Six Ways to Bake in Smarter Security with Containers

Containers have generated plenty of fear, uncertainty, and doubt in the blogosphere about what's needed to secure them. They are ephemeral, they are too numerous to count, they talk to each other (East-West) more than they communicate with the outside World (North-South), and they are typically part of a fast-moving continuous integration/continuous deployment (CI/CD) freight train. They also bring with them an ecosystem of additional tooling that also increases the attack surface.

At the same time, containers bring with them significant security benefits, depending on how you implement and configure your container environment. You can leverage many inherent security advantages as you move applications to containers – even if you're porting a monolithic app and not yet adopting microservices. With containers, you can:

- Minimize your attack surface much more effectively than is possible with a virtual machine (VM) or bare metal server by reducing functionality in the container to the bare minimum set of processes, image components, filesystem access, and other settings.
- Reduce the likelihood that an attacker will be able to exploit the packages, tools, and other elements that do remain.

1. Set Filesystem as Read-Only

Containers are supposed to be immutable; therefore, once running, many containers do not need changes to the root filesystem. For those containers that do not require writing files, set the filesystem as read-only.

Why Do This?

If the container does not require filesystem writes, the only processes that will attempt changes to the filesystem should be considered malicious (e.g., someone is trying to escalate privilege or drop a payload). To prevent this risk, make the filesystem read-only.

2. Implement Multi-stage Builds

Before Docker introduced multi-stage builds, it was common for developers to have one big development Dockerfile with everything needed to build the application and a separate production Dockerfile containing only the application and necessary services. This development practice is cumbersome to manage and difficult to support in a fast-moving CI/CD environment.

Docker's multi-stage builds allow you to use FROM statements in the Dockerfile to establish multiple build stages and discern packages and data that are not necessary at runtime in the container.

Why Do This?
By copying only the required files and libraries into the final version of the image, you minimize the attack surface. For example, you might build a static Golang binary that requires multiple compile time dependencies, but you shouldn't include those dependencies in the final image. Multi-stage builds allow you to create an image with just the final executable.

3. Leverage CAP DROP

In the Linux kernel, specific units of privilege are called Linux capabilities – 38 distinct capabilities exist today. Docker has a feature called Cap Drop that allows you to drop specific Linux capabilities, which reduces the privileges of the container. Docker already limits the default capabilities, but you can further minimize risk by dropping additional unnecessary capabilities.

Why Do This?
Following the principle of least privilege, it's always best to grant only the needed capabilities. One capability to consider dropping is CAP_NET_RAW, which allows for binding to any address for transparent proxying, which typically isn't desirable. A RAW socket gives adversaries the vector to inject all sorts of nastiness onto the network. Shut it down.

4. Leverage CAP ADD

CAP ADD allows a container to have finer-grained privileges in the form of specific capabilities.

A typical Docker container will offer the following capabilities by default: CHOWN, DAC_OVERRIDE, FSETID, FOWNER, MKNOD, NET_RAW, SETGID, SETUID, SETFCAP, SETPCAP, NET_BIND_SERVICE, SYS_CHROOT, KILL, and AUDIT_WRITE. Applications will often require additional capabilities, and it's tempting to run containers in privileged mode for simplicity. However, this approach opens up all sorts of potential security risks. Instead, introduce only those incremental capabilities that an application requires to run.

Why Do This
Using CAP ADD prevents having to set the --privileged flag since only specific capabilities may be required. For example, running the official elasticsearch image requires mlockall, which is not part of the default Docker runtime capabilities. Using CAP ADD with IPC_LOCK allows the container to use mlockall without granting the rest of the capabilities.

5. Manage Secrets

A secret is data that happens to be sensitive. It might be a password, an SSH private key, an SSL certificate, or any data that should not be transmitted or stored unencrypted. Docker 1.13 and higher provides the ability to centrally manage secrets, including during transit and while at rest. Use the secrets management capabilities built into orchestrators such as Kubernetes or Docker Swarm. You can also use third-party secrets management software.

Why Do This?
Use secrets management tools instead of environment variables because the secrets management tools keep the secrets themselves invisible, but environment variables do not. Secrets management tools inject values into the container during runtime. Secrets may be used both for confidentiality and integrity of sensitive data and as an abstraction layer separating a container and its credentials. A great use case is never having "live" secrets during dev/test by automatically drawing from different data sets than those used in production.

6. Impose PID Limits

One of the advantages of containers is tight process identifier (PID) control. Each process in the kernel carries a unique PID, and containers leverage Linux PID namespace to provide a separate view of the PID hierarchy for each container. Putting limits on PIDs effectively limits the number of processes running in each container. PID limits is generally available in Docker and as an alpha feature in Kubernetes (K8s) 1.10.

Why Do This?
Limiting the number of processes in the container prevents excessive spawning of new processes and potential malicious lateral movement. Imposing PID limits also prevents fork bombs (processes that continually replicate themselves) and anomalous processes. Mostly, the benefit here is if your service always runs a specific number of processes, then setting the PID limit to that exact number mitigates many malicious actions, including reverse shells and remote code injection –really, anything that requires spawning a new process.

Time to Bake in Smarter Security

So, get out your container cookbook and cook up better security by focusing on these six key container ingredients. Even moving a monolithic application to a container architecture, leveraging these settings, can reduce your attack surface dramatically.

Want to learn more about locking down your container environment? Check out the *StackRox Container Security Platform* for more capabilities.

StackRox

StackRox helps enterprises secure their containerized, cloud-native applications at scale. The StackRox Container Security Platform enables security teams to discover the full container environment and ensure they adhere to security policies, and it detects and stops malicious activity. StackRox customers span Global 2000 enterprises, including in financial services, technology, and E-Commerce industries, as well as government agencies.

LET'S GET STARTED

Request a demo with StackRox today!

info@stackrox.com
+1 (650) 489-6769
www.stackrox.com

© 2018 StackRox, Inc. All rights reserved.