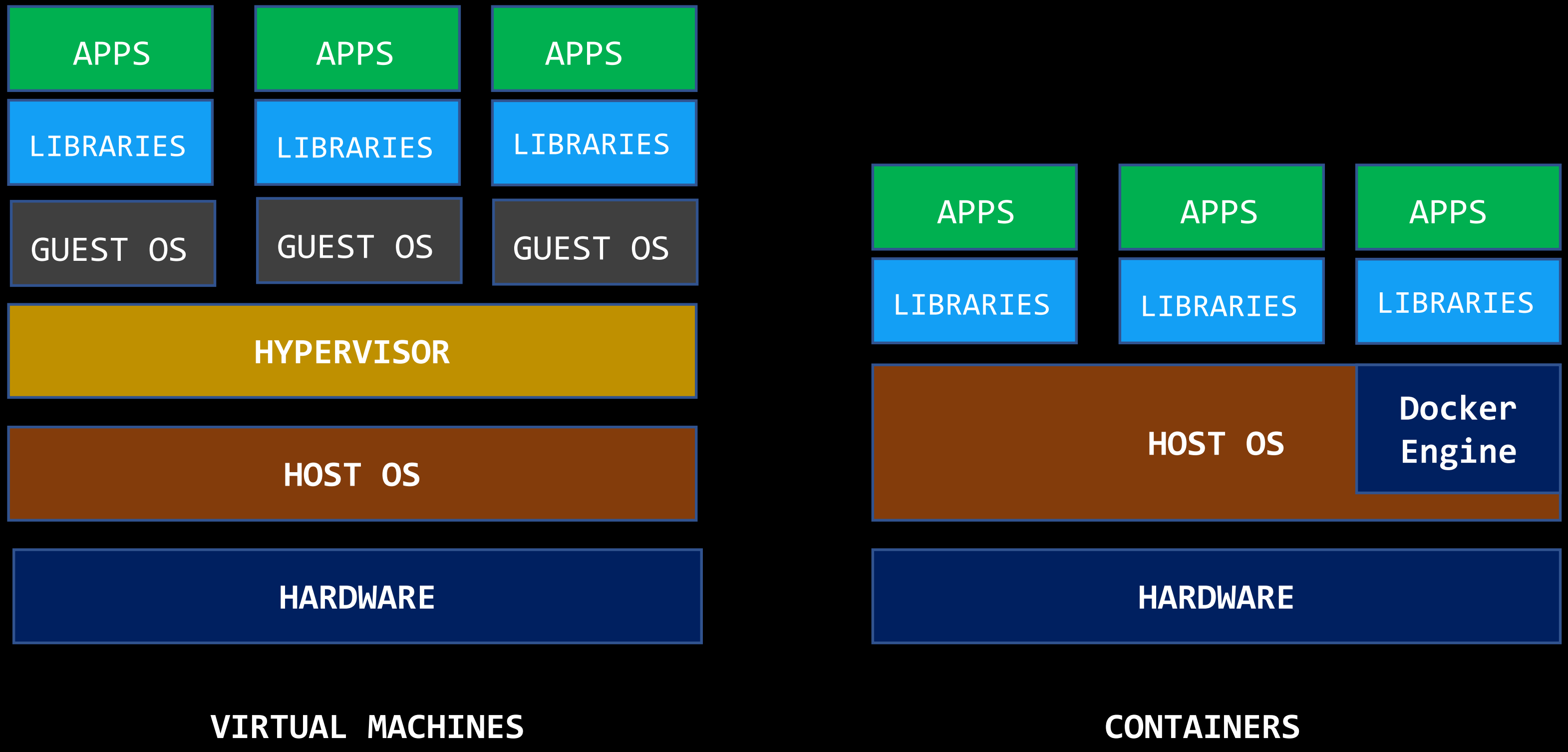
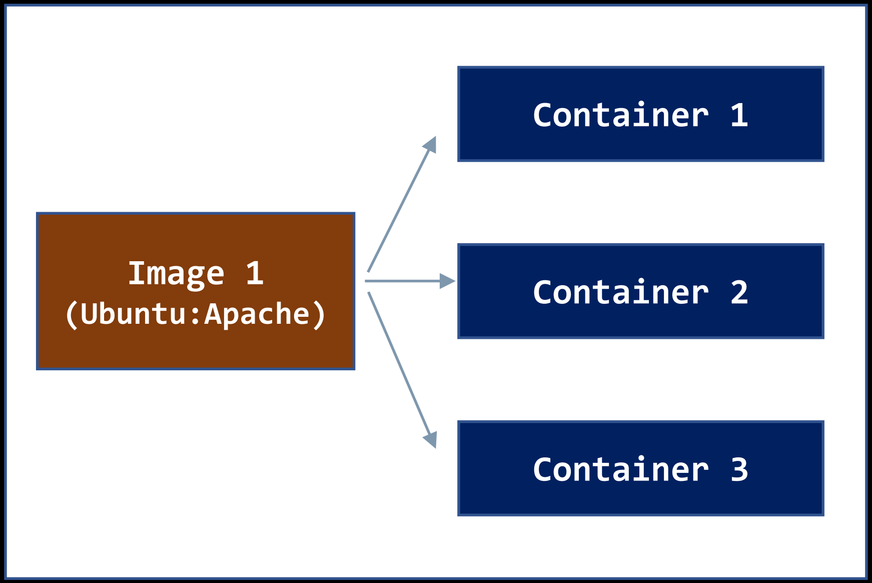


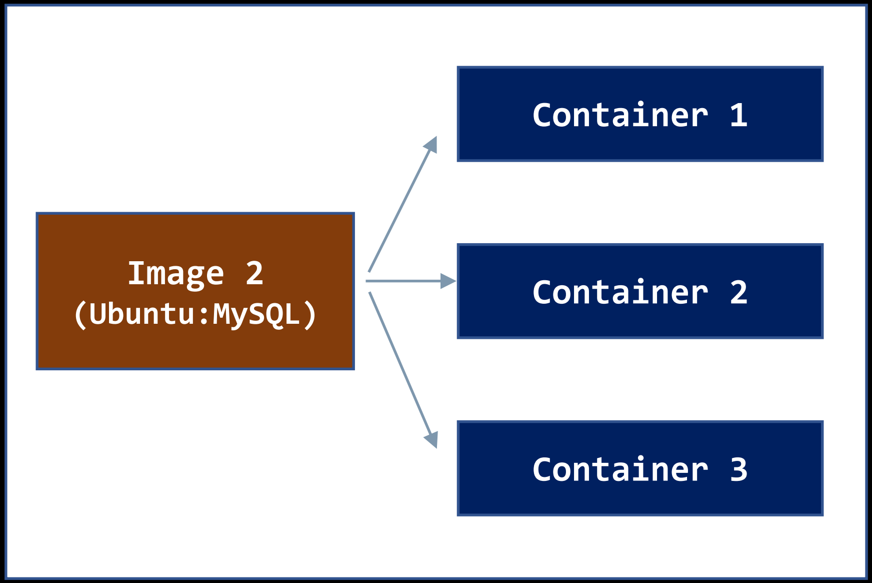
Image vs container

Docker container image is a lightweight, standalone, executable package of software that includes everything needed to run an application: code, runtime, system tools, system libraries and settings.

Docker container is an instance of an image. It 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

Image vs container





Let’s build your first docker image

cgroups

A Linux Kernel feature that allows you to limit the access that processes and containers have to system resources such as CPU, RAM, IOPS and network

Limits apps to specific set of resources

Enforce limits and constraints

One of the fundamental aspects of Docker Containers

Demo

About namespaces

Namespaces are one of the Linux Kernel features.

This is one of the fundamental aspects for containers on Linux.

Docker uses namespaces to provide isolation to the containers from host.

Docker Engine uses the following namespaces on Linux:

PID namespace for process isolation.
NET namespace for managing network interfaces.
IPC namespace for managing access to IPC resources.
MNT namespace for managing filesystem mount points.
UTS namespace for isolating kernel and version identifiers.
User ID(user) namespace for privilege isolation.

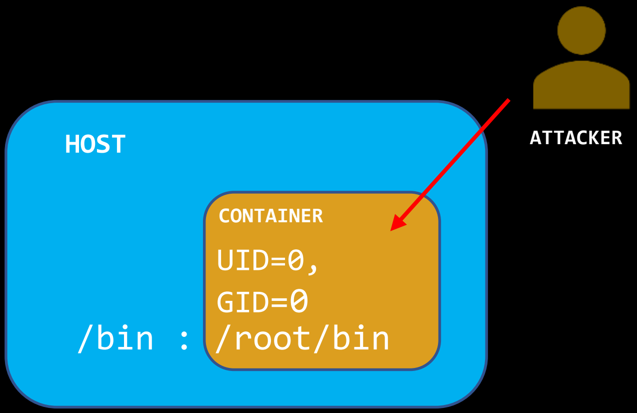
More about namespaces

Your application required root on the container

An attacker compromised this application and got root access on the container.

We mounted the **/bin** directory of the host on to the container.

Can someone modify files on the host's **/bin** directory?



Hint?

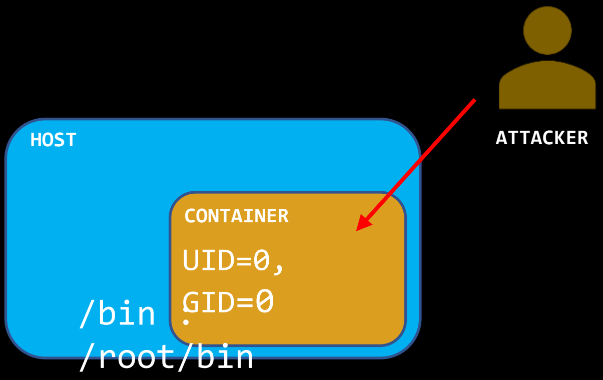
root inside the container = root on the host

User namespaces

If we enable user namespaces for docker daemon, it will ensure that:

Root inside the container is run in a separate context that is different from the host’s.

This will ensure that root on the container is not equivalent to root on the host.



Hint?

root inside the container != root on the host

Final.. namespaces

- Similarly, PID namespaces can also be used for Process ID isolation between container and host
- Processes running inside the container will have it’s own PID starting at 1 as the container will have it’s own PID namespace.
- This helps us providing additional layer of security as we cannot have visibility to the PID of the host.
- We can override this behavior by adding --pid host to the docker create or docker run commands. This will allocate container PIDs in the host’s PID namespace instead of container’s PID namespace.

Clean up

Unnecessary images and containers occupy a lot of space

Stopping and deleting all running containers:

```
1. Docker stop $(docker ps -aq)
2. Docker rm $(docker ps -aq)
```

Stopping and deleting a single container

```
1. Docker ps -a -q
2. Docker stop <container_id>
3. Docker rm <container_id>
```

Docker Registry

Docker registry is a system for distributing and storing Docker images

When we run *docker pull alpine:latest*

Docker Engine communicates with Docker's public registry - Docker hub

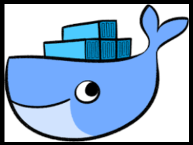
It is also possible to run your own private registry to limit what images can be downloaded by your Docker users

The current default docker registry can be seen using the command *docker info*

Hacking Docker Containers

Threat model

External Attacker



Malicious Insider

Compromised Containers

4GIFs.com
imgflip.com

Let’s dive deep

Vulnerable Images

Docker images are commonly downloaded from public repos

It is possible that these Docker images can have publicly known vulns
Ex: *shellshock, Dirtycow etc*

These vulnerabilities can potentially provide foot hold on your containers and the hosts where docker is being run when exploited

Check if you are inside a container

An attacker has exploited a remote code execution vulnerability

Is it a container shell or a shell from the host?

Look for docker artifacts when in doubt

```
cat /proc/self/cgroup
```

Container breakout

Existing exploits – outdated docker installations

Mounting host’s filesystem onto the containers

Over privileged capabilities – *CAP_SYS_ADMIN, CAP_SYS_MODULE*

Dangerous mountpoints – */var/run/docker.sock*

Backdooring existing docker images

Always download images from a trusted source

Malicious images are everywhere and they are always a danger

How do we create these malicious images from the original?

It can be done manually but using *dockerscan, it* is easier

Official page: *https://github.com/cr0hn/dockerscan*

Use *docker content trust* to ensure that images are verified

PrivEsc using volume mounts

Docker volumes are a way to provide persistent storage to Docker containers

Docker Daemon requires root privileges to perform some of it’s operations

If a user is part of Docker group, it is possible to elevate their privileges to root

~Demo

Credits: “Root you docker host in 10 seconds by electricmonk”

Container escape using docker.sock

Some container need */var/run/docker.sock* to be mounted in the running container

Required for interacting with containers on the host

Access to */var/run/docker.sock* is similar to ROOT

~Demo

--privileged flag

--privileged gives many capabilities to the container

An attacker who has access to the container can take advantage of these capabilities to escape the container and gain foothold on the host

Examples: CAP_SYS_MODULE, CAP_SYS_ADMIN

~Demo

Abusing CAP_SYS_MODULE

When a container is run with --privileged flag or specifically CAP_SYS_MODULE capability, an attacker who has access to the container can escape from the container by installing a kernel module directly on the host’s kernel.

Let us see how this can be used to get a reverse shell on the host where the privileged container is running.

~Demo

Dangling Volumes

Deleting a container will not delete the mounted volume

It is possible to have unused volumes on the host’s file system

This is a feature so that we can attach this volume to another new container

If sensitive data is pushed to these shard volumes and left unused after the container is terminated, it can be dangerous

~Demo

Exploiting Docker remote API

Unauthenticated Remote API = ROOT

Exploiting Docker remote API

Docker remote API allows us to interact with the Docker containers remotely using a REST API

When Docker remote API is configured to be exposed over the network, it exposes your Docker Daemon to anyone on the network.

By default no authentication is required

An attacker can remotely gain root access on the Docker host

~Demo

Accessing Docker Secrets

Secrets management is one of the challenges in Docker

It is often seen that secrets are kept in places such as environment variables

Let us see how Docker Secrets can be abused.

~Demo

Automated Vulnerability Assessments

Static analysis using Clair

Clair is an open source project for the static analysis of vulnerabilities in appc and docker containers

Vulnerability data is continuously imported from known set of sources and correlated with the indexed contents of container images in order to produce lists of vulnerabilities that threaten a container

Source: Clair website

~Demo

Docker Bench

Another popular tool for auditing docker environments

Comes as a container and easy to use.

Once the scan finishes, the output shows detailed information about what is right and what needs to be fixed.

https://github.com/docker/docker-bench-security

~Demo

Defenses

Apparmor & Seccomp profiles

AppArmor is a linux security module that can be used to protect Docker containers from Security threats

To use it with Docker, we need to associate an AppArmor security profile with each container

Docker expects to find AppArmor policy loaded and enforced

More info: *https://docs.docker.com/engine/security/apparmor/*

Seccomp is another Linux security feature, which acts as firewall for syscalls.

For example, you can block CHOWN syscall on a container by loading a seccomp module when starting the container

~Demo

Capabilities

- ROOT users in Linux are very special
- They have super powers – more privileges
- If you break all these super powers into distinct units, they become capabilities.
- Almost all the special powers associated with the Linux root user are broken down into individual capabilities
- Being able to break down these permissions allows you to:

Have granular control over controlling what root user can do - less powerful.

Provide more powers to standard user at a granular level.

Capabilities

By default, Docker drops all capabilities except those needed, using a whitelist approach.

You can use Docker commands to add or remove capabilities to or from the bounding set.

Examples of capabilities constants in Linux vs Docker

Linux	Docker
CAP_CHOWN	CHOWN
CAP_NET_ADMIN	NET_ADMIN
CAP_SETUID	SETUID
CAP_SYSADMIN	SYSADMIN

Wrap up, what did we learn TBC

- Basics of Docker – Building images, namespaces, cgroups etc
- Backdooring images, Privilege escalation and abusing Docker Misconfigurations
- Container break out techniques.
- Automated vulnerability assessment
- Hardening Docker container, AppArmor and Seccomp profiles

THANKS FOR LISTENING TO US