Glossary

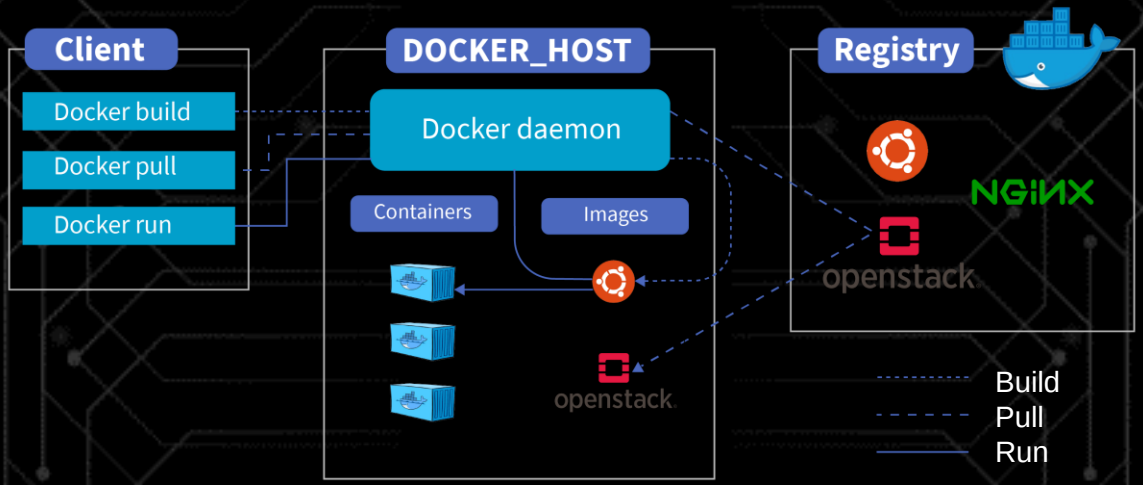
Switches Used

-u	= Specifies user
-it	= Interactive terminal
--rm	= End session
--read-only	= Gives Read only perms
--privileged	= Give extended privileges
-perm	= State permission
-t	= Gives a new tag/name
-exec	= Execute the command given
-v	= Specifies value, version and volume
ls -l	= List using long list format
-r	= Read the group name
-g	= Fetch the group name to add
--cap-drop	= Drop Linux capabilities
--cap-add	= Add Linux capabilities
groupadd	= Add additional groups to run as
useradd	= Add or Create users to run as
net_admin	= Performs network-related Operations

Abbreviations Used

sudo	= Super user do
CVE	= Common vulnerabilities & Exposures
SCAP	= Security Content Automation Protocol
N-Map	= Network Mapper
SSH	= Secure Shell
TLS	= Transport Layer Security
MITM	= Man in the Middle
ARP	= Address Resolution Protocol
MAC	= Media Access Control
HTTP	= Hyper Text Transfer Protocol
OS	= Operating System
CAP	= Linux Capabilities
API	= Application Program Interface
DCT	= Docker Content Trust
AuthN	= Authentication
AuthZ	= Authorization
ROI	= Return on Investment

Docker Architecture



→ **Client** = The primary way that provides you a terminal to interact with Docker.
→ **Host** = It is a compute instance which manages storage, networks, builds and many other services.
→ **Registry** = It is also known as Docker Hub that let you stores and distribute Docker images.
→ **Build** = This command builds Docker images from a Docker File.
→ **Pull** = It uses Registry to fetch a particular image or set of images.
→ **Run** = This command is used to run the built images present in the docker directory.

Use-Cases [P(#) represents Priority]

P(0) **Security** = Containers add a level of implicit isolation between system components, which naturally allows for a higher level of security, granting you complete control over traffic flow and management. No Docker container can look into processes running inside another container. From an architectural point of view, each container gets its own set of resources ranging from processing to network stacks.

P(1) **Rapid Deployment** = Docker manages to reduce deployment to seconds. This is due to the fact that it creates a container for every process and does not boot an OS. Data can be created and destroyed without worry that the cost to bring it up again would be higher than what is affordable.

P(2) **Multi Tenancy** = Docker allows us to host multiple instances (tenants) at the same time, which helps the developer to perform multiple tasks at once. Also, it will avoid users invading other users' containers, since the containers owned will be invisible to other tenants or users except the admin or the owner of the containers.

P(3) **Availability** = To start containers within milliseconds, in comparison to seconds or minutes in the case of virtual machines, makes a great difference to the availability of the applications in situations when hardware fails (which eventually, it will), the application crashes or an operator needs to reset it because, for example, it was modified by a hacker.

Solution Statement / Proposed Approach

Basically, we have come up with an approach to tackle the docker attacks and enhance it's security. The approach is known as HAND approach. Let us see how it works …

1) **Host Security** here refers to the base system on which the docker container is hosted or working. To avoid attacks like privilege escalation we need to harden the host security.

2) **Automated Tools** are an essential part of scanning as they detect vulnerabilities which are quite troublesome to be found manually.

3) **Network Security** helps to create our own safe and secure network which can be very much helpful to developers keeping their data in a secure environment.

4) **Docker Security** should be implemented to make secure images and containers to begin with.

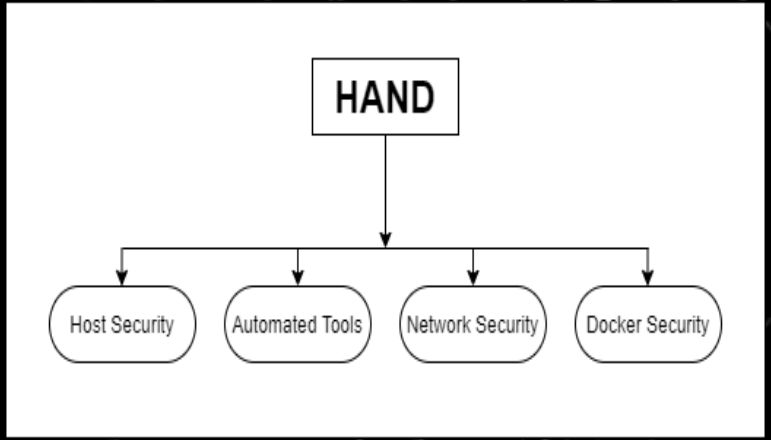


Fig. 1.1 Shows the divisions of HAND approach.

Host Security

→ Set Filesystem and Volumes to Read-Only

A simple and effective security trick is to run containers with a read-only filesystem. This can prevent malicious activity such as deploying malware on the container or modifying configuration.

```
docker run --read-only -u testuser -it --rm [container_id] /bin/bash
```

→ Never use switches like --privileged or user=root

Don’t use docker exec command with privileged or user=root option, since this setting could give the container extended Linux capabilities. This can also lead to a privilege escalation attack.

```
docker run --privileged
```

→Remove SETUID & SETGUID permissions from Images

It prevents them from being misused for the privilege escalation attacks. To find all executable files in a Docker image with these permissions run the command given below.

```
docker run <Image Name/ID> find / -perm +6000 -type f -exec ls -ld {} \; 2> /dev/null
```

→ Do not mount sensitive host system directories on containers

If sensitive directories are mounted in read-write mode, it would be possible to make changes to files within those sensitive directories. The changes might bring down the Docker host in compromised state. Below given example can actually shutdown your host machine.

```
$ docker run -ti -v /run/systemd:/run/systemd centos /bin/bash
[root@1aca7fe47882 /]# shutdown
```

Automated Tools

→ Anchore Engine

The Anchore Engine is an open-source project that provides a centralized service for inspection, analysis, and certification of container images. It provides you two modules known as Syft and Grype Vulnerability Scanner. To install it please go through the below given guide.

Link = https://engine.anchore.io/docs/quickstart/

→ App Armor

It is a Linux kernel security module that allows the system administrator to restrict program's capabilities with per-program profiles. It comes pre-installed in Ubuntu since version 7.10, so you may not be aware of what it is and what it's doing. If you are using any other linux distro, just type the command *"sudo apt install apparmor"* and you're good to go.

Link = https://gitlab.com/apparmor/apparmor/-/wikis/Documentation

→ Dagda

It is a tool to perform static analysis of known vulnerabilities, trojans, viruses, malware & other malicious threats in docker images/containers and to monitor the docker daemon and running docker containers for detecting anomalous activities.

Link = https://github.com/eliasgranderubio/dagda

→ CoreOS Clair

It is an open-source project which provides a tool to monitor the security of your containers through the static analysis of vulnerabilities in appc and docker containers. It inspects containers layer-by-layer based on CVEs and similar databases from Ubuntu, Debian and Red Hat. To install it refer to the below given repo.

Link = https://github.com/quay/clair

→ SELinux

SELinux is an acronym for Security-enhanced Linux. It is a security feature of the Linux kernel. It is designed to protect the server against misconfigurations and compromised daemons. It put limits and instructs server daemons or programs what files they can access and what actions they can take by defining a security policy.

Link = https://www.linode.com/docs/guides/how-to-install-selinux/

→ SCAP

It is a simple tool providing interface to use oscap in Docker environment. It allows you to scan running Docker containers and images almost in the same way as scan of local machine. It utilizes the Extensible Configuration Checklist Description Format (XCCDF).

Link = https://open-scap.org/resources/documentation/security-docker/

Network Security

→ Avoid mapping ports below 1024

Don’t map any ports below 1024 within a container as they are considered privileged because they transmit sensitive data. By default, Docker maps container ports to one that’s within the 49153 - 65525 range, but it allows the container to be mapped to a privileged port. As a general rule of thumb, ensure only needed ports are open on the container. Also, mapping any ports below 1024 makes them an easy target to be scanned using network-based attack tools such as N-Map. The attacker can easily know the service running on a certain port.

→ Containers should not run an SSH server

Don’t run ssh within containers. By default, the ssh daemon will not be running in a container, and you shouldn’t install the ssh daemon to simplify security management of the SSH server. The SSH server is pretty safe, but still, when a security issue arises, you will have to upgrade *all* the containers using SSH. That means rebuilding and restarting *all* of them. That also means that even if you need a pretty innocuous memcached service, you have to stay up-to-date with security advisories, because the attack surface of your container is suddenly much bigger.

→ Prefer using User-Defined Network

It is usually advised to make and use our own network bridge for a container. Using the default bridge “*docker0*” leaves us open to attacks like ARP Spoofing and MAC Flooding, which not only affect the target machine, but also pose a threat to all the devices/instances in a network. Furthermore, using user-defined network prevents containers from communicating with each other and considerably lowers the risk of supply-chain attacks on our machines.

Docker Security

→ Use trusted images as base image

Use minimal base images from a trusted source to ensure that there are no suspicious files crawling into our host. Install packages only from the trusted repositories. In addition to this, use checksums and digital signatures to verify the authenticity and integrity of downloaded packages. This will protect the packages from accidental corruption and MITM if the packages are downloaded over an insecure channel (HTTP).

```
#Enable Docker Content Trust
export DOCKER_CONTENT_TRUST=1

#Use Tags and Checksums
FROM alpine:3.11
FROM alpine@sha256:ddba4d27a7ffc3f86dd6c2f92041af252a1f23a8e742c90e6e1297bfa1bc0c45
```

→ Run Container in Rootless mode

Configuring a container to use an unprivileged user is the best way to prevent privilege escalation attacks. If you do not specify a user while starting a container, it will run by default as the root set in the image. You can add a specify user into the specific user group while creating an image.

```
#Include the code while building your Image

RUN groupadd -r groupname && useradd -r -g username groupname
```

→ Drop all Unnecessary Capabilities

When running containers, remove all capabilities not required for the container to function as needed. You can use Docker's CAP DROP capability to drop a specific container's capabilities (also called Linux capability), and use CAP ADD to add only those capabilities required for the proper functioning of the container. Let us see how to give unique capabilities to a user.

```
#Adding Network Admin Capability

docker run --cap-drop all --cap-add NET_ADMIN -it --rm -u username [container_id] /bin/bash
```

→ Do not store any secrets in DockerFile

It is advised not to keep any secrets or API keys in a docker image unless there is no way around. In which case, Secret Management Tool can be used to keep such confidential files/entities in encrypted form. Secrets in Dockerfile can easily be exposed by simple commands and once exposed can be exploited by an adversary.

```
#Check Docker History to find secrets in your logs by this simple command

docker history <Image Name/ID>
```

Conclusion

1) Container Security has significantly evolved from early days of docker.

2) Verify your dependencies and sign your artifacts with Docker Content Trust.

3) You must deal with the docker security infrastructure based on your use case.

4) Hardening Systems like AppArmor, SELinux and GRSEC provides an extra layer of safety.

5) Performing System Audits and Log Checks helps you keep the live status of container in track.

6) Containers are by default quite secure, especially if you run your processes as non-privileged users.

7) It has made Linux containerization technology accessible and easy-to-use, and, more importantly, manageable.

8) As always, staying up to date by doing system updates on time keeps common vulnerabilities away from the system.

Limitations

1) “*Root”* protection seems to be never ending problem.

2) Running automated tools can cause a lot of server utilization.

3) Neglecting even one part of the approach can increase the attack surface of the container.

4) Docker may provide useful defaults, but lacks support for fine-granular of AuthZ and AuthN.

5) Using minimal base images to start your work can consume a lot of time which can result in delay of project completion.

6) Sometimes, it becomes necessary to store credentials in the Docker File itself which increases the risk of sensitive information disclosure.

Future Scope

Containers are very lightweight and highly portable. This will enable organizations to deploy the same container on varied environments with ease.

Using Docker container platform, teams will be easily able to adopt practices like DevOps and Infrastructure As a Service.

Containers provide better utilization of hardware than virtualized machines. So, Organizations reap a higher ROI.

Using Docker containers, teams will be able to achieve Dev-Prod Parity and no longer face environment compatibility related issues.

Docker needs very few resources to run an application. Hence, organizations can save on everything from server costs to the employees needed to maintain them.