



KENNESAW STATE
UNIVERSITY

Module 5: Containers Security in Physical System

Dr. Maria Valero

Adapted from Gursimran Singh

Agenda

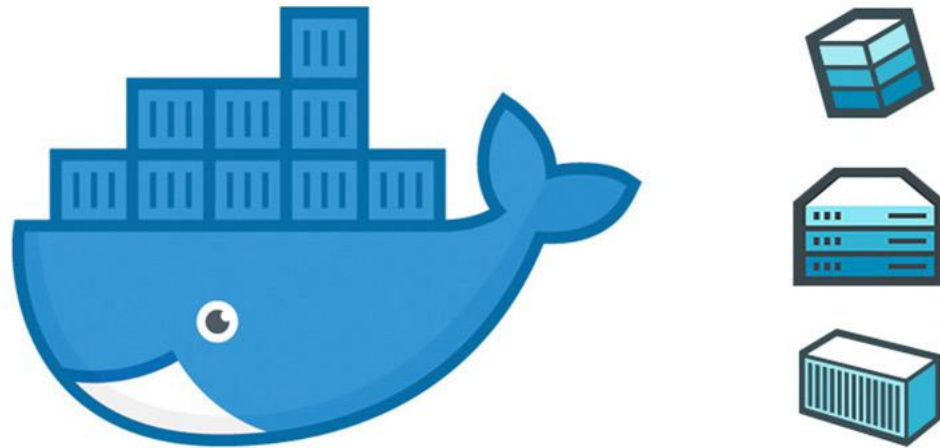
- Refreshing what is a container
- Benefits of enabling container security
- Container Security Mechanisms
- Guide to Container Security Best Practices
- Summarizing Container Security

A perspective view of a server room aisle. The racks on both sides are filled with server units, many of which have small, glowing lights (likely status LEDs) that create a starry pattern. The floor is light-colored with dark grates. The lighting is a cool blue/teal color, creating a high-tech atmosphere. The text "Containers Security" is overlaid in the center in a white, sans-serif font.

Containers Security

Refreshing: What is a container?

- Containers are based on an entirely isolated environment; they provide a solution to the problem of how to get the software to run reliably when migrating from one computing ecosystem to another computing ecosystem.



Benefits of enabling Container Security

- Allows development teams to move fast, deploy software efficiently
- Less overhead operations as Containers require fewer system resources.
- Applications operating in containers can be deployed quickly to different operating systems

Benefits (2)

Containers make apps portable - It looks the same everywhere.

- No matter where you run it.
- Doesn't need to install all the app dependencies.
- Containerization allows for greater modularity i.e., and the application can split into modules.



Security Mechanisms



1. Docker Image Provenance (1)

- The gold standard for image provenance is Docker Content Trust (DCT).
- DCT presents the capability to use digital signatures for data sent to and received from remote Docker registries.
- With DCT enabled, a digital signature is added to the image before they are pushed into the registry.
- When the image pulls DCT will verify the signature, by ensuring that the image comes from the correct organization and the content of the image exactly matched with the image that was pushed.

1. Docker Image Provenance (2)

- It is also possible to verify the image using digests.
- A digest is the sha256 hash of a docker image.
- When the image is pushed, a docker client will return a string that represents the digest of an image. Whenever the image is pulled, the docker will verify the digest matches with an image. Any update in the image will result in the generation of the new digest.

1. Docker Image Provenance (3)

```
$ docker pull nginx
```

```
nginx:latest: The image you are pulling has been verified
```

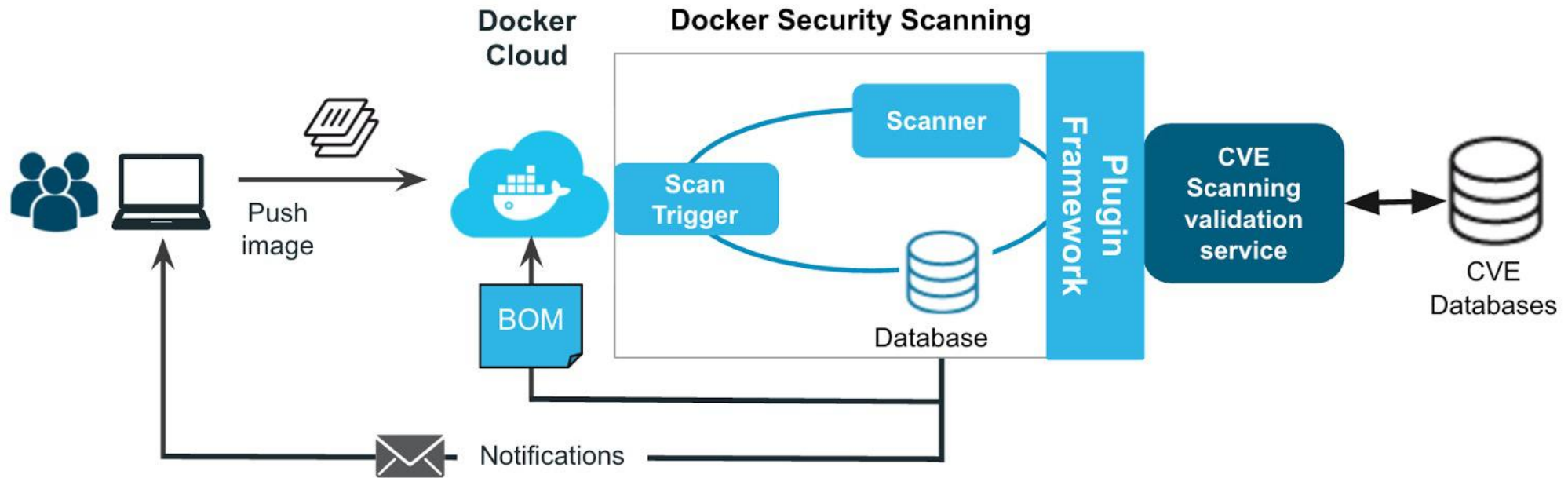
```
5a7d9470be44: Pull complete
```

```
feb755848a9a: Pull complete
```

2. Security Scanning (1)

- Docker security scanning gives the ability to do a binary level scan of all images.
- The image scanner automatically helps to identify all the vulnerabilities and reduce risk.
- It also ensures the integrity of a container image.
- Docker security scanning is available as an integral part of the docker cloud and docker data center but not as a stand-alone service.

2. Security Scanning (2)




3. Auditing (1)

- The production environment is regularly edited to ensure that all the containers are based on up-to-date containers, and both hosts and containers are securely configured.
- Auditing directly follows security scanning and image provenance.
- It isn't enough to scan image before they are deployed as new vulnerabilities are reported. Therefore, it is essential to scan all the images that are running.

3. Auditing (2)

- Some tools can be used to verify that the container files system has not diverged from the underlying file systems. Tools, such as docker diff is used to list the changed files and directories in a container file system since the container was created.

```
student@debian:~$ docker diff checkintegriy
A /data
A /data/output
C /root
A /root/.bash_history
C /.dockerenv
student@debian:~$
```



We can list the changed files and directories in a containers file system

There are 3 events that are listed in the diff

- A - Add
- D - Delete
- C - Change

4. Isolation and Least Privileged (1)

- The significant security benefit of the container is the extra tooling around isolation.
- Containers work by creating a system with separate namespaces.
- The principle of least privilege is defined as "Every program and privileged user of the system should function using the least amount of privilege required to complete the job"
- Concerning containers this represents that each container should run with minimal set of privileges possible for its effective operation.

4. Isolation and Least Privileged (2)

- A container can also be secured by running containers with read-only file systems.
- In docker, this is achieved by only passing the read-only flag to docker run.
- By passing the read-only flag, the attacker will be unable to write the malicious scripts to the file systems and unable to modify the contents.

5. Runtime Threat Detection (1)

- No matter how good a job is done with vulnerability scanning and container hardening. There are always unknown bugs and vulnerabilities that may recognize in runtime and cause a disturbance. That is why real-time threat detection is essential.
- Tools like AuqaSecurity and Twistlock offer runtime threat detection.
- Twistlock provides full-stack container and cloud-native cybersecurity for teams using Docker, Kubernetes, serverless, and other cloud-native technologies.

5. Runtime Threat Detection (2)

- Twistlock automatically learns the behavior of the images and microservices while preventing anything anomalous.

The screenshot displays the Twistlock Radar interface, which is used for monitoring and managing container security. The interface is divided into several sections:

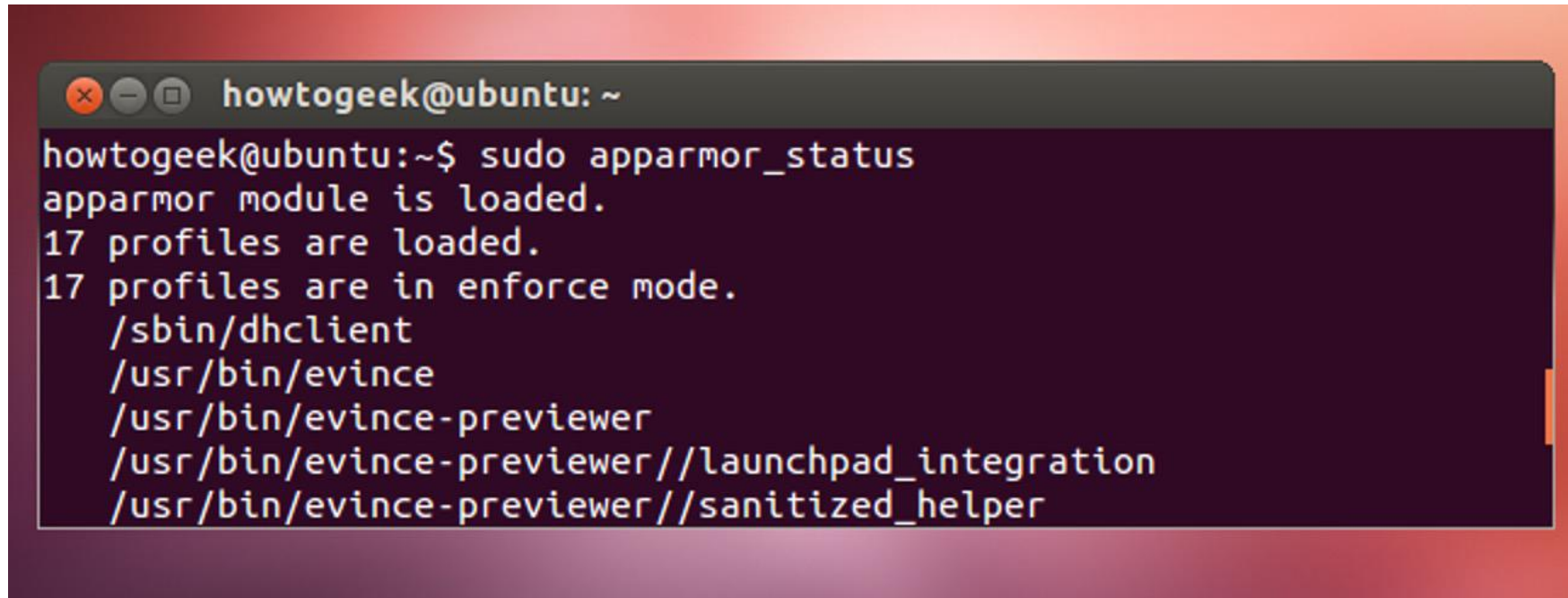
- Top Bar:** Shows 'Containers' (45), 'Learning' (0), and 'Vulnerabilities' (1104, 1505, 598, 91).
- Left Sidebar:** Contains navigation options for 'Defend' (Firewalls, Runtime, Vulnerabilities, Compliance, Access) and 'Monitor' (Firewalls, Runtime, Vulnerabilities, Compliance, Access), along with a 'Manage' section.
- Central Area:** Features a network diagram of containers. A detailed view for 'orders:0.4.7' is shown below, including:
 - Metadata:** Image (weaveworksdemos/orders:0.4.7), Namespace (sock-shop), Service (orders), Service IP (10.110.219.198), Image ID (8275c5b9181b2311), Label (orders-787bf5b89f), OS distro (Alpine Linux v3.4), and Host (demo-keith-lab-twistlock.com).
 - Runtime events:** Processes (0), Network (0), File System (0), System Calls (0).
 - Vulnerabilities:** 15 Critical risk, 20 High risk, 15 Medium risk, 2 Low risk.
 - Compliance:** Critical risk (0), High risk (0), Medium risk (0), Low risk (0).
 - Status:** DNS connected.
- Right Panel:** Includes 'Deployed Defenders' (2 Container Defenders, 0 Host Defenders, 0 Serverless Defenders), a 'Number of incidents' line graph for 'Last week' (Nov 19-24), and a 'Compliance - Vulnerabilities' section with 'Impacted images', 'Impacted containers', and 'Impacted hosts'.

6. Access Control (1)

- The most two standard security modules are SELinux and AppArmor.
- They both are an implementation of the Mandatory Access Control (MAC) mechanism.
- SELinux and AppArmor are brave attempts to clean up the security holes in Linux containers.
- MAC will check that the user or process has the right to perform various actions such as reading and writing.
- Application Armor (AppArmor) is an effective and easy to use Linux application security system. It protects the OS and applications from any kind of internal and external threat.

6. Access Control (2)

- AppArmor is available for docker containers and applications present in the containers.
- AppArmor is always recommended to use as is by default with Ubuntu 16.04.

A terminal window with a dark background and light text. The window title is 'howtogeek@ubuntu: ~'. The command 'sudo apparmor_status' has been executed, and the output is displayed. The output indicates that the AppArmor module is loaded, 17 profiles are loaded, and 17 profiles are in enforce mode. A list of profiles is shown, including /sbin/dhclient, /usr/bin/evince, /usr/bin/evince-previewer, /usr/bin/evince-previewer//launchpad_integration, and /usr/bin/evince-previewer//sanitized_helper.

```
howtogeek@ubuntu:~$ sudo apparmor_status
apparmor module is loaded.
17 profiles are loaded.
17 profiles are in enforce mode.
  /sbin/dhclient
  /usr/bin/evince
  /usr/bin/evince-previewer
  /usr/bin/evince-previewer//launchpad_integration
  /usr/bin/evince-previewer//sanitized_helper
```

7. Avoid Root Access (1)

- The namespace feature in Linux containers allows developers to avoid root access by giving isolated containers a separate user account.
- So, a user from one container does not have access to another container.
- System Administrator should have to enable this feature, as this feature is not enabled by default.

Guide to Container Security Best Practices (1)

- **Create Immutable Containers**
 - Immutable infrastructure is a paradigm in which dockers are never modified after they are deployed, i.e., they can be only rebuilt
- **Securing Images for Container Security**
 - Specify the list of trusted sources for the images and libraries
- **Securing Registries for Container Security**
 - Once the image is built and secured in the best way possible, so now the image must be stored in a registry. If the image is stored in a registry, one should scan them regularly for the vulnerabilities

Guide to Container Security Best Practices (2)

- **Run Images From Trusted Sources**
 - Building images from trusted sources can minimize the attack surface. While building images from trusted sources, there are still some chances that vulnerabilities can be present. Therefore, it is recommended to scan the content with the scanning tool.
- **Securing Deployment for Container Security**
 - The target environment needs to be secure, i.e., the operating system should be appropriately hardened on which containers are running. If deployed to cloud environments, one should consider immutable deployments

Guide to Container Security Best Practices (3)

- **Keeping Containers Lightweight**
 - While running containers, it is possible to load too many packages. Therefore, lightweight containers should be chosen for reliability
- **Implement Robust Access Control**
 - In containers, all the users are assigned root privileges by default. Therefore, it is necessary to change their access privileges to a non-root user. By using role-based access control (RBAC), you can configure specific sets of permissions.

Guide to Container Security Best Practices (4)

- Handle Confidential Data With Care
 - Never store secrets like keys, tokens, passwords, and confidential information inside docker files, because even if the data is deleted, it can easily be retrieved from the image history.

Summarizing Container Security

- Containers are gaining popularity as they are efficient and fast. Therefore, Containers Security must require a different approach. So, one should follow these Container Security Best Practices.