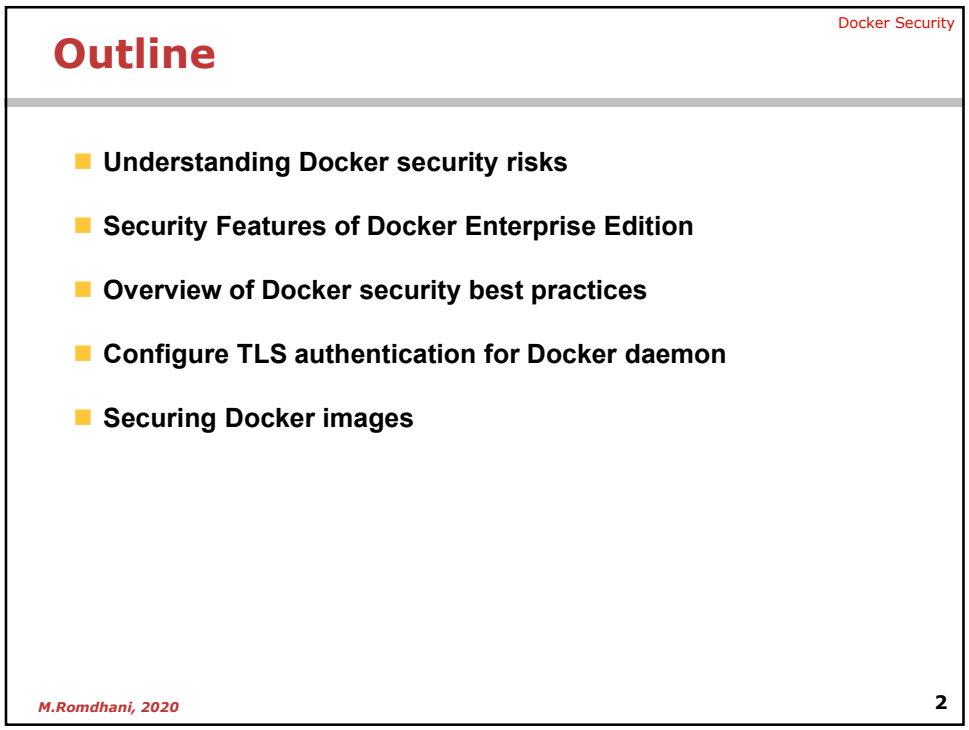

docker
Unit 6
Docker Security
Best Practices
Business
Training

1


Docker Security
Outline
Understanding Docker security risks
Security Features of Docker Enterprise Edition
Overview of Docker security best practices
Configure TLS authentication for Docker daemon
Securing Docker images
M.Romdhani, 2020
2

2

Understanding Docker security risks

3

Top Security Risks in Docker Container

[NIST Application Container Security Guide
https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-190.pdf]

- The risks are related to the core components of container technologies: images, containers, registries, orchestrators, and host OSs.
- The are two types of risks:
  1. **Compromise of an image or container**. The primary "data" to protect is the images and containers, which may hold app files, data files, etc. The secondary data to protect is container data within shared host resources such as memory, storage, and network interfaces.
  2. **Misuse of a container to attack other containers, the host OS, other hosts, etc.**
- Top 5 Security Risks in Docker Container Deployment
  1. Unsecured communication and unrestricted network traffic
  2. Unrestricted access of process and files
  3. Kernel level threats
  4. Unverified Docker Images
  5. Inconsistent Update And Patching Of Docker Containers

4

Unsecured communication and unrestricted network traffic

- **By default, in some versions of Docker, all network traffic is allowed between containers on the same host.**
  - This increases the risk of unintended and unwanted disclosure of information to other containers.
  - **Developers should only allow intercommunication that is necessary by linking specific containers.** This will significantly reduce attack surfaces by restricting container access.
- **To protect confidentiality and integrity of all network traffic, communications with Docker registries should be encrypted through TLS security protocol.**

5

Unrestricted access of process and files

- **An attacker who gains access to one container may have the capability to gain access to other containers or the host.**
  - For example, a container may have the ability to access system file directory on the host via remounting, which is critical to security enforcement.
- **If the attacker has root access to the container, he or she may have the ability to gain root access to the host, often through bugs in the application code.**
  - The access control best practice recommendations includes **the principle of least privilege**. The user namespace feature in Linux containers will allow developers to avoid root access by giving isolated containers separate user accounts, and mandate resource constraints, so users from one container do not have the capability to access other containers or exhaust all resources on the host.
- **However, this feature is not enabled by default. System administrators should have this feature enabled to leverage application sandboxing in Docker.**

6

Kernel level threats

- Docker is designed to have all containers share the same kernel and the host. This provides convenience but also amplifies the impact of any vulnerabilities present in the kernel.
- System administrators should ensure that the host environment is properly hardened and patched to reduce the likelihood of privilege escalation, arbitrary code execution, and denial of services.
- They should also restrict applications that run on privileged ports besides the ones that are necessary (e.g. Apache server), since those ports have more access to the kernel.

7

Unverified Docker Images

- Developers need to make sure they are downloading Docker images from trusted sources that are curated by the Docker community or the vendor, and run vulnerability scans against those images before running them in the host environment.
- To ensure integrity of a container image, content trust should be enabled to allow digital signatures for data sent to and received from remote Docker registries.

8

Inconsistent Update And Patching Of Docker Containers

- Running an older version of Docker containers can expose internal IT environments to higher risks of breach, and potential loss of sensitive information.
- New security features and bug fixes are often included in the update packages. Like any technology we run in the IT environment, a patching policy should be in place and enforced.

9

Security Features of Docker Enterprise Edition

10

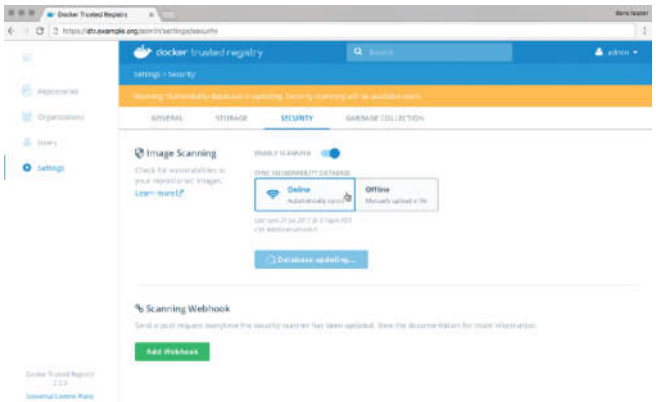
Docker Entreprise Security

- **Docker Entreprise Edition adds an extra layer of protection that travels with your applications in a secure supply chain that traverses any infrastructure and across the application lifecycle.**
  - And with a single interface and centrally-managed content, you get a seamless workflow that improves governance and ensures compliance across your whole organization.
- **Should You Use Docker CE or Docker EE?**
  - Deciding between Docker CE and Docker EE? For most, it boils down to these key questions:
    - Will you need same-day Docker support?
    - Does your organization's security policies compel you to use certified Docker images and plugins?
    - Would you like a single pane view to assist you in running, managing, and securing a wide range of containers in Linux and Windows?
    - How do you plan to visualize and manage your container environment? Do you require the 'Standard' and 'Advanced' Docker EE solutions which include this functionality?
    - Do you require enhanced security protocols?

11

Automated Governance

- **Docker Enterprise oversees your containerized applications at scale, interweaving flexible governance rules with your existing policies.**
  - **Granular & flexible role-based access controls (RBAC)**
  - **Secure application zones enable your team to provide secure multi-tenancy within individual clusters.**

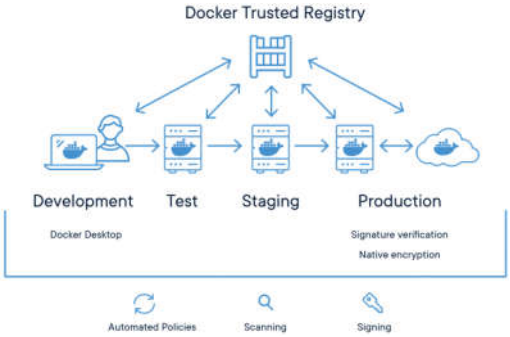


12

Secure Content Across the Software Supply Chain

Docker Security

- **Docker Enterprise offers cryptographic digital signing to confirm container image provenance and authenticity**
  - Image signing and vulnerability scanning allow your operations teams to have a clear understanding of what is inside your container;
  - Policy-based image promotion and image pruning accelerates the DevOps pipeline, allowing you to act on images that pass security scans by promoting them automatically



M.Romdhani, 2020

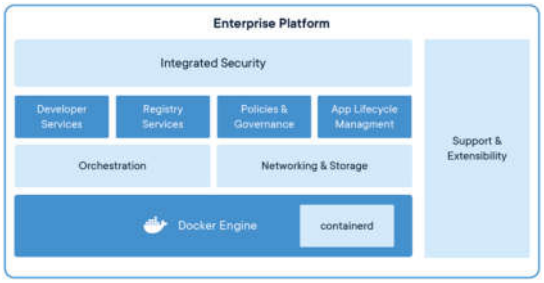
13

13

Secure Platform

Docker Security

- **Out-of-the-box security defaults include:**
  - **System-level mutual TLS authentication and cryptographic node identity** ensure that communications stay inside the cluster, and foreign nodes stay outside, preventing data leakage and attacks.
  - **Application-level isolation with authentication/authorization** lets you share resources without sacrificing security because you must explicitly open network communications to an application for any application or person to see or access it.
  - **FIPS 140-2 validated cryptographic modules** ensure that Docker Engine - Enterprise meets the standards required by the US Federal government and other regulated industries.



M.Romdhani, 2020

14

14

Overview of Docker security best practices

15

Auditing Security with DockerBench for Security

- **The Docker Bench for Security is a script that checks for dozens of common best-practices around deploying Docker containers in production.**
  - The tests are all automated, and are inspired by the CIS Docker Benchmark v1.2.0 [**https://www.cisecurity.org/benchmark/docker**/].
- **Docker Bench Security should be considered a must-use script. Here's what you can check:**
  - Host Configuration
  - Docker Daemon Configuration
  - Docker Daemon Configuration Files
  - Container Images and Build Files
  - Container Runtime Host Configuration
  - Docker Daemon Configuration
  - Docker Daemon Configuration Files
  - Container Images and Build Files
  - Container Runtime

16

Docker Security

Container Security Best Practices

- **Protecting Root Privileges**
  - By default, Docker requires root privileges to create and manage containers. The malicious script can leverage this attack surface to escalate to a superuser on a Linux host and eventually access sensitive files/folders, images, certificates, etc.
  - To prevent it, we can decide to drop capabilities such as **setgid** and **setuid** to prevent other programs or processes from changing their GID to another GID which can result in escalation privilege.
  - The command below runs the apache webserver container and drops the setgid and setuid capabilities via **--cap-drop** to prevent the apache container from changing its GID and UID to another UID and GID.

```
docker run -d --cap-drop SETGID --cap-drop SETUID apache
```

M.Romdhani, 2020 17

17

Docker Security

Container Security Best Practices

- **Docker User**
  - Apart from preventing other programs or processes, you can also create a user to manage docker operations such as docker run instead of managing it via a superuser.
  - You can add or create a docker user via the following commands:

```
sudo groupadd docker
sudo useradd john
```

  - Finally use the command below to add a user mike to the group docker to administer docker operations.

```
sudo usermod  -aG docker mike
```

M.Romdhani, 2020 18

18

Container Security Best Practices

- **Securing the Docker Daemon**
  - It is also necessary to configure the Docker daemon to ensure secure communication between docker client and docker daemon via **TLS**.
  - Use the following command to open **daemon.json** file and copy and paste the following content (replace the IP with your actual) as shown below

daemon.json

```
{
  "debug": false,
  "tls": true,
  "tlscert": "/var/docker/server.pem",
  "tlskey": "/var/docker/serverkey.pem",
  "hosts": ["tcp://192.168.16.5:2376"]
}
```


Managing Container with CGroups

- **Use Cgroups to set resource limits for your containers**
  - You can actually set resource limits for your individual containers right from the run command. This is useful when a container goes awry and begins to consumer all of your host's resources? This is certainly not a recipe for success and security.
- **For example, say you want to limit a container to 1GB of memory, you can add the --memory="1000M" option to the run command.**
- **You can also limit the number of CPUs with the addition of the --cpus=X (Where X is the number of CPUs you want available to your container).**

Managing Container with Namspaces

- **Managing Containers with Namespaces**
  - A namespace can prevent containers from running as privileged users, which can help to avoid privilege escalation attacks.
  - We can enable namespace in docker by making use of **/etc/subuid** and **/etc/subgid** files as shown below.
    - create a user using the adduser command

      `sudo adduser dockremap`
    - Setup a subuid for the user dockremap

      `sudo sh -c 'echo dockremap:400000:65536 > /etc/subuid'`
    - Then set up subgid for the user dockremap

      `sudo sh -c 'echo dockremap:400000:65536 > /etc/subgid'`
    - Open **the daemon.json** file and fill it with the following content to associate the userns-remap attribute to the user dockremap

```
{
 "userns-remap": "dockremap"
}
```

    - Finally restart docker to enable namespaces on a docker host

      `sudo /etc/init.d/docker restart`

21

Configure TLS authentication for Docker daemon

22

Docker Security

Protect the Docker daemon socket

- **By default, Docker runs through a non-networked UNIX socket. It can also optionally communicate using an HTTP socket.**
  - If you need Docker to be reachable through the network in a safe manner, you can enable TLS by specifying the **tlsverify** flag and pointing Docker's **tlscacert** flag to a trusted CA certificate.
  - In the daemon mode, it only allows connections from clients authenticated by a certificate signed by that CA. In the client mode, it only connects to servers with a certificate signed by that CA.
- **Create a CA, server and client keys with OpenSSL[https://docs.docker.com/engine/security/https/]**
  - First, on the Docker daemon's host machine, generate CA private and public keys:

```
openssl genrsa -aes256 -out ca-key.pem 4096
openssl req -new -x509 -days 365 -key ca-key.pem -sha256 -out ca.pem
```

  - Now that you have a CA, you can create a server key and certificate signing request (CSR). Make sure that "Common Name" matches the hostname you use to connect to Docker:

```
openssl genrsa -out server-key.pem 4096
openssl req -subj "/CN=$HOST" -sha256 -new -key server-key.pem -out
server.csr
```


23

Docker Security

Protect the Docker daemon socket

- Next, we're going to sign the public key with our CA:
  - Since TLS connections can be made through IP address as well as DNS name, the IP addresses need to be specified when creating the certificate. For example, to allow connections using 10.10.10.20 and 127.0.0.1

```
echo subjectAltName = DNS:$HOST,IP:10.10.10.20,IP:127.0.0.1 >> extfile.cnf
```

  - Set the Docker daemon key's extended usage attributes to be used only for server authentication:

```
$ echo extendedKeyUsage = serverAuth >> extfile.cnf
```

  - Now, generate the signed certificate:

```
$ openssl x509 -req -days 365 -sha256 -in server.csr -CA ca.pem -CAkey ca-
key.pem -CAcreateserial -out server-cert.pem -extfile extfile.cnf
```


24

Docker Security

Protect the Docker daemon socket

- After generating cert.pem and server-cert.pem you can safely remove the two certificate signing requests and extensions config files:

```
$ rm -v client.csr server.csr extfile.cnf extfile-client.cnf
```

- Now you can make the Docker daemon only accept connections from clients providing a certificate trusted by your CA:

```
$ dockerd --tlsverify --tlscacert=ca.pem --tlscert=server-cert.pem --
tlskey=server-key.pem -H=0.0.0.0:2376
```

- To connect to Docker and validate its certificate, provide your client keys, certificates and trusted CA. This step should be run on your Docker client machine.

```
$ docker --tlsverify --tlscacert=ca.pem --tlscert=cert.pem --tlskey=key.pem
-H=$HOST:2376 version
```


25

Docker Security

Protect Private Registries

- **Running a local registry**
  - Use a command like the following to start the registry container:
    - `$ docker run -d -p 5000:5000 --restart=always --name registry registry:2`
  - Running a registry only accessible on localhost has limited usefulness. In order to make your registry accessible to external hosts, you must first secure it using **TLS**

- **Steps to protect the local registry**
  - Get a Certificate
    - **Create a certs directory**.

```
$ mkdir -p certs
```

Copy the **.crt** and **.key** files from the CA into the **certs** directory.

- Stop the registry if it is currently running.

```
$ docker container stop registry
```

- Restart the registry, directing it to use the TLS certificate.

```
$ docker run -d \
  --restart=always \
  --name registry \
  -v "$(pwd)"/certs:/certs \
  -e REGISTRY_HTTP_ADDR=0.0.0.0:443 \
  -e REGISTRY_HTTP_TLS_CERTIFICATE=/certs/domain.crt \
  -e REGISTRY_HTTP_TLS_KEY=/certs/domain.key \
  -p 443:443 \
  registry:2
```


26

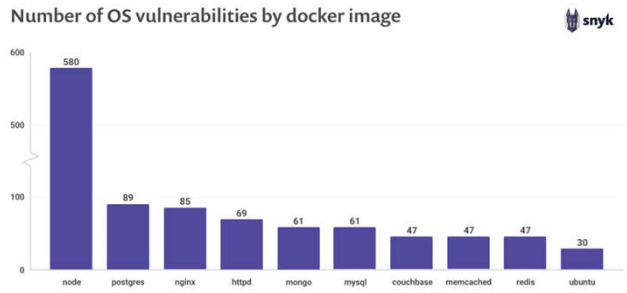
Securing Docker images

27

Docker Security

Docker Image Security Best Practices

- **Prefer minimal base images**
  - Often times, you might start projects with a generic Docker container image such as writing a Dockerfile with a FROM node, as your “default”. However, when specifying the node image, you should take into consideration that the fully installed Debian Stretch distribution is the underlying image that is used to build it.



28

Docker Image Security Best Practices

- **Least privileged user**
  - When a Dockerfile doesn't specify a USER, it defaults to executing the container using the root user.
  - To minimize exposure, opt-in to create a dedicated user and a dedicated group in the Docker image for the application; use the USER directive in the Dockerfile to ensure the container runs the application with the least privileged access possible.

```
FROM ubuntu
RUN mkdir /app
RUN groupadd -r john && useradd -r -s /bin/false -g john john
WORKDIR /app
COPY . /app
RUN chown -R john:john/app
USER john
CMD node index.js
```

  - The example above:
    - creates a system user (-r), with no password, no home directory set, and no shell
    - adds the user we created to an existing group that we created beforehand (using groupadd)
    - adds a final argument set to the user name we want to create, in association with the group we created

29

Docker Image Security Best Practices

- **Sign and verify images to mitigate MITM attacks**
  - Authenticity of Docker images is a challenge. Tampering may occur over the wire, between the Docker client and the registry, or by compromising the registry of the owner's account in order to push a malicious image to.
  - **Verify docker images**
    - Docker defaults allow pulling Docker images without validating their authenticity, thus potentially exposing you to arbitrary Docker images whose origin and author aren't verified.
    - To experiment with verification, temporarily enable Docker Content Trust with the following command:

```
export DOCKER_CONTENT_TRUST=1
```

    Or by CLI`> docker pull–disable-content-trust=false alpine`

    Now attempt to pull an image that you know is not signed—the request is denied and the image is not pulled.
  - **Sign docker images**
    - Prefer Docker Certified images that come from trusted partners who have been vetted and curated by Docker Hub rather than images whose origin and authenticity you can't validate.
    - Docker allows signing images, and by this, provides another layer of protection. To sign images, use **Docker Notary**. Notary verifies the image signature for you, and blocks you from running an image if the signature of the image is invalid.

30

Docker Image Security Best Practices

- **Don't leak sensitive information to Docker images**
  - Sometimes, when building an application inside a Docker image, you need secrets such as an **SSH private key to pull code from a private repository**, or you need tokens to install private packages. If you copy them into the Docker intermediate container they are cached on the layer to which they were added, even if you delete them later on. **These tokens and keys must be kept outside of the Dockerfile**.
  - **Using multi-stage builds**
    - By leveraging Docker support for multi-stage builds, fetch and manage secrets in an intermediate image layer that is later disposed of so that no sensitive data reaches the image build. Use code to add secrets to said intermediate layer, such as in the following example:

```
FROM ubuntu as intermediate

WORKDIR /app
COPY secret/key /tmp/
RUN scp -i /tmp/key build@acme/files .

FROM ubuntu
WORKDIR /app
COPY --from=intermediate /app .
```


31

Docker Image Security Best Practices

- **Don't leak sensitive information to Docker images (Cont'd)**
  - **Using Docker secret commands**
    - Use the experimental feature in Docker for managing secrets to mount sensitive file without caching them, similar to the following:

```
# syntax = docker/dockerfile:1.0-experimental
FROM alpine

# shows secret from default secret location
RUN --mount=type=secret, id=mysecret cat /run/secrets/mysecret

# shows secret from custom secret location
RUN --mount=type=secret,id=mysecret,dst=/foobar cat /foobar.
```

  - **Beware of recursive copy**
    - You should also be mindful when copying files into the image that is being built. For example, the following command copies the entire build context folder, recursively, to the Docker image, which could end up copying sensitive files as well:

```
COPY . .
```

    - If you have sensitive files in your folder, either remove them or use `.dockerignore` to ignore them:

```
# .dockerignore
private.key
appsettings.json
```


32

Docker Image Security Best Practices

- **Use fixed tags for immutability**
  - The most common tag is latest, which represents the latest version of the image. Image tags are not immutable, and the author of the images can publish the same tag multiple times.
  - Prefer the most specific tag available. If the image has multiple tags, such as :8 and :8.0.1 or even :8.0.1-alpine, prefer the latter, as it is the most specific image reference.
- **Use metadata labels**
  - Image labels provide metadata for the image you're building. This help users understand how to use the image easily. The most common label is "maintainer", which specifies the email address and the name of the person maintaining this image. Add metadata with the following LABEL command:

```
LABEL maintainer=me@mycompangy.com
```

  - It is good practice to add a SECURITY.TXT (RFC5785) file that points to your responsible disclosure policy for your Docker label schema when adding labels, such as the following:

```
LABEL securitytxt="https://www.example.com/.well-known/security.txt"
```


33

Docker Image Security Best Practices

- **Use multi-stage build for small and secure images**
  - While building your application with a Dockerfile, many artifacts are created that are required only during build-time. Keeping these artifacts in the base image, which may be used for production, results in an increased Docker image size, and this can badly affect the time spent downloading it as **well as increase the attack surface because more packages are installed as a result**.
- **Use COPY instead of ADD**
  - While subtle, the differences between ADD and COPY are important. Be aware of these differences to avoid potential security issues:
    - When remote URLs are used to download data directly into a source location, they could result **in man-in-the-middle attacks that modify the content of the file being downloaded**. Moreover, the origin and authenticity of remote URLs need to be further validated. When using COPY the source for the files to be downloaded from remote URLs should be declared over a secure TLS connection and their origins need to be validated as well.
    - When local archives are used, ADD automatically extracts them to the destination directory. While this may be acceptable, it adds **the risk of zip bombs and Zip Slip vulnerabilities** that could then be triggered automatically.

34

Docker Image Security Best Practices

- **Use multi-stage build for small and secure images**
  - While building your application with a Dockerfile, many artifacts are created that are required only during build-time. Keeping these artifacts in the base image, which may be used for production, results in an increased Docker image size, and this can badly affect the time spent downloading it as **well as increase the attack surface because more packages are installed as a result**.
- **Use COPY instead of ADD**
  - While subtle, the differences between ADD and COPY are important. Be aware of these differences to avoid potential security issues:
    - When remote URLs are used to download data directly into a source location, they could result **in man-in-the-middle attacks that modify the content of the file being downloaded**. Moreover, the origin and authenticity of remote URLs need to be further validated. When using COPY the source for the files to be downloaded from remote URLs should be declared over a secure TLS connection and their origins need to be validated as well.
    - When local archives are used, ADD automatically extracts them to the destination directory. While this may be acceptable, it adds **the risk of zip bombs and Zip Slip vulnerabilities** that could then be triggered automatically.

35

Docker Image Security Best Practices

- **Use a linter**
  - Adopt the use of a linter to avoid common mistakes and establish best practice guidelines that engineers can follow in an automated way.
  - One such linter is **hadolint**. It parses a Dockerfile and shows a warning for any errors that do not match its best practice rules.

36