docker

# Strengthen your supply chain with secure containers

# Strengthen your supply chain with secure containers

Containers are continuing to revolutionize the software development landscape, bringing agility, scalability, and portability to the businesses that implement this technology. However, bad actors also continue to capitalize on this additional attack surface, leading to unique security challenges that cannot be ignored. In 2022, 61% of US businesses were directly impacted by a software supply chain threat[1], and there has been a 644% increase in supply chain attacks since 2021[2].These threats aren't going anywhere, so understanding the risks and potential vulnerabilities that attackers may exploit is key to a clean and secure software supply chain.

By integrating robust security measures throughout the software development lifecycle, businesses can reduce their vulnerability and strengthen operational resiliency. Secure development plays a pivotal role in this endeavor by enabling the creation of safe, reliable, and maintainable software systems.

[1] https://www.infosecurity-magazine.com/news/software-supply-chain-attacks-hit/

[2] https://www.csoonline.com/article/573925/supply-chain-attacks-increased-over-600-this-year-and-companies-are-falling-behind.html

## Why secure containers?

A secure supply chain starts with secure development. Container development tools have revolutionized the way software is packaged, shipped, and executed. These tools provide developers with an efficient and scalable solution to build applications and their dependencies in an isolated environment.

Proactive container development security means stopping anything malicious from getting into your developer environments in the first place. This is where using container technology becomes advantageous. Alongside the amplified scalability, flexibility, and reproducibility that containers offer, in general, running software inside a container is more secure than running it natively on a host. This is because the container is a smaller attack vector and the host is isolated. Most container development tools already come with baseline security features to protect against common security risks. These baseline container security features:

- Allow users to run images unrestricted (no admin rights needed).
- Run all containers in a Linux Virtual Machine (VM) to isolate them from the underlying Mac/Windows/Linux host.
- Prevent containers from accessing host files not owned by the user.
- Offer configurations for network proxies, registries, tool updates, host file sharing, and more.

## Container security risks

While container development tools offer baseline security functionality, if a container is breached, it can lead to unauthorized network access, data breaches, and compromise of the entire system. Understanding how you can be breached and how to combat those risks is essential to ensuring the integrity, confidentiality, and availability of containerized applications and their underlying infrastructure.

The top five security risks we see to container development tools today are malware threats, supply chain attacks, inadequate access control, misconfigurations, and insider threat.

### RISK #1 Malware threats

If malware enters your network infrastructure, it can exfiltrate sensitive data and credentials and compromise the container and its underlying host system. The most common way for malware to get into your systems is when developers inadvertently download a malicious container image on their local machine and run it, or when a developer exposes the container engine API on the network without protection.

Developers often rely on pre-built images from public repositories or create their own images. Unfortunately, these images can be vulnerable or contain outdated software components. Free and open source software has its advantages, but also means those container images can contain vulnerabilities. While one advantage of using open source software is that developers can achieve faster release cycles, one significant disadvantage is that organizations may not collaborate at the speed needed to maintain a secure development life cycle (SDLC) practice. In fact, an alarming 87% of container images running in production have critical or high-severity vulnerabilities, up from 75% a year ago.[3]

[3] https://sysdig.com/2023-cloud-native-security-and-usage-report/

The same malware threats apply for public registries. If these registries lack proper access controls or authentication mechanisms, they are susceptible to image tampering. An attacker gaining control over a compromised container registry can distribute malicious images leading to widespread security breaches.

## RISK #2 Supply chain attack

Supply chain attacks aim to compromise the integrity of containerized applications or their dependencies. Attackers may inject malicious code or tamper with the software supply chain, leading to the distribution of compromised containers.

Once an attacker successfully breaks out of a container, they can move laterally within the containerized environment, escalate privileges, access sensitive resources, and launch further attacks. Container breakout and lateral movement attacks can have severe consequences, jeopardizing the security and integrity of the entire container infrastructure.

## RISK #3 Inadequate access controls

Without robust access controls, malicious actors can gain unauthorized access to sensitive containers, manipulate their configurations, or exfiltrate sensitive data. Insufficient authentication practices can lead to unauthorized access to container orchestration systems, management consoles, or containerized applications, allowing attackers to exploit vulnerabilities and compromise the entire environment.

If a developer has too many local admin rights, they can relax firewall rules, turn off anti-malware software, disable updates, and install untrusted software.

If a developer's workstation is not configured securely per organizational policies, attackers can much more easily compromise these machines and from there, the organization's network.

## RISK #4 Misconfigurations

Improperly configured containers, orchestrators, or network settings may allow unauthorized access, data leaks, or unintended exposure of sensitive information. Attackers actively scan for misconfigurations and exploit them to gain control over containers or compromise the underlying infrastructure.

Misconfigurations often happen because there is no centralized or mandatory organizational control over the tool's security settings. Even if an IT admin initially configures the tool's settings per organizational security policy, a developer can always modify these settings at will.

## RISK #5 Insider threat

Insider threats don't always have to have ill intent behind them. These risks can come from anyone who has access to container development tools intentionally and unintentionally misusing their privileges. Proper access controls, monitoring, and security awareness training are useful in mitigating insider threats and preventing data breaches, unauthorized access, or malicious modifications to containerized applications. However, many organizations have a long way to go to mitigate insider threats

and embrace cross-team security training: 77% of organizations acknowledged that they lack effective collaboration among developers and security teams.[4]

## How to mitigate these risks

In recent years there has been increasingly heightened attention on putting guardrails in place to further secure the software development lifecycle. These standards and controls, such as the SLSA (supply-chain Levels for Software Artifacts) security framework and the NIST (National Institute of Standards and Technology) Cybersecurity Framework promote clean code practices, to reduce vulnerabilities, improve software integrity, and secure infrastructure. While these frameworks are currently suggestions, it's not unimaginable that we'll see a shift towards enforcing these standards, similar to the Executive Order released in 2022 addressing the heads of executive departments and agencies on enhancing the security of the software supply chain through secure software development practices.[5] Here are some additional steps you can take to prevent security risks in your container development tools:

1. Generate a Software Bill of Materials (SBOM) - these are essential to understanding the provenance of where an artifact came from. Employing SBOMs gives you a starting point for cleaning up your code.
2. Start with secure base images - begin your container development with secure and trusted base images from reputable sources. These images should be regularly reviewed for vulnerabilities and updated to include the latest security patches and fixes.
3. Receive real time vulnerability updates - Regularly index container images with known vulnerabilities, ensuring timely identification and remediation of security risks.
4. Employ image signing and verification - Ensure the integrity and authenticity of container images using digital signatures.
5. Restrict secure container registries - Implement strong access controls to only allow access to trusted sources for container images and limit access to non-authorized registries. You want to ensure accessible registries are regularly monitored for vulnerabilities and malware.
6. Enforce access controls - Use role-based access controls (RBAC) to grant permissions based on job responsibilities.
7. Fortify network isolation - Minimize the impact of a potential breach/container breakout or lateral movement within the containerized environment by hardening container isolation. Regularly patch and update container runtimes, host systems, and underlying dependencies.

## Resolve security issues before they make it into production

All of the measures listed above work great for baseline container security, but in organizations with stringent security and compliance requirements, sometimes you need a bit more. You need a mechanism for IT admins to enforce security settings on the container development platforms (e.g., proxies, registries, Docker Engine configs) with stronger guarantees that won't be easily bypassed by developers or malicious containers.

[4] https://snyk.io/reports/state-of-cloud-security/

[5] https://www.whitehouse.gov/wp-content/uploads/2022/09/M-22-18.pdf

**Advanced isolation measures**

Strong container isolation within the VM makes it that much harder for malicious users, or workloads running in the container, to compromise the host. Container isolation mechanisms should be applied automatically so they do not impact developer workflows and teams can continue using Docker as usual.

Hardened Docker Desktop (HDD) provides an additional layer of security for Docker Desktop users that further fortifies container isolation. With HDD, containers inside the Docker Desktop Linux VM run unprivileged (i.e., root user in the container maps to an unprivileged user inside the VM) to prevent them from taking control of the VM. This keeps containers from bypassing Docker Desktop registry configurations, accessing sensitive data or secrets stored in the VM, accessing the Docker Engine, peeking into other containers, or attacking the host machine.

In addition, HDD's Settings Management feature allows admins to preset and "lock" several Docker Desktop configurations on a developer's machine. Once set, these configurations cannot be altered by developers.

**Access management**

Access controls and granular permission settings ensure the security and confidentiality of sensitive information. This includes the types of images and repositories developers are allowed to pull. As open source software continues to make up the majority of our applications, developers benefit from these guardrails to significantly reduce the chance of using a container image with a malicious payload.

Docker's Trusted Content provides vetted, reliable, and trusted container images for your teams to build from. When used in combination with Registry Access Management and Image Access Management, you can ensure that your developers are only using images that fit business compliance requirements.

*Trusted Content includes:*

- Docker Official Images - Container images curated and maintained by Docker, providing a trusted and reliable foundation for building containers. These images exemplify container image best practices.
- Docker Verified Publisher images - Container images coming from trusted publishers and their repositories. Each repository is vetted by Docker.
- Docker-Sponsored Open Source images - Community-driven projects supported by Docker.

**Vulnerability visibility**

Proactively identifying and mitigating security risks starts during development. Visibility into vulnerabilities will not only help agencies maintain compliance with necessary security standards and regulations, but will also improve the overall security posture of your software systems, ultimately reducing the likelihood of a successful attack.

Docker Scout helps developers find and fix container vulnerabilities at the earliest stages of software development. It provides a unified, layer-by-layer view of software dependencies, their known vulnerabilities, and recommended remediation paths. Docker Scout includes a software bill of materials (SBOM) that integrates seamlessly with any existing CI/CD pipeline or build process. As a result, admins maintain a verifiable record of their containerized software components.



## Preventing a weak supply chain with Docker

The modern era demands businesses to fortify their operations and infrastructure against an ever-growing array of cyber threats. Secure development tools provide a way to create resilient software systems while mitigating the risk of breaches. By adopting preventive measures and implementing best practices, organizations can strengthen the security posture of their containerized environments, ensuring the integrity, confidentiality, and availability of their applications and underlying infrastructure.

**Learn more about security and the Docker platform.**