docker

Five best practices for securing your software supply chain

Five best practices for securing your software supply chain

Traditional software development is often characterized by fragmented processes, long release cycles, and security vulnerabilities. Implementing a comprehensive, secure software supply chain (SSSC) framework that encompasses the entire software development lifecycle can address these challenges, from initial ideation to deployment and maintenance. By viewing the whole supply chain, you can enhance project visibility, collaboration, and traceability across stakeholders.

This white paper delves into the best practices and key benefits of securing your software supply chain. With a secure SSC, you can confidently pave the way for faster, more reliable, and more efficient software development and deployment.

Components of a secure supply chain

Core components of a secure SSC include automation, standardization, transparency, and trust:

- **Automation** streamlines repetitive tasks, accelerating the software delivery process and minimizing human error.
- **Standardization** ensures consistency and interoperability, seamlessly integrating components from various vendors.
- **Transparency** facilitates visibility into the software bill of materials (SBOM), enabling collaboration between stakeholders to identify potential vulnerabilities and make data-driven decisions.
- **Provenance & Trust** ensures that all artifacts and participants adhere to security requirements, fostering a culture of accountability and reliability.

Benefits of securing your software supply chain

Your SSC is composed of important tools that help you develop software, which is why it is essential to ensure it is secure. Supply chain attacks are continuing to increase at a rapid rate, with an over 600% increase in attacks since 2021[1] and according to Gartner, 45% of companies will be subject to such attacks by 2025.[2] What's more, 97% of software today is built on open source software[3] (OSS), which can come with vulnerabilities. Some are known, but new ones are found constantly and reported by the community. They are hard to find, keep track of, and properly match to a software artifact built

[1] https://info.aquasec.com/argon-supply-chain-attacks-study?utm_campaign=WP%20-%W20Jan2022%20Argon%20Supply%20Chain%20Study&utm_source=PR_Argon_Study

[2] https://research.contrary.com/reports/unlocking-the-cybersecurity-landscape

[3] https://venturebeat.com/programming-development/github-releases-open-source-report-octoverse-2022-says-97-of-apps-use-oss/

by a developer. As a result, most developers consider security a pain that gets in the way of building great software applications. Both of these factors can lead to massive and unforeseen application performance issues, data leaks, and consumer data privacy. Securing your software supply chain protects your business from these issues and helps:

- Enable faster time-to-market, empowering organizations to respond swiftly to market demands and gain a competitive advantage.
- Reduce human-induced errors and improve software quality by automating development, testing, and deployment processes.
- Facilitate interoperability, seamlessly integrating diverse software components from multiple sources through a standardized approach.
- Mitigate security vulnerabilities with enhanced transparency ensures comprehensive visibility into the SSSC.
- Create a trusted development environment by establishing a robust governance framework, fostering collaboration among stakeholders, prioritizing security and compliance, and adopting advanced technologies (such as containerization).

A secure SSC expedites enterprise software delivery by streamlining processes and enhancing collaboration while incrementally reducing security risks and reducing developer pain points. Embracing the benefits of this transformative process enables organizations to go to market quickly and stay ahead of competition.

Five best practices for securing your supply chain

PRACTICE #1 Build secure container images

Building secure container images is a critical aspect of modern software development that ensures the integrity of the overall application. With the rising popularity of containerization technologies like Kubernetes, software developers and system administrators face the challenge of managing many external software components while maintaining a robust security posture. Third-party dependencies introduce potential vulnerabilities that, if left unchecked, can compromise the overall security of containerized applications. For instance, running a Hello World sample project in React includes more than 2,800 dependencies[4]. Ensuring the integrity of each of these dependencies is paramount to building secure container images.

Managing third-party dependencies

Developers can best safeguard the integrity of third-party dependencies and libraries if they implement a strict version control strategy to track and manage the inclusion of external components within their codebase. By explicitly specifying the versions of dependencies, developers can ensure consistent and controlled integration, reducing the risk of compatibility issues or the inclusion of outdated and vulnerable software. Many organizations already do this, meaning they already meet supply chain levels for software artifacts (SLSA) build level one.[5]

Another secure SSC best practice is to use package managers and registries that provide trusted sources for obtaining software components. Through these mechanisms, developers can verify the

[4] https://medium.com/frontendweb/find-how-many-packages-we-need-to-run-a-react-hello-world-app-695fbb755af7

[5] https://slsa.dev/spec/v1.0/levels

authenticity and integrity of the dependencies, mitigating the risk of tampering or malicious injection.

Regularly updating and patching dependencies is crucial to address known and unknown vulnerabilities and to ensure the codebase remains secure over time. Developers can proactively protect their container images from potential attacks by staying informed about security advisories and promptly addressing vulnerabilities.

Selecting a container image

When selecting a container image, developers should consider several factors to ensure the security and reliability of their containerized applications. To build secure container images, they'll need:

Official and trusted sources. Developers should choose base images from official repositories or trusted sources, such as Docker Hub's verified publishers and official images programs (Docker Official Images and Docker Verified Publisher) or other reputable distribution vendors. These sources typically provide regularly updated and well-maintained base images with strong security practices.

Minimal and purpose-built images. Developers can opt for minimal base images that contain only the necessary components for their application. Using purpose-built images decreases the likelihood of including unnecessary or vulnerable dependencies.

Regular updates and security patches: Developers should select base images that receive frequent updates and security patches. Regularly updating the base image ensures that known vulnerabilities are addressed promptly, improving the overall security of the container.

Image provenance and integrity: Cryptographic verification techniques, such as image signatures and checksums, allow developers and their organizations to verify the authenticity and integrity of the container images. This helps prevent tampering or unauthorized modifications to container images during distribution or deployment.

Software Bill of Materials (SBOMs): SBOMs are crucial for understanding what components are in your container images. They provide a detailed breakdown of software components, their versions, and dependencies. With SBOMs, developers can track ownership, uncover licenses, and identify potential vulnerabilities within the layers of a container image. SBOMs play a vital role in vulnerability detection and image analysis, providing a comprehensive inventory of software components. This enables vulnerability detection tools to cross-reference against databases of known common vulnerabilities and exposures (CVEs), and provide accurate assessments so developers can identify and remediate issues in container images more effectively.

Vulnerability detection and advanced analysis. Developers should, at the very least, employ vulnerability detection tools to analyze container images for known security vulnerabilities. Advanced organizations take this a step further, employing advanced container image analysis to create and read from the SBOM to determine the content of an image and track its security posture over the lifetime of the image.

License compliance. SBOMs also facilitate license compliance. Developers can determine the licenses associated with each software component to ensure the container image complies with relevant open-source license requirements and avoids potential legal issues.

Software supply chain policy. A software supply chain policy defines guidelines, standards, and processes for selecting, verifying, and updating third-party dependencies, base images, and libraries used in container images. It helps establish a clear and consistent approach to SSC security, reducing the risk of incorporating vulnerable or untrusted components into the images.

Selecting the right container base image, adhering to secure building best practices, leveraging SBOMs, and employing vulnerability detection and image analysis tools are all essential components of securing your container images from the start. These processes ensure that containerized applications are built on a foundation of trusted and updated components, with a clear understanding of their licenses, provenance, and vulnerabilities, ultimately enhancing the security and reliability of the SSSC.

PRACTICE #2 Secure local development environments

Securing local development environments is crucial to protect sensitive data and ensure the integrity of containerized applications. Recommendations for securing local development environments include:

Use isolated networks. When developing locally, create separate networks for different projects or applications. This isolation prevents unintended network access between containers and reduces the impact of any potential security breaches.

Employ access controls and least-privilege principles. Ensure that only authorized individuals have access to resources, such as the container daemon. Apply role-based access controls (RBAC) and strong authentication mechanisms to restrict access and minimize the risk of unauthorized actions.

Regularly update container components. Keep applications for managing containers and their components up-to-date with the latest security patches and updates. Regular updates address known vulnerabilities and provide improved security features, as well as bug fixes and performance enhancements.

Secure container images. Best practices include selecting secure base images, regularly updating dependencies, and analyzing images for vulnerabilities. Apply secure configurations within the images, such as running containers with non-root users and turning off unnecessary services.

Implement image verification. Teams should be verifying the integrity and authenticity of container images during development. This can involve using digital signatures or checksums to ensure the images have not been tampered with or compromised.

Monitor container activity. Use monitoring tools and logging capabilities to track container activities during development. Monitoring allows you to detect suspicious behavior, identify potential security incidents, and respond promptly to mitigate harm.

Educate developers. Promote security awareness among developers. Provide training on secure coding practices, container security best practices, and guidelines for handling sensitive data. Encourage developers to adopt secure development habits and follow security guidelines throughout the development process.

By following these recommendations, developers can enhance the security of local development environments when working with an application like Docker Desktop. Implementing secure defaults, isolating networks, enforcing access controls, updating components, securing images, monitoring container activity, and educating developers all contribute to a robust security posture for containerized development workflows.

PRACTICE #3 Continuous integration and delivery (CI/CD)

Integrating security practices around container images into CI/CD pipelines is another key element for ensuring secure software delivery. Here are some important considerations when implementing security practices in CI/CD pipelines:

Automate basic security checks. Automated security checks should be a part of the CI/CD pipeline to analyze container images for known vulnerabilities, adherence to security policies, and compliance requirements. Automated checks include using container security tools to analyze the image layers and associated software components. These automated checks provide early detection of vulnerabilities, enabling prompt remediation and reducing the risk of deploying insecure images.

Differential vulnerability analysis. Employ differential vulnerability analysis techniques to compare new container image versions against the previous ones. Differential vulnerability analysis helps identify any newly introduced vulnerabilities or changes in the security posture of a container image. This allows security teams to focus on reviewing and addressing specific changes, rather than analyzing the entire image, saving time and resources.

Consider these practices when integrating security into CI/CD pipelines to establish a robust security posture for containerized deployments. Automated security checks, differential vulnerability analysis, secure deployment practices, software supply policies, continuous monitoring, and incident response mechanisms all contribute to enhancing the security and resilience of container images throughout the SSSC.

PRACTICE #4 Secrets management and configuration

Managing secrets and sensitive information in containers is essential to protect sensitive data from unauthorized access or exposure. By taking some basic precautions, organizations can reduce the chances of secrets being leaked in container images.

Secure environment variables. Avoid hardcoding secrets directly into Dockerfiles or container configurations. Instead, use environment variables to pass sensitive information to containers at runtime. Ensure that these environment variables are securely stored and encrypted, limiting access to authorized users or services.

Externalize configuration files. Store sensitive configuration files outside of container images. Mount the necessary configuration files as volumes or use a secrets management tool to securely provide the required configuration to containers during runtime. Storing the files outside of container images allows for easy management and updating of sensitive information without requiring changes to the container image.

Implement role-based access control (RBAC): Enforce RBAC to control access to secrets and sensitive information. Grant permissions to only the necessary individuals or services that require access to secrets, ensuring that sensitive data is accessible only by authorized entities.

Encrypt persistent storage. If sensitive information must be stored persistently within containers, ensure the storage volume is encrypted. This protects data at rest and adds a layer of security in case of unauthorized access to the underlying storage.

Regularly rotate secrets. Implement a practice of regularly rotating secrets, such as passwords or API keys, to minimize the impact of potential breaches. Automated processes or scripts can assist in seamlessly updating secrets in containers without causing disruptions to the running services.

Audit and monitor access. Implement logging and monitoring mechanisms to track and identify any unauthorized access attempts or suspicious activities related to secrets. Monitor access logs, container logs, and system logs to detect any potential security breaches or misuse of secrets.

By effectively managing secrets and sensitive information in containers, organizations will reduce the risk of unauthorized access and maintain a strong security posture within containerized environments.

PRACTICE #5 Secure production

Bringing metadata about production back into development is essential to creating a secure SSC as it enables a closed feedback loop that enhances security and reliability. This type of data also helps developers understand how their software will perform in real-world scenarios, so they can make informed decisions and enhance overall software quality and security.

Automate checks in production workloads against security policies. Validate the presence and correct usage of security controls, encryption mechanisms, access controls, and other relevant security measures. Automated checks help identify any deviations or vulnerabilities that may have been introduced during the development, build, or deployment stages, enabling prompt remediation and maintaining a secure SSC. This practice helps organizations ensure that the deployed software aligns with predefined security requirements.

Get visibility into production workloads with runtime agents. Runtime agents provide real-time visibility into production workloads by monitoring and collecting data on the behavior and performance of running applications. By deploying runtime agents, development teams can gain insights into the execution environment, identify potential vulnerabilities, and detect anomalous activities or security incidents. With this level of visibility, organizations can proactively address security issues, optimize application performance, and make informed decisions regarding future development and deployment strategies.

Share vulnerability information. The software supply chain benefits from sharing vulnerability information discovered in production environments back to development teams. This includes sharing details about common vulnerabilities and exposures (CVEs) and vulnerability exploitability (VEX) assessments. CVEs provide standardized identifiers for known vulnerabilities, while VEX assessments determine the exploitability of a vulnerability in a specific environment. By sharing this information, development teams can assess the impact of vulnerabilities on their software components, prioritize remediation efforts, and implement security controls to mitigate risks in future software versions.

Continuous monitoring and incident response. Implement continuous monitoring of containerized applications in production environments. Utilize log management, intrusion detection, and anomaly detection tools to promptly identify and respond to potential security incidents. Monitor image repositories for unauthorized access or image tampering. Establish an incident response plan to address security breaches and vulnerabilities promptly and efficiently.

Immutable infrastructure and rollback mechanisms. Use immutable infrastructure practices to ensure consistent and secure deployments. By treating infrastructure as immutable and deploying new versions as complete and separate entities, rollbacks become easier in case of security incidents or issues. Maintain backups and version control of container images to facilitate swift and secure rollback to a known good state if necessary.

Securely deploy container images to production environments. Best practices include using secure repositories like Docker Hub to store container images, implementing strong access controls and authentication mechanisms, and encrypting image transport. Additionally, utilize container orchestrators like Kubernetes with secure configurations to ensure proper isolation, network segmentation, and access control for deployed containers.

Bringing metadata about production back into development establishes a feedback loop that fosters continuous improvement in the software supply chain's security and reliability. Automating checks against security policies in production workflows, leveraging runtime agents for visibility, and sharing vulnerability information facilitate proactive identification and resolution of security issues, leading to more robust and secure software deployments. Following these best practices for securing production will significantly enhance the security of the supply chain, reduce vulnerabilities, and instill confidence in development and deployment practices.

How does Docker fit into a secure software supply chain?

Docker is a popular containerization platform that can address common security challenges in software development and deployment. Here are some key challenges that Docker can help mitigate as part of a secure SSC:

Isolation and environment consistency. Traditional software deployment often suffers from compatibility issues between different software components and dependencies. Docker provides lightweight, isolated containers that encapsulate the application and its dependencies, ensuring consistency across different environments. This isolation prevents conflicts between applications, reducing the risk of security vulnerabilities arising from shared resources.

Vulnerability management. Keeping software dependencies up-to-date is crucial for maintaining a secure environment and avoiding software breaches due to unpatched vulnerabilities. Docker Scout, our supply chain security tool, simplifies the process of identifying vulnerabilities in repositories. It seamlessly integrates into developer tools and systems, providing real-time vulnerability updates so developers can prioritize risks by severity. This enables basic point-in-time vulnerability detection and more advanced techniques, such as image analysis, which ensures known vulnerabilities are identified and addressed promptly, reducing the risk of exploitation.

Resource limitations. Attackers can exploit software vulnerabilities to gain unauthorized access to system resources, impacting the performance and security of other applications. Docker provides resource isolation capabilities, so administrators can limit the amount of CPU, memory, and disk space allocated to each container. Resource isolation capabilities prevent malicious activities within a compromised container from affecting the underlying host system or other containers, enhancing overall security.

Controlled access and privilege separation. Docker supports fine-grained access control mechanisms through user namespaces and container-level permissions. Access controls let administrators restrict container capabilities and privileges, ensuring containers run with minimal privileges necessary for their specific tasks. By separating permissions and limiting the scope of each container, Docker reduces the potential impact of container compromises, strengthening overall security posture.

Immutable infrastructure and deployment consistency. With Docker, container images can be treated as immutable artifacts, meaning they remain unchanged once built. This approach enables the creation of consistent deployment environments so developers can deploy the same container image across various stages of the SSSC. Immutable infrastructure ensures that the production environment matches the development and testing environments, reducing the risk of configuration drift and ensuring consistent security measures are applied throughout.

Policy definition and evaluation. With Docker Scout, you can take natural language requirements about software development standards and capture them in machine-readable code. Security teams and developers can work together to proactively ensure that software meets these standards as it is being developed, and not simply rely on being reactive when issues are detected in software that's already running in production. This prevents disruption to software development teams and increases development speed and developer satisfaction.

The Docker platform provides robust security features and capabilities. When leveraged correctly, enterprises can address these common security challenges and bolster their overall security posture. However, it is important to note that security is a shared responsibility. Developers should also implement secure container image practices, adopt least-privilege principles, regularly update and patch containerized applications, and implement proper network security measures to fully harness the security benefits of using Docker.

How else can Docker Scout be used to secure a software supply chain?

In addition to the vulnerability management and policy definition capabilities described in the prior section, Docker Scout provides a comprehensive solution for quickly remediating security issues. It can be integrated early into developer workflows, allowing for a more proactive approach to ensuring a secure build environment. It dissects intricate container images and transforms them into insightful and actionable data. With Docker Scout, developers gain a deeper comprehension of their image architectures all while avoiding reactive fixes later in the build process.

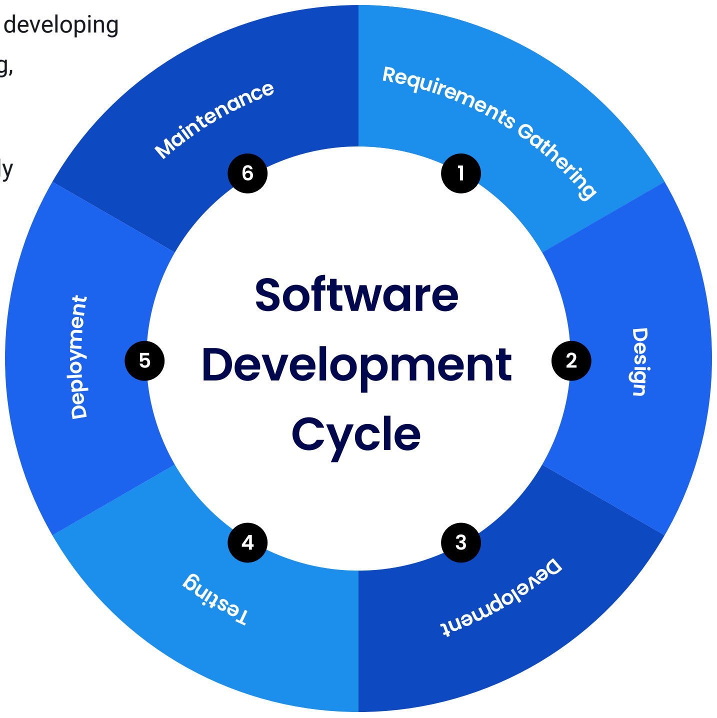
Images generally split into two categories: the base image and the application layers. Docker Scout excels in identifying vulnerabilities in the base images, a common source of issues and a critical component for developers to keep reliable. When a vulnerability surfaces within the application layers, Docker Scout also assists in identifying the affected file and its origin. Its design entails a unique layer-by-layer image analysis to optimize the discovery and remediation of vulnerabilities, grounded in the software's architecture. Rather than presenting a broad vulnerability scan, Docker Scout pinpoints the exact layer and origin of the issue. Docker Scout not only identifies these vulnerabilities but also recommends and facilitates the remediation to the most suitable base image.

Leveraging Docker Scout is a proactive security approach that empowers your developers to maintain compliance, protect sensitive data, and reduce the risk of costly security incidents. Overall, it delivers the necessary visibility for a secure container development journey.

Use Docker to simplify your software development lifecycle

The software development lifecycle (SDLC) is an iterative process of developing high quality software with stages that include requirements gathering, design, development, testing, deployment, and maintenance.

To build credible software, it is important for developers to continually assess their code and software artifacts for vulnerabilities throughout the entire SDLC, and that's where Docker provides solutions for devs who want to spend less time dealing with vulnerabilities and more time on innovation.



Secure your images today with Docker Scout, a seamless integration into your SDLC.