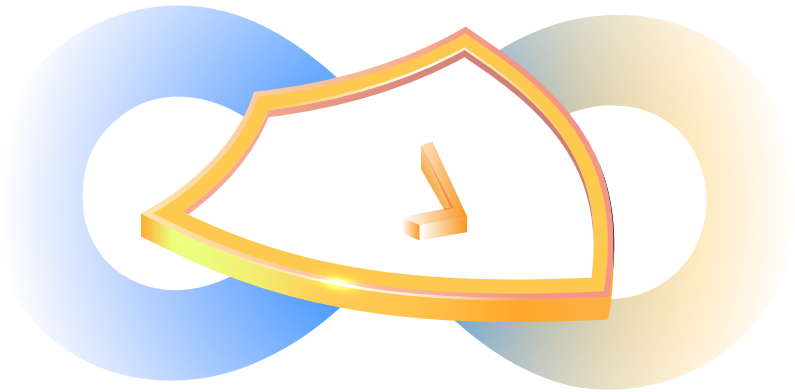
OpsMx

A Guide to Achieving

Secure Software Delivery

EBOOK

Table Of Contents



Why Software Attacks are Increasing and Real-Life Examples

In today's software delivery landscape, the stakes have never been higher. Take, for example, the SolarWinds attack, a devastating breach that exploited vulnerabilities in the software delivery process, resulting in severe consequences for the company and its customers. This attack exposed the urgent need for secure software delivery. Other notable attacks include the Kaseya ransomware attack, the Codecov unauthorized access incident, and the Log4j vulnerability. These real-life examples illustrate the potential impact of software supply chain attacks and highlight the criticality of implementing robust security measures.

Examples of attacks include:

Attack Name	Description	Root Cause	Business Impact
SolarWinds	Hackers compromised SolarWinds' IT monitoring system, Orion, and delivered backdoor malware in an Orion software update. The malware could access system files and work among SolarWinds' legitimate activities, going undetected even by antivirus software. Approximately 18,000 customers installed the malicious Orion update, allowing hackers to unleash even more malware on their systems.	Compromised software update	Affected organizations include Cisco, Deloitte, Intel, Microsoft, FireEye, and various government departments, including Homeland Security.
Kaseya	REvil hackers exploited a vulnerability in Kaseya's VSA software to carry out ransomware attacks on multiple managed service providers (MSPs) and their customers. They infiltrated systems via a fake update. The attack affected around 60 of Kaseya's customers and a further 1,500 businesses.	Exploited vulnerability in VSA software	Multiple managed service providers and their customers were affected.
Codecov	Hackers gained unauthorized access to Codecov's Bash Uploader script by altering it due to an error in the image creation process. They gleaned customers' private credentials, keys, and tokens for two months before detection.	Unauthorized access due to an error in image creation process	Customers' private credentials, keys, and tokens were compromised.
Log4j	The Log4Shell vulnerability in Log4j meant attackers could break into systems, steal data, uncover logins and passwords, and unleash additional malicious software. Log4j is used by many individuals and organizations, putting an extraordinary amount of users and businesses at risk of attack.	Log4Shell vulnerability	Affected organizations include Belgium's Ministry of Defence and Vietnam-based crypto platform Onus.

Table 1: Software Supply Chain attack examples

As digital innovation accelerates and the role of Artificial Intelligence (AI) expands, the risk of software supply chain attacks intensifies, presenting a significant challenge for organizations of all sizes. According to Gartner, by 2025, it is projected that 45% of organizations worldwide will have fallen victim to such attacks—a staggering three-fold increase from 2021.

Despite DevSecOps gaining traction as a practice, the path to its successful implementation remains fraught with complexities. It's important to note that DevSecOps is not a silver bullet. While it promises to integrate security into the software supply chain, ensuring that applications are secure is challenging, especially in the domain of software delivery.

Challenges Enterprises are Facing in Addressing this Problem

To underscore the gravity of the situation, let's enumerate the top five challenges that organizations face in achieving secure software delivery:

Fragmented Data. Security data is fragmented across teams and tools that don't work together. It is almost impossible to get a holistic view of the security posture of an application or understand the security impacts of a release.

Difficult to Make Good Security Decisions. When all of the relevant data is available, it is almost impossible for any person to evaluate the risks of a new release or to know if some critical step has been missed. There are too many attack vectors and unpredictable dependencies for humans to manage.

Manual Approvals and Policy Enforcement. "Pushing Left" application security moves responsibility for many security policies to the world's 200M+ developers. Organizations often add manual review and approval steps to make sure developers did what they are supposed to do. The volume of such checks at scale can quickly over any operations team.

Vulnerability Tracing and Response. Freezing code doesn't freeze the constantly changing vulnerability landscape. When a new vulnerability is discovered, there is no good way to know where that vulnerability is currently running in the production environment.

Manual Audits. Answering an audit or compliance request too often requires a manual review of system logs if they are still available.

Despite these challenges, it is critical to understand the undeniable importance of secure software delivery. It forms the bedrock of trust between businesses and their customers, safeguarding sensitive data and preserving the integrity of digital services. In an era where software vulnerabilities can lead to severe financial and reputational losses, secure software delivery is not merely a best practice; it's an absolute necessity.

There are many challenges in implementing secure software delivery, but they are not insurmountable. As we delve further into this guide, we will explore the principles, strategies, and best practices to help your organization achieve secure, continuous software delivery, building a robust defense against the cyber threats of today and tomorrow.

DevOps is not Optimized for Security

DevOps, with its focus on speed and continuous delivery, has revolutionized software development and deployment. However, the inherent nature of DevOps practices did not initially prioritize security. Instead, security was often an afterthought or a bolt-on issue that emerged as organizations realized the importance of protecting their software delivery pipelines.

The Continuous Delivery (CD) environment, designed for rapid and frequent updates, presents an attractive target for security attacks. The CD environment is particularly vulnerable to security risks. Here are some reasons why the CD environment is particularly ripe for security attacks and vulnerability issues:

Rapid Pace and Frequent Changes: CD environments are designed to deliver software updates quickly and frequently. This fast-paced nature can lead to oversight or neglect of security measures. The pressure to release new features or updates promptly may result in developers bypassing certain security checks or not thoroughly vetting changes, inadvertently introducing vulnerabilities.

Automation and Orchestration: CD relies heavily on automation and orchestration tools to streamline the software delivery process. While automation brings efficiency, it can also introduce security risks if not properly implemented. Attackers may attempt to exploit vulnerabilities in the automation scripts or manipulate the orchestrated processes to gain unauthorized access or introduce malicious code.

Integration of Third-Party Components: CD environments often incorporate third-party components, such as libraries, frameworks, and plugins, to accelerate development. However, these components may contain vulnerabilities that attackers can exploit. The challenge lies in managing and securing these dependencies, ensuring they are up-to-date and free from known vulnerabilities.

Access to Sensitive Data and Systems: The CD environment typically has access to sensitive data, including source code, credentials, and deployment configurations. Attackers target these environments to gain unauthorized access to valuable assets or leverage compromised CD processes to propagate attacks to production systems. A successful breach in the CD environment can have far-reaching consequences across the entire software supply chain.

Lack of Proper Security Measures: Historically, CD environments have focused primarily on speed and quality, with security being an afterthought. Security teams often operated separately from the CD pipeline, leading to delayed security checks or insufficient security measures. This disconnect creates opportunities for attackers to exploit undetected vulnerabilities until later stages or even production deployments.

Organizations must recognize the significance of secure software delivery and proactively protect their CD pipelines. By implementing robust security measures, they can significantly reduce the likelihood of security breaches, protect sensitive data, and uphold the trust and confidence of their customers. Embracing a security-first mindset and integrating security practices into every step of the software delivery lifecycle will pave the way for a more resilient and secure software ecosystem.

Attack Surface Vectors and Vulnerabilities in Continuous Delivery

The software supply chain supporting CI/CD consists of several stages, each playing a crucial role in the delivery process. These stages include code develop, commit, build, test, artifact management, deployment, and monitoring. While CI/CD initially focused on speed and quality, security was often overlooked as security teams operated outside the initial CI/CD environment. However, the dynamic and rapid nature of the CD environment exposes it to various security risks and makes it an attractive target for attackers.

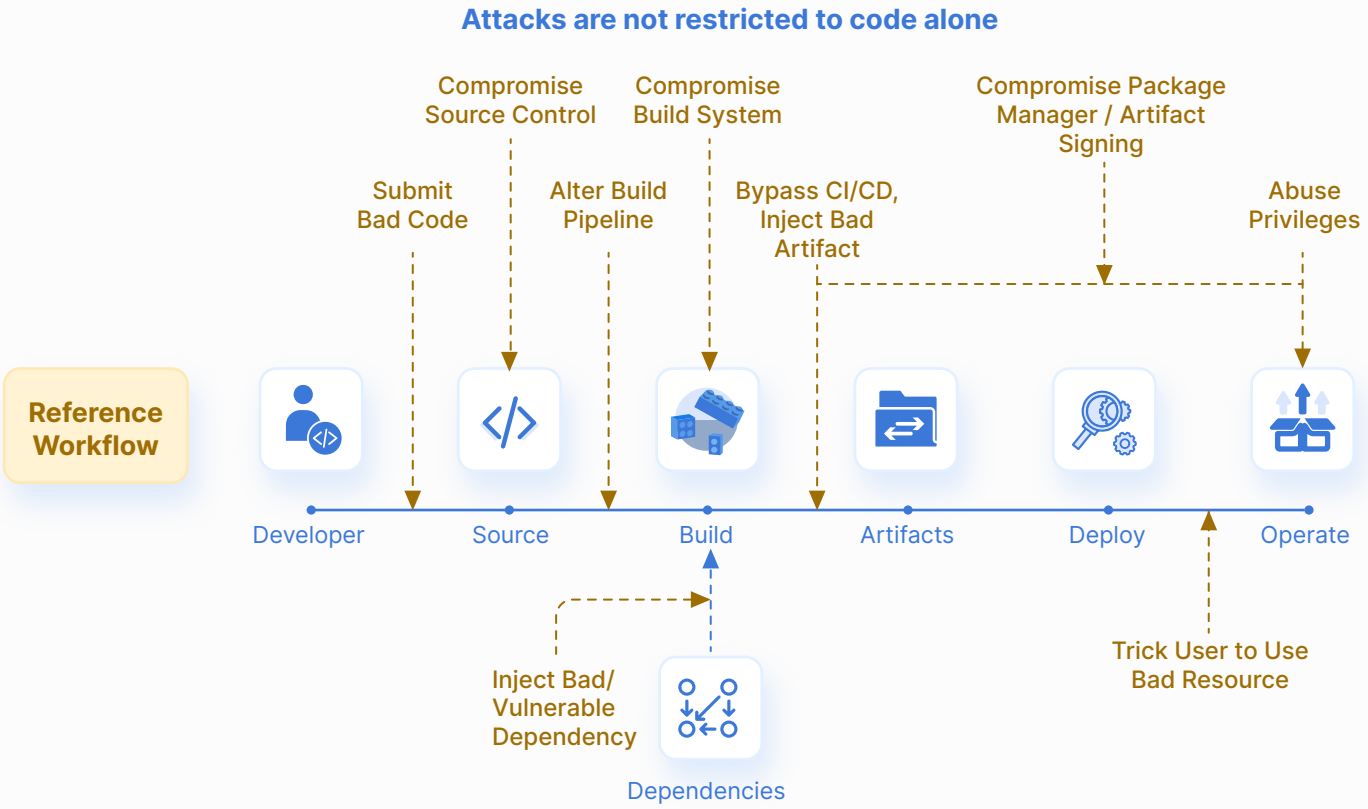


Image 1 - Software Supply Chain stages and security risks

Specific attack vectors and vulnerabilities can be exploited in each stage of the CD process. The post-build stage, in particular, becomes a focal point for Secure Software Delivery. Attack vectors within this stage include code injection, unauthorized access, tampering, insecure dependencies, and insider threats. These vectors can lead to compromised software artifacts, unauthorized access to Build servers or deployment environments, or the introduction of malicious code. Organizations must know these risks and take appropriate measures to secure their software supply chain.

Attack Vector	Description	Stage(s) of Occurrence	Past Examples
Code Injection	Injecting malicious code into the build process or software artifacts.	Build, Deployment	The Equifax data breach, where attackers injected malicious code into an Apache Struts library during the build process.

Unauthorized Access	Gaining unauthorized access to software artifacts, build servers, or deployment environments.	Artifact Management, Deployment	In the Capital One breach, an attacker gained unauthorized access to an S3 bucket containing customer data.
Tampering	Modifying software artifacts, configurations, or scripts during the build or deployment process.	Build, Deployment	The Magecart attacks, where attackers tampered with JavaScript files during the build process to inject credit card skimmers on e-commerce websites.
Insecure Dependencies	Exploiting vulnerabilities in third-party libraries or dependencies used in the software.	Build, Deployment	The Heartbleed vulnerability affected OpenSSL, a widely-used dependency in many software applications.
Insider Threats	Malicious insiders compromise the integrity or confidentiality of software artifacts.	Artifact Management, Deployment	The Edward Snowden incident, where a contractor leaked classified documents by accessing and copying them from the deployment environment.

Image 2 - Common attack vectors in the post-build stage

Emerging standards, such as NIST SP 800-218 and NIST SP 800-53, provide guidelines and controls for securing the software supply chain. These standards focus on supply chain risk management, ensuring software artifacts' integrity, confidentiality, and availability. Implementing these controls helps organizations establish a more secure and compliant software delivery process.

The Software Bill of Materials (SBOM) and the Supply Chain Levels for Software Artifacts (SLSA) model are gaining traction as mechanisms for enhancing supply chain security. The SBOM provides a complete inventory of software components used in an application, enabling organizations to track and manage dependencies effectively, but more is needed. The SBOM only captures software components, and security risks don’t stop once the software is built. The SLSA model defines different levels of assurance for software artifacts, guiding organizations in evaluating and assessing the security of their supply chain.

Integrated DevSecOps: Extending Security from Code to Cloud

While the concept of "Shift Left" has gained popularity in DevSecOps, it is essential to recognize that secure software delivery goes beyond just shifting security practices to the left. The last mile, from the post-build process to production, is an area that is often exposed and particularly vulnerable to security risks. To achieve a truly secure software delivery environment, organizations must adopt an integrated approach that extends security practices from code to cloud.

DevSecOps is not just about introducing security checks earlier in the software development lifecycle; it's about seamlessly integrating security throughout the entire delivery pipeline. By integrating security practices into each stage of the software delivery process, from code creation to deployment and beyond, organizations can establish a robust security posture that protects their software assets and data.

In the following sections of this eBook, we will explore key principles and best practices for secure software delivery. These principles encompass automating risk prevention, ensuring compliance throughout the pipeline, establishing end-to-end traceability, verifying deployment integrity, and implementing runtime monitoring and security. By adopting these principles and leveraging secure software delivery solutions, organizations can build a secure and resilient software delivery process that safeguards their applications, data, and reputation.

Key Principles of Secure Software Delivery

Principle 1: Automating Risk Prevention

Automating risk prevention is a fundamental principle of secure software delivery. Organizations can proactively identify and mitigate security risks by leveraging automation, ensuring that vulnerabilities and potential threats are addressed early in the software development lifecycle. In this section, we will explore the concept of automated risk prevention, the security risks in deployments, and the advantages it brings to the software delivery process.

Understanding the Security Risks in Deployments

Deployments are a critical stage in the software delivery pipeline and, unfortunately, an area where security risks can emerge. Attackers may exploit vulnerabilities, tamper with software artifacts, or gain unauthorized access during deployment. These risks pose significant threats to the software's integrity, confidentiality, and availability. Drawing on the prior section, we have identified attack vectors such as code injection, unauthorized access, tampering, insecure dependencies, and insider threats in the post-build and deployment stages.

Automated risk prevention in the post-build stage focuses on mitigating security risks and vulnerabilities that emerge during software delivery. Organizations can proactively identify and address potential threats by leveraging automated tools and practices before deploying the software.

For example, consider a scenario where a company is developing a web application. In the traditional approach, after the code is built, a manual code review is conducted to identify security flaws. However, this manual process is time-consuming and prone to human error, leading to potential vulnerabilities being overlooked.

With automated risk prevention, security tools are integrated into the CI/CD pipeline. These tools automatically analyze the application's code, dependencies, and configurations, flagging potential security issues such as code vulnerabilities, insecure dependencies, or misconfigurations. The system can enforce secure coding practices, conduct security assessments, and provide real-time alerts on identified risks.

By automating risk prevention post-build, organizations can mitigate various types of risks, including:

1. Code vulnerabilities: Automated code analysis detects common security flaws like injection attacks, cross-site scripting (XSS), and authentication bypasses.
2. Insecure dependencies: Vulnerability scanning tools identify known vulnerabilities in third-party libraries and components, enabling organizations to address them promptly.
3. Configuration weaknesses: Automated assessments check for misconfigurations in the application's environment, preventing potential security gaps.
4. Compliance violations: Organizations can ensure compliance with industry regulations and internal security policies by enforcing security checks.
5. Insider threats: Automated risk prevention systems monitor suspicious activities, helping detect and mitigate insider threats.

By integrating automated risk prevention in the post-build stage, organizations can significantly enhance the security of their software delivery process. They can identify and address vulnerabilities early on, reducing the chances of breaches and ensuring the deployment of more secure and resilient applications.

To effectively implement automated risk prevention, it is crucial to aggregate security and CI/CD tools and data from the existing DevOps toolchain. This integration enables the aggregation and automation of risk prevention measures across the stages of the software delivery lifecycle, ensuring comprehensive security coverage throughout the software delivery pipeline.

A Secure Software Delivery (SSD) automation solution acts as a central hub or control plane that integrates with various DevOps and security tools and processes, allowing for the seamless flow of security-related information and actions. It provides a unified view of the entire software delivery process, from code commit to deployment, and enables organizations to enforce consistent security practices across the different stages.

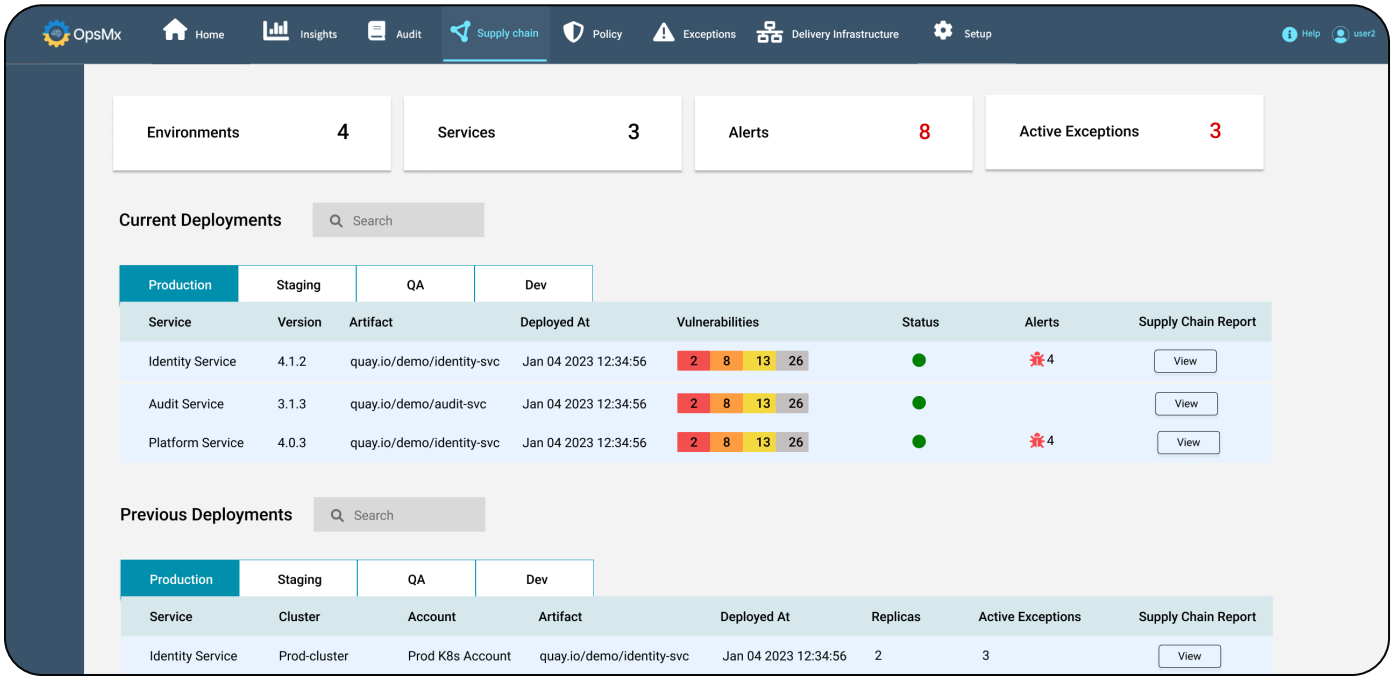


Image 2: Application security details

Here's how an SSD automation solution can integrate with a DevOps toolchain to aggregate and automate risk prevention across the stages of SSD:

1. Code Commit Stage:
    A. Integration with version control systems, such as Git, to enforce branch protection mechanisms and secure coding practices.
    B. Integration with code analysis tools to perform automated security scans and identify vulnerabilities at the earliest stage.
    C. Identify hardcoded secrets and vulnerabilities
2. Build and Continuous Integration (CI) Stage:
    A. Integration with CI/CD tools, such as Jenkins or GitLab CI/CD, to incorporate security checks into the build process.
    B. Implementation of secure code scanning tools to detect and flag potential security issues during the build stage.
    C. Integration with vulnerability management tools to ensure vulnerabilities are identified and prioritized for remediation.
3. Artifact Repository and Deployment Stage:
    A. Integration with artifact repositories, such as Nexus or JFrog Artifactory, to securely store and manage deployment artifacts.
    B. Implementation of mechanisms to verify the integrity and authenticity of deployment artifacts, including image integrity and signature verification.
    C. Composition analysis and known vulnerabilities checks for binaries/libraries in artifacts (Containers, VM Images).
    D. Automation of deployment audit records generation and storage for traceability and compliance purposes.

By integrating an SSD automation solution with the existing DevOps toolchain, organizations can achieve a centralized risk prevention solution that can enforce consistent security practices and policies across all stages of the software delivery lifecycle.

Advantages and Benefits of Automating Risk Prevention

Early Detection and Mitigation: Automation enables the identification of security risks at an early stage, often before the software is deployed to production. This early detection allows organizations to promptly mitigate risks, reducing the window of vulnerability and preventing potential breaches.

Consistency and Standardization: Automation ensures that security measures, such as secure coding practices, vulnerability scanning, and compliance checks, are consistently applied across the entire deployment process. This standardization reduces the chances of human error and ensures that security controls are in place consistently.

Scalability and Efficiency: Manual risk prevention can be time-consuming and error-prone, especially as software deployments become more frequent and complex. Automation allows for scalability and efficiency, enabling organizations to handle a high volume of deployments without compromising security.

Rapid Response to Emerging Threats: Automated risk prevention tools can quickly adapt to emerging threats and vulnerabilities. By leveraging threat intelligence and automated security updates, organizations can stay ahead of potential risks and apply timely remediation measures.

Integration with DevOps Practices: Automated risk prevention aligns with DevOps principles, integrating security seamlessly into the software delivery pipeline. It enables the implementation of security measures as code, incorporating security checks and controls into the CI/CD process.

Best Practices for Automating Risk Prevention

Best Practice	Description	Benefit
Implement Branch Protection and Secure Coding Practices	Enforce branch protection mechanisms and promote secure coding practices to prevent unauthorized changes	Ensures that only authorized changes are merged into production branches, reducing the risk of introducing vulnerabilities and unauthorized code
Conduct Proper Security Assessments and Vulnerability Management	Implement automated security assessments to continuously scan the codebase, dependencies, and infrastructure	Early detection and mitigation of security risks, reducing the window of vulnerability and preventing potential breaches
Address Vulnerable Dependencies based on Company Specific Policies	Establish company-specific policies for addressing vulnerable dependencies and automate their identification	Timely remediation of vulnerabilities in dependencies, reducing the risk of using insecure or outdated components, libraries, or frameworks

Ensure Secure Code Scanning and Library Change Detection	Integrate automated code scanning tools to detect and flag potential security issues during build and deployment	Early identification and prevention of security vulnerabilities, ensuring the software is free from common security weaknesses
Securely Store and Generate Deployment Audit Records	Implement mechanisms to securely store and generate audit records of deployment activities	Enhanced traceability and accountability in the software delivery process, facilitating compliance audits and incident investigations

Table 3: Best practices for automating risk prevention

By following these best practices, organizations can effectively automate risk prevention throughout the software delivery pipeline. Automated tools and processes enable early detection and mitigation of security risks, enforce secure coding practices, address vulnerabilities, and enhance the overall security posture of the software being delivered.

In the following sections, we will explore the remaining key principles of secure software delivery, including ensuring compliance throughout the pipeline, establishing end-to-end traceability, deployment verification and integrity, and runtime monitoring and security. These principles collectively contribute to a comprehensive and robust approach to secure software delivery, empowering organizations to build, deploy, and maintain software with confidence in its security and compliance.

Principle 2: Ensuring Compliance Throughout the Pipeline

Ensuring compliance throughout the software delivery pipeline is an essential principle of secure software delivery. Compliance with industry regulations, standards, and internal policies is critical to protect sensitive data, maintain customer trust, and mitigate legal and financial risks. In this section, we will explore the importance of ensuring compliance throughout the pipeline and discuss the benefits it brings to the software delivery process.

Understanding Compliance Risks in Software Delivery

The software delivery pipeline involves multiple stages, from development to deployment, and each stage introduces potential compliance risks. Non-compliance can result in severe consequences, such as regulatory penalties, reputational damage, and legal liabilities. Therefore, it is crucial for organizations to identify and address compliance risks throughout the pipeline, including security controls, data protection, and adherence to relevant industry standards and regulations.

The following table outlines the top compliance risks in software delivery, their descriptions, the stages where these risks typically occur, and examples of each risk. It provides an overview of the key compliance concerns organizations may face throughout the software delivery process.

Compliance Risk	Description	Stage	Example
Data Privacy and Protection	Risk of mishandling or unauthorized access to sensitive data	Throughout the pipeline	Non-compliance with GDPR, resulting in data breaches
Access Control	Risk of unauthorized access to systems, data, or functionalities	Development and deployment stages	Inadequate access controls leading to data leaks
Software Licensing Compliance	Risk of using unlicensed or improperly licensed software	Development and deployment stages	Unauthorized use of proprietary software, violating license terms

Regulatory Compliance	Risk of non-compliance with industry-specific regulations	Throughout the pipeline	Failure to comply with PCI DSS requirements in payment processing
Security Vulnerabilities	Risk of exploitable vulnerabilities in software or infrastructure	Development and deployment stages	Failure to address known vulnerabilities, leading to cyberattacks

Table 4: Top compliance risks in software delivery

As the software progresses through the pipeline, compliance risks can also emerge during testing and integration. It is essential to conduct thorough testing to ensure that the software meets functional requirements and adheres to industry standards and regulations. This includes validating data protection mechanisms, verifying the proper handling of personally identifiable information (PII), and performing penetration testing to identify potential vulnerabilities.

During deployment, organizations must focus on ensuring that the software is deployed in a compliant manner. This involves enforcing environment-specific security checks, monitoring compliance with industry regulations, tracking and managing vulnerability exceptions, and generating compliance reports and documentation. By automating these processes, organizations can streamline compliance efforts, reduce the risk of human error, and maintain an auditable trail of compliance activities.

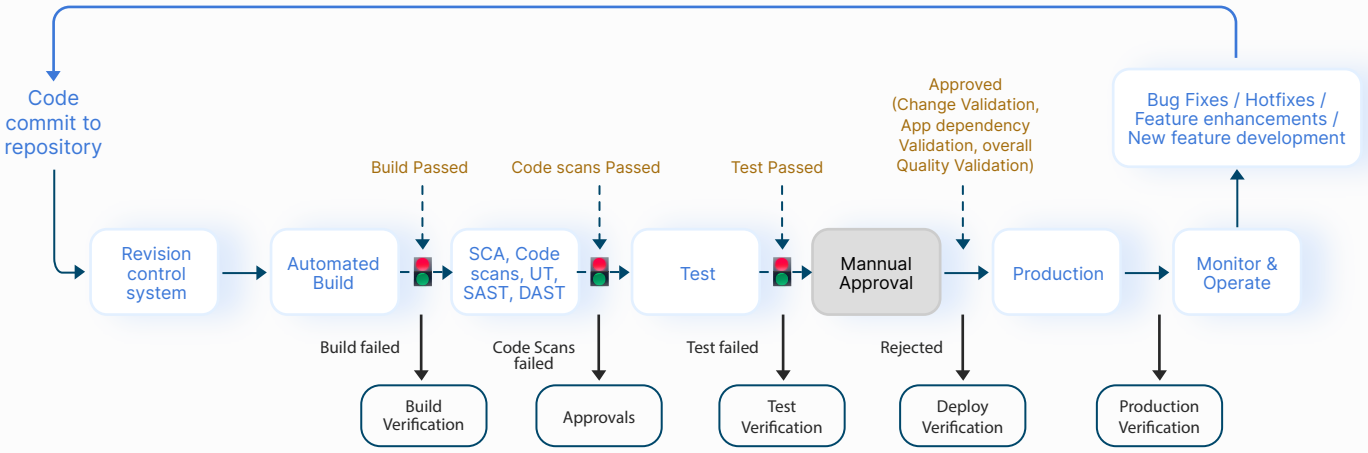


Image 3: Snapshot of security and compliance checks.

A concrete example of compliance risk in software delivery is the General Data Protection Regulation (GDPR). Organizations that handle personal data of European Union citizens need to ensure compliance with GDPR requirements throughout the software delivery process. This includes obtaining proper consent, implementing data protection measures, and providing individuals with the ability to exercise their data rights. Failure to comply with GDPR can lead to significant fines and reputational damage.

The Role of Automated Compliance Checks

In today's rapidly evolving regulatory landscape, organizations face increasing pressure to ensure compliance with many internal policies, industry standards, and government regulations. Manual compliance checks and verification processes are often time-consuming, error-prone, and challenging to scale as software delivery pipelines become more complex. As a result, organizations are turning to automated compliance solutions to streamline and enhance their compliance practices.

Automating compliance checks offers several advantages over manual processes. Firstly, it eliminates the potential for human error, ensuring consistent and accurate validation of compliance requirements throughout the software delivery lifecycle. Manual compliance checks can be prone to oversight, especially when dealing with many internal and external policies. With automation, organizations can ensure that no compliance aspect is overlooked, minimizing the risk of non-compliance.

Furthermore, automating compliance checks enables organizations to keep pace with the ever-changing regulatory landscape. As regulations evolve and new requirements emerge, manual processes struggle to adapt quickly. In contrast, automated solutions can be updated in real-time to incorporate the latest compliance standards, ensuring that organizations remain compliant at all times.

To effectively automate compliance checks across the software delivery lifecycle, organizations should leverage a comprehensive Secure Software Delivery Solution (SSD). An SSD solution integrates with the DevOps toolchain and employs a combination of policy-based automation, continuous monitoring, and risk assessment techniques to ensure compliance from development to deployment. Let's explore some examples of how a SSD solution can automate compliance checks at each stage:

Stage	Automated Compliance Checks
Development	• Security code analysis for identifying vulnerabilities • Enforcing coding best practices • License compliance verification for authorized software use
Testing	• Vulnerability assessment for identifying security weaknesses • Data privacy compliance checks for sensitive data protection
Staging	• Configuration management checks for adherence to policies • Environment-specific security checks for regulatory compliance
Deployment	• Security patch management for timely application of patches • Deployment audit and documentation for compliance reporting

Table 5: Compliance risk by stage

By implementing an SSD solution that encompasses these automated compliance checks, organizations can achieve numerous benefits. Firstly, it ensures consistent adherence to regulatory requirements and industry standards, mitigating the risk of non-compliance penalties and reputational damage. Additionally, it reduces the manual effort and time required for compliance checks, enabling teams to focus on more value-added activities.

Furthermore, automated compliance checks provide real-time visibility into compliance status and potential risks, empowering organizations to proactively address any issues. By continuously monitoring compliance throughout the software delivery lifecycle, organizations can promptly identify and rectify compliance gaps, minimizing the impact on operations and customer trust.

In conclusion, automated compliance checks are indispensable in today's regulatory environment. By leveraging an SSD solution to automate compliance checks across the software delivery lifecycle, organizations can ensure consistent adherence to regulatory requirements, industry standards.

Advantages and Benefits of Automating and Ensuring Compliance Throughout the Pipeline

Proactive Risk Mitigation: Automated compliance checks allow for proactive identification and mitigation of compliance risks throughout the pipeline. By detecting and addressing non-compliance issues early, organizations can minimize the potential impact and reduce the likelihood of regulatory violations.

Standardization and Consistency: Automation ensures that compliance checks are consistently applied across all stages of the software delivery process. This standardization reduces the chances of human error and ensures that compliance measures are uniformly implemented, regardless of the team or environment.

Regulatory Compliance and Avoidance of Penalties: Automated compliance monitoring and validation help organizations maintain adherence to industry-specific regulations, such as PCI DSS, HIPAA, GDPR, or other relevant frameworks. By avoiding regulatory penalties and legal liabilities, organizations can protect their reputation and financial stability.

Streamlined Audits and Reporting: Automated compliance tools simplify the audit process by generating comprehensive reports and documentation on-demand. This streamlines compliance audits and helps organizations demonstrate their adherence to regulatory requirements and industry standards.

Risk Mitigation in Vendor Management: Compliance throughout the pipeline extends beyond an organization's internal processes. It also involves managing compliance requirements for third-party vendors and service providers. By ensuring compliance in vendor management, organizations can mitigate risks associated with external dependencies.

Best Practices for Ensuring Compliance Throughout the Pipeline

Best Practice Name	Description	Benefit
Enforce Environment-Specific Security Checks	Implement automated security checks tailored to each environment, application, or pipeline stage to ensure compliance with relevant standards.	Ensures that security measures are consistently applied at each stage, reducing the risk of vulnerabilities and ensuring compliance with regulations.
Monitor Compliance with Industry Regulations	Utilize automated monitoring tools to regularly assess compliance with industry regulations, standards, and internal policies.	Provides ongoing visibility into compliance status, enabling organizations to identify and address any deviations promptly, reducing compliance risks.
Track and Manage Vulnerability Exceptions	Establish a centralized system to track and manage vulnerability exceptions, documenting identified risks and prioritizing them for remediation.	Enables organizations to effectively track and prioritize vulnerability remediation efforts, reducing the overall risk exposure and enhancing security.
Automate Compliance Reporting and Documentation	Implement automated reporting mechanisms to generate compliance reports on-demand, facilitating the demonstration of adherence to standards.	Streamlines the process of generating compliance reports, saving time and effort while ensuring accurate and up-to-date documentation for regulatory purposes.

Establish Continuous Compliance Monitoring	Integrate continuous compliance monitoring into the software delivery pipeline to detect and address deviations in real-time.	Provides real-time visibility into compliance deviations, allowing organizations to proactively identify and address compliance issues as they arise.

Table 6: Best practices for ensuring compliance throughout the pipeline

By implementing these best practices and leveraging automation to ensure compliance throughout the pipeline, organizations can reduce compliance risks, avoid penalties, and maintain a robust security posture in their software delivery process. The combination of targeted security checks, ongoing monitoring, effective vulnerability management, streamlined reporting, and continuous compliance monitoring contributes to a more secure and compliant software delivery environment.

In the following sections, we will explore the remaining key principles of secure software delivery, including establishing end-to-end traceability, deployment verification and integrity, and runtime monitoring and security. Each principle contributes to a comprehensive approach to secure software delivery, emphasizing the importance of security and providing actionable insights for implementation.

Principle 3: Establishing End-to-End Traceability

Ensuring traceability throughout the software delivery pipeline is a fundamental principle of secure software delivery. Traceability gives organizations complete visibility and accountability, allowing them to track and manage various aspects of the software development and delivery process. In this chapter, we will delve into the importance of establishing end-to-end traceability and explore how it enhances the security and reliability of software delivery.

Understanding the Significance of End-to-End Traceability

In the complex landscape of software delivery, traceability plays a vital role in ensuring that every step of the process is documented and transparent. It enables organizations to track critical information, such as image provenance, build details, dependency differences, and vulnerability tracking, throughout the software delivery lifecycle.

End-to-end traceability enables organizations to identify the origins and history of software artifacts, including container images, libraries, and dependencies. This information is crucial for evaluating the security and integrity of the software supply chain, mitigating risks associated with third-party components, and ensuring compliance with industry standards and regulations.

Establishing End-to-End Traceability

Traceability in the context of software delivery refers to the ability to track all activities across the delivery lifecycle, including changes to code, environment configurations, testing outcomes, and deployment details. It's like a breadcrumb trail of actions that leads to the final software product. Having complete traceability is crucial for various reasons, such as incident management, auditing and compliance, debugging and troubleshooting, and improving software quality.

However, maintaining manual traceability can be challenging. It involves recording, maintaining, and managing a vast amount of data across different tools and stages of the software delivery lifecycle. This becomes increasingly difficult as the complexity of the software, tools, and processes increases. Manual methods also often result in gaps in the data trail due to human error, making it difficult to reconstruct the exact path followed in software development and delivery.

To overcome these hurdles, organizations need an automated, real-time traceability solution, a central feature of Secure Software Delivery (SSD) systems. An effective SSD solution operates as a hub or control plane, automatically tracking every step of the software delivery process and instantly updating records. By leveraging SSD solutions, organizations minimize human error, rapidly identify and address issues that arise, and streamline compliance with security and regulatory requirements.

A key aspect of real-time traceability facilitated by SSD solutions is the enforcement of 'Separation of Duties'. This concept is an essential internal control in secure software delivery. It helps prevent fraud and error by ensuring that at least two individuals are responsible for separate parts of any critical task. By clearly defining and enforcing these separate responsibilities, an SSD solution promotes accountability and security throughout the software delivery lifecycle.

Examples of traceability changes that are key to each stage of software delivery:

Stage	Traceability Changes
Code Changes	Who made changes, what changes were made, why were the changes made
Testing	What tests were run, the results of the tests, and who ran the tests
Deployment	What versions were deployed, when they were deployed, and who deployed them
Environment Configuration	Changes in environment configurations, reasons for changes, and who made these changes
Incident Management	Details of the incident, how it was resolved, who was involved in resolving it

Table 7: Traceability changes by stage

Implementing automation and real-time traceability throughout the software delivery lifecycle not only helps maintain a secure and compliant environment but also optimizes the software delivery process, ensuring high-quality and reliable software products.

Tracking Image Provenance

Image provenance is a critical aspect of the DevSecOps world. It involves understanding the origin, composition, and history of every software artifact within a container image or software package. The provenance of an image helps to identify and trace the origins of all components within it, including base images, libraries, and dependencies. Comprehensive visibility into image provenance is paramount for evaluating the security and integrity of the software supply chain and pinpointing potential risks linked to third-party components.

To comprehend image provenance, there are essential questions we need to answer for every image in our cluster: What is it? Where did it come from? How can I rebuild it? Does it have any known vulnerabilities? Is it up-to-date?

Let's explore these:

1. What is it? This refers to identifying every component within the image, along with its version and purpose. It helps in understanding the functionality and the role of each component in the overall application.

2. Where did it come from? Tracing the origin of every component aids in assessing its trustworthiness. Components from unknown or unverified sources pose security risks, while those from reputable sources generally follow best practices and are more likely to be secure.
3. How can I rebuild it? A critical aspect of modern software delivery is the ability to recreate the application from scratch using the source code and dependencies. This capability is vital for maintaining the software's longevity and adaptability.
4. Does it have any known vulnerabilities? Tracking vulnerabilities in all components is critical to maintaining a secure software environment. This involves scanning the components for known vulnerabilities and assessing their impact on the application.
5. Is it up-to-date? Software components need to be kept up-to-date to leverage improvements and security patches. Using outdated components can expose the application to known vulnerabilities and compromise performance.

But the question remains, can we prove these answers? To this end, organizations must employ tools and technologies that enable image scanning and analysis. These tools help to automatically identify and assess the security posture of individual components within an image, allowing for proactive risk mitigation and ensuring that only trusted and verified components are included in the final software deliverables.

For instance, consider a scenario where a software delivery team uses an open-source library in their application. The team should be able to trace the library's origin, understand its functionality, identify its version, determine any known vulnerabilities, and confirm that it is the latest version. If the library has known vulnerabilities, the team can take immediate action, such as updating the library or using an alternative.

Best Practices for Image Provenance

Let's look at the top five best practices for image provenance: In essence, tracking image provenance is crucial for secure software delivery. By implementing best practices, we can ensure we know what's happening in our cluster, enhancing security, transparency, and accountability in our software delivery process.

Best Practice	Description	Stage	Benefit
Component Identification	Ensures each component within the image, its version, and its purpose is clearly understood.	Build	Provides an understanding of the role and functionality of each component.
Origin Tracking	Traces the origin of every component in the software delivery to assess its trustworthiness.	Build and Deployment	Assesses the trustworthiness of components and mitigates potential security risks.
Rebuild Capability	Ensures the ability to recreate the application from scratch using the source code and dependencies.	Continuous Integration	Maintains software longevity and adaptability by ensuring software can be reliably built from source code.
Vulnerability Tracking	Involves scanning the components for known vulnerabilities and assessing their impact on the overall application.	Continuous Integration and Continuous Deployment	Ensures a secure software environment by proactively identifying and addressing security vulnerabilities.
Component Updating	Keeping software components up-to-date to leverage improvements and security patches.	Continuous Deployment	Leverages improvements and security patches, and reduces exposure to known vulnerabilities.

Table 8: Best practices for image provenance

In essence, tracking image provenance is crucial for secure software delivery. By implementing best practices, we can ensure we know what's happening in our cluster, enhancing security, transparency, and accountability in our software delivery process.

Tracking Build Details, Dependency Differences, and Vulnerability Tracking

Tracking build details, dependency differences, and vulnerability management are three cornerstones of end-to-end traceability, providing an indelible record of the software delivery lifecycle. They are essential for any organization aiming to enhance security, compliance, and resilience within its delivery pipeline.

When we delve into build details, we are talking about the fine specifics of the software building process. These include aspects like the build configurations, the environment variables in use, and the creation of build artifacts. By capturing these details, organizations gain the ability to recreate builds with precision, dissect and investigate issues with detailed information, and ensure build consistency across different environments, thus mitigating software delivery risks.

But how can we stay on top of the changing dependencies that could introduce vulnerabilities to our software? That's where the concept of tracking dependency differences comes in. Automated tools are pivotal in comparing the dependencies utilized during different stages of software delivery, such as development, testing, and deployment. They can spotlight any deviation or vulnerabilities that may have been accidentally introduced, thus enabling proactive security measures.

This leads us to vulnerability tracking. By integrating vulnerability management tools into the CI/CD pipeline, organizations can continuously monitor and identify potential vulnerabilities across their software stack. This process not only allows for timely remediation but also supports the maintenance of a secure and updated software ecosystem.

The importance of the Software Bill of Materials (SBOM) becomes clear in this context. An SBOM is an exhaustive record detailing each software component in a product. Created during the build stage, it is a crucial tool in identifying and managing the dependencies that a software product has. This is because it lists all the details of every software piece, including the version number, the licensing, and the library dependencies it has.

Consider, for instance, a use case where a vulnerability has been discovered in a widely used open-source library. With an SBOM, organizations can quickly assess if any of their applications are using the compromised library and, if so, which versions are affected.

This allows them to swiftly undertake remediation actions, minimizing the window of exposure to the vulnerability.

The SBOM is not just about tackling vulnerabilities, though. It can also provide insights into the licensing obligations associated with each software component, helping to maintain compliance with legal requirements. With these facets, an SBOM becomes a crucial element in pursuing end-to-end traceability, providing a clear view of what’s in the software, enhancing the security posture, and ensuring compliance.

Ensuring Transparency and Accountability in the Software Delivery Process

End-to-end traceability fosters transparency and accountability in the software delivery process. By capturing and documenting relevant information at each stage, organizations can maintain a comprehensive audit trail that records activities, decisions, and changes made throughout the pipeline.

This audit trail serves multiple purposes. Firstly, it enables organizations to comply with regulatory requirements and industry standards by providing evidence of adherence to established processes and policies. Secondly, it facilitates incident investigations and problem resolution by providing a historical record of activities and changes that occurred during software development and delivery. Lastly, it promotes accountability and collaboration among teams, as individuals can be held responsible for their actions and decisions throughout the software delivery lifecycle.

Principle 4: Deployment Verification and Integrity

As we continue discussing the key principles of secure software delivery, we arrive at the fourth principle, focusing on establishing deployment verification and integrity. This stage involves confirming the integrity of the deployment artifacts, generating and securely storing audit records, and performing code scanning and library change detection. Let's delve deeper into these facets and how they contribute to a secure software delivery pipeline.

One of the most notable examples of the importance of deployment verification and integrity is the SolarWinds supply chain attack in 2020. We mentioned this attack in the introduction, but as a reminder, In this breach, threat actors managed to infiltrate the software build system of SolarWinds, a provider of IT management software, and insert malicious code into one of their products. This manipulated software was then distributed to about 18,000 customers worldwide.

The malicious code provided backdoor access to the affected systems, leading to massive breaches at several high-profile targets, including government agencies and private companies. Although the specific details of how the attackers achieved this are complex and multifaceted, the case serves as a stark reminder of the importance of ensuring integrity throughout the entire software delivery lifecycle, including during deployment.

If rigorous deployment verification measures had been in place, including automated scanning of software artifacts, more rigorous checking of the integrity of code and dependencies, and regular auditing of deployment records, the manipulated software could have been detected earlier, and the scale of the breach might have been significantly reduced. It’s critical to comprehensive record of all security, compliance, and validation actions taken throughout the software delivery process.

Introducing the Delivery Bill of Materials

As a preface, it's crucial to acknowledge the limitations of the Software Bill of Materials (SBOM). While the SBOM provides critical insights into the software's components, it only covers up to the build process. Security and traceability, however, don't end at the build. They extend to the deployment and execution stages, hence the need for a more encompassing construct: the Delivery Bill of Materials (Delivery BOM).

The Delivery BOM extends the SBOM concept to include deployment metadata and artifacts, providing visibility and traceability not just into the construction of the software but also it's delivery.

By extending traceability into the deployment process, the Delivery BOM can help verify the integrity of deployed components and prevent unauthorized or compromised components from reaching production environments.

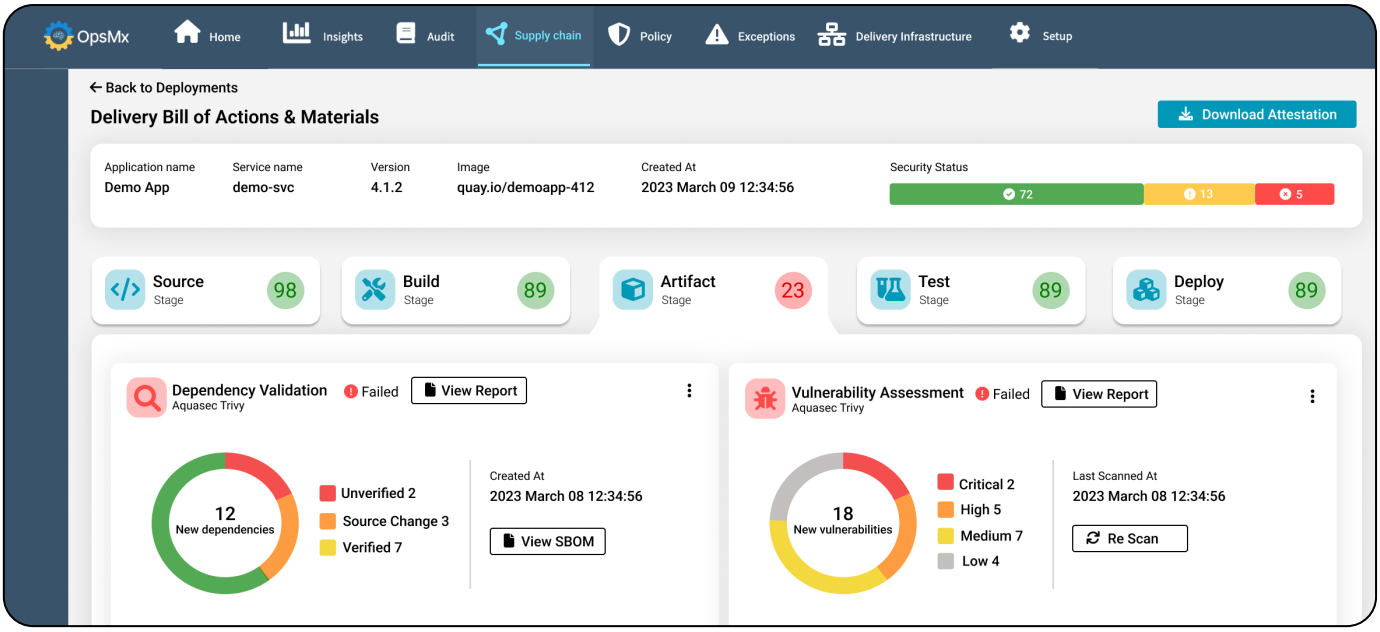


Image 4: Delivery Bill of Actions and Materials

Delivery Bill of Materials (DBOM)

The Delivery Bill of Materials serves as a comprehensive record of all security, compliance, and validation actions taken throughout the software delivery process. It encompasses various stages, including source code, build validation, artifact validation, and deployment validation.

Stage	Delivery BOM Role	Benefit
Source Code	Includes SAST/DAST checks, branch protection, code review validation	• Ensures that source code meets security standards • Identifies and mitigates security vulnerabilities early in the development process • Validates proper branch protection and code review processes
Build Validation	Validates build system, CI configurations, unit test coverage	• Verifies the integrity and reliability of the build • Ensures that proper CI configurations are followed • Validates adequate unit test coverage to maintain code quality
Artifact Validation	Checks dependencies and performs vulnerability assessment	• Identifies and mitigates security risks associated with dependencies • Ensures the use of secure and up-to-date software components
Deployment Validation	Involves CIS benchmark checks, deployment performance validation, approval validation	• Verifies compliance with security benchmarks and industry best practices • Ensures optimal deployment performance • Validates necessary approvals and adherence to release processes

Table 9: Delivery BOM role by stage

One key distinction between the Delivery Bill of Materials and the Software Bill of Materials (SBOM) is that the DBOM provides a comprehensive view of the entire deployment, whereas the SBOM is focused on software dependencies. The DBOM consolidates data from the entire delivery process, including security and vulnerability checks, static and dynamic code analysis, unit test coverage, source code branch validation, and deployment environment information.

Delivery Bill of Materials vs. Software Bill of Materials

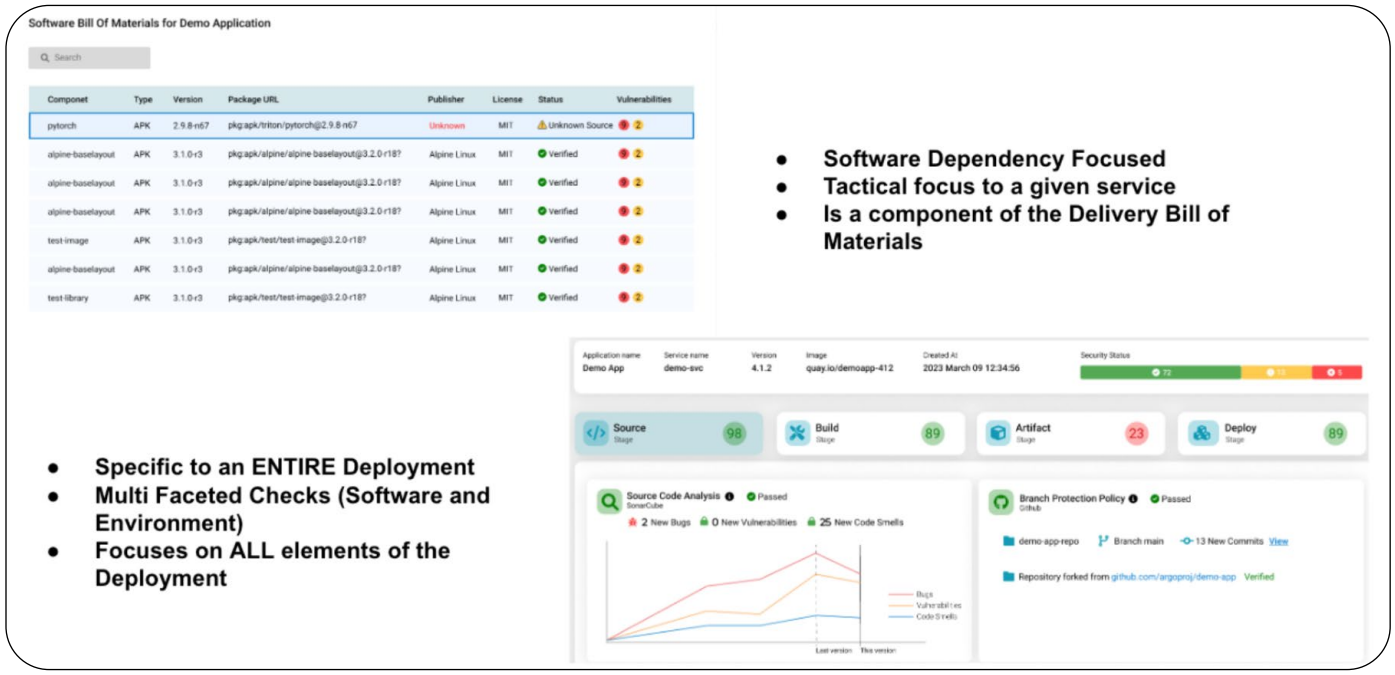


Image 5: Difference between SBOM and DBOM

The Delivery Bill of Materials plays a critical role in security and compliance. It ensures that all necessary checks and policies are followed throughout the software delivery process, reducing the risk of security breaches and compliance violations. By consolidating data from the entire delivery process, the DBOM provides a holistic view of the deployment's security posture and compliance status, enabling organizations to assess and address any issues.

For DevOps and DevSecOps teams, the Delivery Bill of Materials offers valuable insights and benefits. DevOps teams can gain a real-time view of the changing delivery landscape, understand the necessary actions during the delivery process, and document and provide audit information for each deployment. This visibility enables efficient collaboration and informed decision-making among team members.

On the other hand, the Delivery Bill of Materials empowers security teams with a comprehensive view of the application deployment's security posture. It consolidates security reporting across the entire CI/CD ecosystem, simplifying security audits and providing security attestation. Real-time gating of releases based on Delivery Bill of Materials rules ensures that only compliant and secure software is deployed.

To build a Delivery Bill of Materials, data collection and consolidation are key. A Secure Software Delivery (SSD) solution is needed to tap into the entire ecosystem and collect relevant information from various stages of the software delivery process. This includes integrating with build systems, CI pipelines, security scanning tools, vulnerability management platforms, and deployment monitoring systems. By centralizing and consolidating this data, organizations can construct a comprehensive Delivery Bill of Materials that provides a complete view of the security and compliance aspects of their software deployments.

Delivery BOM Benefits

Ensures all necessary checks and policies are followed

Reduces the risk of security breaches and compliance violations

Provides a holistic view of the deployment's security posture and compliance status

Facilitates collaboration and decision-making for DevOps and DevSecOps teams

Simplifies security audits and provides security attestation

Enables real-time gating of releases based on Delivery BOM rules

In summary, the Delivery Bill of Materials goes beyond the Software Bill of Materials to encompass the entire software delivery process. It ensures traceability, integrity, and compliance throughout the deployment stages, providing valuable insights for both DevOps and DevSecOps teams. By leveraging the Delivery Bill of Materials, organizations can enhance their security posture, streamline compliance efforts, and deliver software with confidence.

Best Practices for Establishing Deployment Verification and Integrity

Let's now delve into the best practices for establishing deployment verification and integrity, each of them a significant contributor to the security of your software delivery pipeline:

Best Practice	Description	Benefit	Implementation Tips
Delivery Bill of Materials	A comprehensive accounting of all security, compliance, and validation actions taken in association with a software deployment.	Provides end-to-end visibility and traceability of the software delivery process.	- Leverage an SSD solution to establish a standardized framework for capturing and documenting security, compliance, and validation activities throughout the delivery process. - Include details such as source code checks, build validation, artifact validation, deployment validation, and other relevant information.
Image Signature Verification	Checking the digital signatures of images before deployment.	Ensures image integrity and prevents tampered images from being deployed.	- Utilize cryptographic algorithms to generate and verify digital signatures. - Implement a secure repository to store signed images and keys.
Deployment Audit Records	Creating and securely storing logs of all deployment activities.	Supports accountability and provides critical information for forensic investigations.	- Implement a centralized logging system to capture deployment activities. - Encrypt and protect audit records to maintain their integrity.
Secure Code Scanning	Automated scanning of the source code for security vulnerabilities.	Detects and fixes security issues early in the pipeline.	- Integrate static code analysis tools into the build process. - Configure regular scans to ensure ongoing security assessment.

Library Change Detection	Tracking changes in library dependencies between different versions.	Detects introduction of insecure or out-of-date libraries, enabling proactive risk mitigation.	- Utilize dependency management tools to track library changes. - Regularly update libraries and address security vulnerabilities promptly.
Runtime Environment Validation	Validating the security posture of the runtime environment before deployment.	Prevents deploying into insecure environments that could expose the software to risks.	- Define security standards for the runtime environment. - Conduct thorough security assessments and vulnerability scans of the runtime environment.

Table 10: Best practices for deployment verification

In conclusion, deployment verification and integrity involve a broad array of practices and concepts. From validating image signatures and code scans to maintaining comprehensive audit records and tracking library changes, these activities form a robust defense against security vulnerabilities. By extending the concepts of SBOM with the Delivery BOM, organizations can ensure end-to-end visibility, integrity, and security throughout the software delivery lifecycle.

Principle 5: Runtime Monitoring and Security

The software delivery journey doesn't end at deployment. Secure Software Delivery (SSD) must incorporate active monitoring and security management during software execution or runtime. This involves watching for out-of-policy actions, increasing the incident response capabilities, and securing sensitive data and communications. This fifth principle of secure software delivery underscores the need to be vigilant even after the software delivery and to detect and respond to any potential threats in real time.

The concept of runtime security monitoring isn't just about observing the behavior of software in the execution environment. It's about the constant verification of the state of your software and systems, checking for potential breaches, intrusions, or anomalies that could signal security risks. It implies the use of sophisticated technology such as AI/ML to constantly evaluate the integrity of running software and systems, compare behavior against known policies and procedures, and flag or respond to any deviations.

Runtime monitoring allows organizations to detect and respond to out-of-policy actions, proactively identify security threats, and ensure the integrity and confidentiality of sensitive data.

Importance of Runtime Monitoring and Security

Runtime monitoring and security play a pivotal role in identifying and addressing security threats that may arise during the execution of software. While production security measures such as DDoS protections, perimeter security, and zero trust protections for privilege escalation are crucial, runtime monitoring focuses on monitoring for changes in the runtime environment and detecting any deviations or anomalies that may indicate a security breach or unauthorized access. By automating this principle through a Secure Software Delivery (SSD) solution, organizations can proactively detect and respond to security incidents, ensuring the overall security and integrity of their applications.

How Runtime Monitoring and Security Works

Runtime monitoring and security involve implementing mechanisms to continuously monitor the runtime environment, detect security threats, and respond effectively. This includes

1. Runtime Verification: Implementing runtime verification mechanisms to monitor the behavior and integrity of the software during execution. This can involve monitoring for out-of-policy actions, detecting unauthorized access attempts, and identifying any anomalous behavior that may indicate a security incident.

2. Incident Response Enhancements: Strengthening incident response capabilities by integrating the SSD solution with incident response processes. This ensures that security incidents detected during runtime monitoring are promptly addressed and mitigated. Incident response workflows and playbooks should be defined to align with the capabilities of the SSD solution, enabling effective collaboration between security and incident response teams.
3. Secure Data and Communications: Implementing robust measures to secure sensitive data and communications within the runtime environment. This includes encryption of data at rest and in transit, secure communication protocols, and access controls to protect sensitive information from unauthorized access or disclosure.
4. Drift Detection: Monitoring for changes in the runtime environment, such as configuration drift or changes in binaries. This helps identify potential security vulnerabilities or unauthorized modifications that may compromise the integrity of the software.

The essence of runtime monitoring and security is vigilance. By implementing runtime verification, enhancing incident response, and securing data, organizations can ensure that their software remains secure throughout its lifecycle. An SSD solution that provides these capabilities will play an invaluable role in protecting your software assets in the dynamic, threat-filled landscape of today's digital world.

Best Practices for Runtime Monitoring and Security

Implementing robust runtime monitoring and security practices is crucial for maintaining a secure software environment. To achieve this, organizations should adopt a set of best practices that focus on continuous monitoring, integration with security tools, threat intelligence, automation of incident response, and proactive vulnerability management. The following best practices provide a comprehensive approach to runtime monitoring and security, ensuring the integrity, confidentiality, and availability of software systems throughout their lifecycle.

Best Practice	Description	Benefits	Implementation Tips
Continuous Monitoring	Implement continuous monitoring mechanisms for real-time event capture and analysis.	- Proactive threat detection and timely incident response. - Enhanced visibility into runtime environment activities.	- Utilize automated monitoring tools to capture and analyze runtime events in real-time. - Define alert thresholds and configure notifications.
Security Information and Event Management (SIEM) Integration	Integrate the SSD solution with a SIEM system for centralized aggregation and analysis of runtime security events.	- Enhanced detection and investigation of security incidents. - Improved incident response capabilities.	- Ensure proper integration between the SSD solution and the SIEM system. - Configure event forwarding and log collection mechanisms.
Threat Intelligence Integration	Integrate threat intelligence feeds and services to enhance threat detection capabilities.	- Early detection and mitigation of known threats. - Improved security posture against evolving threats.	- Establish integration with threat intelligence platforms or services. - Implement automated threat intelligence analysis.
Incident Response Automation	Automate incident response processes and workflows for prompt and consistent response to security incidents.	- Enables rapid and consistent incident response, reducing response times and potential impact.	- Define incident response playbooks and workflows aligned with the capabilities of the SSD solution. - Automate common response actions and escalation procedures.

Vulnerability Patching and Remediation	Implement a proactive vulnerability management program to regularly patch and remediate vulnerabilities in the runtime environment.	- Reduces the risk of exploitation of known vulnerabilities. - Strengthens the overall security posture.	- Implement vulnerability scanning and assessment tools. - Establish a process for prioritizing and addressing vulnerabilities based on severity.

Table 11: Best practices for runtime monitoring and security

In conclusion, runtime monitoring and security are paramount in ensuring the robustness and protection of software systems. By implementing runtime verification, enhancing incident response capabilities, and securing sensitive data and communications in runtime environments, organizations can proactively detect and respond to out-of-policy actions, mitigate security risks, and maintain a secure software ecosystem. Leveraging automation and following best practices for runtime monitoring and security enable organizations to fortify their defenses, stay ahead of potential threats, and ensure the integrity and reliability of their software throughout its execution.

Conclusion

In conclusion, secure software delivery is crucial in today's complex and dynamic software landscape. With numerous tools, processes, and teams involved, it becomes essential to adopt a holistic approach that integrates all the disparate data and components. This ebook has outlined key principles and best practices to address the challenges and risks associated with the software supply chain.

We highlighted the importance of automating risk prevention, ensuring compliance throughout the pipeline, establishing end-to-end traceability, and verifying deployment integrity. Additionally, we emphasized the significance of implementing runtime monitoring and security measures to detect and respond to out-of-policy actions, enhance incident response capabilities, and secure sensitive data in runtime environments.

Furthermore, we introduced the concept of the Delivery Bill of Materials (DBOM) as a crucial component that extends beyond the build process. The DBOM provides comprehensive accountability and traceability of security and compliance actions throughout the software delivery process, ensuring that security measures are not limited to the build stage alone.

About OpsMx

At OpsMx, we offer a Secure Software Delivery (SSD) solution designed to help organizations implement these principles and best practices seamlessly. With OpsMx SSD, you can streamline your software delivery processes, automate security checks, ensure compliance, and enhance visibility and accountability.

Now is the time to take action and explore OpsMx SSD. Our team is ready to provide a demo and assist you in implementing secure software delivery practices. Embrace a proactive approach to secure software delivery and safeguard your software supply chain. Reach out to us today!

FOR MORE INFORMATION, CONTACT US - OPSMX, INC | 350 OAKMEAD PKWY, SUNNYVALE, CA 94085 | INFO@OPSMX.COM | WWW.OPSMX.COM

OpsMx