Science and Technology 2024, 14(2): 31-40
DOI: 10.5923/j.scit.20241402.03

Software Supply Chain Security- Core Systems and Security Guidelines

Anupam Mehta[1,*], Davinder Pal Singh[2], Viplove Karhade[3], Preyansh Matharoo[4]

[1]Product Security, Stripe, Ashburn, USA
[2]Technical Architect, Salesforce, Toronto, Canada
[3]Product Security, Salesforce, Aldie, USA
[4]Product Security, Salesforce, San Francisco, USA

Abstract Securing the software supply chain has become crucial in today's development environment, where vulnerabilities can severely impact organizational integrity and data security. This paper provides a framework for strengthening key components of the software supply chain, including Source Code Management (SCM) systems, Continuous Integration (CI) systems, Continuous Deployment (CD) systems, and artifact storage. For each component, we outline core security practices, such as architectural design, access management, logging, and monitoring. These measures provide actionable steps for mitigating risks, protecting intellectual property, and maintaining compliance with industry standards, ultimately enhancing the integrity and reliability of software delivery pipelines. Implementing these security practices reinforces the software supply chain's resilience against evolving threats, enabling organizations to deliver reliable, secure software.

Keywords Software Supply Chain Security, Source Code, Continuous Integration and Deployment

1. Introduction

As software supply chains grow more interconnected, they become increasingly vulnerable to security breaches. Each software lifecycle phase, from code development to deployment, presents unique risks that attackers can exploit. Key components in this ecosystem include Source Code Management (SCM) systems, Continuous Integration (CI) systems, Continuous Deployment (CD) pipelines, and artifact storage solutions. Each of these systems plays a vital role in delivering software securely and reliably but also introduces potential entry points for threat actors. This paper explores these core components, detailing security challenges and presenting comprehensive recommendations designed to mitigate risks and enhance the resilience of the software supply chain.

2. Software Supply Chain Overview

The software supply chain encompasses all stages involved in developing, testing, and deploying software, similar to a manufacturing process where each part must operate securely to ensure a quality final product. This paper focuses on critical systems—SCM, CI, CD, and artifact storage—that streamline the delivery of updates and fixes. However, each system can serve as an entry point for attackers, underscoring the need for robust security practices across the entire chain.

The software supply chain includes all the steps, tools, and systems involved in building, testing, and deploying software. Think of it like a manufacturing process, where each part—from coding and integration to storage and deployment—needs to work smoothly and securely to get the final product into users' hands. Key parts of this chain are systems like Source Code Management (SCM) tools, Continuous Integration (CI) platforms, Continuous Deployment (CD) processes, and artifact storage repositories. Together, they help organizations develop software quickly and deliver updates and fixes to customers.

However, just as a physical supply chain can have weak links, so can the software supply chain. Each system in this chain is a potential target for attackers, who look for vulnerabilities that can give them a foothold to introduce malicious code, disrupt operations, or steal sensitive information. Recent incidents [1] [2] [3] have shown how a single security gap in one part of the chain can lead to far-reaching consequences for the affected company and its partners and customers. Securing the software supply chain is a crucial focus for teams managing development and operations.

Securing the software supply [8] chain means protecting each system involved—ensuring that only authorized people can make code changes that the build process produces

* Corresponding author:
anupammeht@gmail.com (Anupam Mehta)
Received: Nov. 3, 2024; Accepted: Nov. 21, 2024; Published: Nov. 22, 2024
Published online at http://journal.sapub.org/scit

trusted software, and that anything released or stored hasn't been tampered with. By tightening security across these areas, organizations can reduce the chances of a supply chain attack, boost trust in their processes, and deliver software more safely and reliably.

2.1. Emerging Threat Landscape in Software Supply Chain Security

The software supply chain is a complex ecosystem comprising interconnected systems that support critical processes such as code management, artifact storage, integration, and deployment. Due to their interconnected nature, these systems are increasingly vulnerable to sophisticated attacks that target multiple stages of the supply chain. Below is an overview of common threats that affect Source Code Management (SCM) systems, Continuous Integration (CI) systems, Continuous Deployment (CD) systems, and artifact storage solutions. Addressing these threats [4] is essential to maintain the integrity, security, and resilience of the software delivery pipeline.

- **Unauthorized Access:** Attackers target SCM, CI, CD, and artifact storage systems to gain unauthorized access to sensitive data and intellectual property. Weak access controls and improper credential management increase this risk, making robust authentication methods like Multi-Factor Authentication (MFA) and Role-Based Access Control (RBAC) essential across all systems.
- **Code and Artifact Tampering:** Malicious actors may inject or alter code and artifacts at various stages, resulting in compromised software being deployed. SCM systems are particularly susceptible to code tampering, while artifact storage systems are vulnerable to malware injection. Regular integrity checks, code reviews, and artifact signing can help detect and mitigate tampering attempts.
- **Credential Theft and Insider Threats:** Stolen credentials or insider access can lead to unauthorized modifications across SCM, CI, CD, and artifact storage systems. Implementing RBAC, conducting regular access reviews, and logging all access activities are crucial to reducing these risks and ensuring accountability.
- **Dependency and Supply Chain Poisoning:** Many software projects rely on third-party libraries and open-source dependencies, which can introduce vulnerabilities. Dependency confusion and typosquatting attacks can compromise these dependencies, especially in automated build and deployment processes. Maintaining a Software Bill of Materials (SBOM), performing regular dependency audits, and using trusted repositories help prevent supply chain poisoning.
- **Configuration Weaknesses:** Default or insecure configurations in CI/CD pipelines and artifact storage systems are common entry points for attackers. Regular configuration audits, adherence to security standards, and applying hardening measures to all systems are necessary to minimize misconfiguration risks.
- **Insufficient Logging and Monitoring:** Inadequate logging and monitoring can delay the detection of unauthorized access, tampering, or failed build and deployment activities across SCM, CI, CD, and artifact systems. Comprehensive logging and centralized monitoring with alerting capabilities enhance visibility and enable a quicker response to suspicious activities.
- **Data Encryption Deficiencies:** Data within SCM, artifact storage, and deployment pipelines is at risk if not encrypted both at rest and in transit. Weak encryption practices expose sensitive code and artifacts to unauthorized access, making it critical to apply strong cryptographic controls, such as TLS for data in transit and AES-256 [5] for data at rest.
- **Dependency Management Gaps:** Automated CI/CD processes may inadvertently incorporate outdated or vulnerable third-party components, increasing exposure to supply chain attacks. Regular scanning, dependency validation, and updates are essential to ensure that builds are secure and up-to-date.
- **Configuration Drift and Compliance Issues:** In CD environments, unauthorized or unsupervised changes can result in configuration drift, undermining both security and compliance. Continuous audits, secure default configurations, and compliance checks help maintain consistent, secure configurations.

2.2. Common Security Recommendations for Software Supply Chain Systems [6]

2.2.1. Architecture, Design, and Operational Process

- **Architecture and Design:** A secure supply chain system begins with a robust architectural framework. Teams managing supply chain systems must maintain detailed architecture diagrams that outline all components, such as build nodes, repositories, and artifact storage.
- **Data Flow Diagrams (DFDs)**: DFDs should also be created to visualize how data moves between different systems, allowing for easier identification of potential vulnerabilities.
- **Security Assessment:** Organizations must perform security assessments for any changes to the supply chain system or the introduction of new features. These assessments should include penetration testing, code reviews, and threat modeling to identify vulnerabilities before implementation.
- **Operational Runbook:** The supply chain system should have a comprehensive operational runbook that describes all security controls, communication protocols, and exception workflows. This document serves as a reference for security practices, outlining how teams should respond to security incidents.

2.2.2. Network and Infrastructure Security

The supply chain system must implement stringent network security measures to protect against unauthorized

access and data breaches.

- **TLS Configuration:** All endpoints communicating with the supply chain system must have Transport Layer Security (TLS) enabled, ensuring secure data transmission.
- **Endpoint Allowlisting:** Establish a list of approved endpoints that can communicate with the supply chain systems. This restricts access to only trusted sources and minimizes the risk of external threats.
- **Egress Documentation:** Document all egress endpoints to monitor and control outbound traffic from the supply chain systems. This practice prevents unauthorized data leaks and ensures that all external communications are monitored.
- **System Hardening:** Follow vendor hardening guides for the supply chain system to ensure that all components are securely configured. This includes applying best practices for securing build nodes and containerized environments.
- **Patch Management:** Organizations must regularly apply security patches to the supply chain system components, following established patch management policies to address vulnerabilities promptly.

2.2.3. Identity and Access Management (IAM)

A robust IAM strategy is critical to securing supply chain systems and ensuring that only authorized personnel have access to sensitive resources.

- **Authentication Standards:** Adhere to established authentication standards for all users and systems interacting with the supply chain systems. Multi-factor authentication (MFA) must be enforced to add a layer of security.
- **RBAC Implementation:** Implement role-based access control (RBAC) to ensure that users can only access resources necessary for their roles/team. Regularly review user roles and permissions to prevent privilege creep.
- **Inter-Tier Authentication:** Implement inter-tier authentication and authorization between different components of the supply chain systems, ensuring that communication between systems is authenticated and authorized.

2.2.4. Logging and Monitoring

Effective logging and monitoring are crucial for detecting and responding to security incidents in supply chain systems.

- **Audit Logging:** Generate and securely store logs for all access to sensitive data and administrative actions within the supply chain system. Logs should be protected from unauthorized access and tampering.
- **Centralized Logging:** Utilize a centralized logging system to aggregate logs from various components of the supply chain system. This facilitates real-time monitoring and analysis of security events. Centralized logging systems such as ELK (Elasticsearch, Logstash, Kibana) or Splunk can help aggregate logs for real-time analysis.
- **Security Activity Logging:** Implement logging for security activities performed during the supply chain system pipeline, such as static code analysis and malware scanning. This information is essential for auditing and incident response.

2.2.5. Storage Security

Data stored within supply chain systems must be managed and protected appropriately to prevent unauthorized access and data breaches.

- **Data Encryption:** All sensitive data must be encrypted both at rest and in transit. Organizations should manage encryption keys securely, ensuring that sensitive keys are not accessible to unauthorized personnel.
- **Backup and Restore Policies:** Follow established backup and restore policies to ensure that data is stored securely and can be restored in the event of a breach or data loss. Regularly test backup processes to ensure data integrity.

2.3. Recent Software Supply Chain Attacks

In this section we talk about a few software supply chain attacks [11] from the past few years that demonstrate the vulnerabilities exploited in software development and deployment pipelines.

- **SolarWinds (2020):** In one of the most significant software supply chain attacks to date, attackers infiltrated SolarWinds' build system, embedding malware known as Sunburst into the Orion software platform. This malicious code was distributed through a legitimate software update, allowing attackers to access networks and gather sensitive data. Approximately 18,000 Solar Winds customers were affected, including prominent U.S. government agencies and corporations. This breach revealed the risks associated with compromised software updates and led to increased scrutiny around supply chain security in trusted software products.
- **Kaseya VSA (2021):** Attackers exploited a vulnerability in Kaseya's Virtual System Administrator (VSA), a tool used for IT management, to deliver ransomware to Kaseya's customers. The REvil ransomware group leveraged the VSA platform to install ransomware across networks, affecting about 1,500 organizations globally. This incident underscored the dangers of vulnerabilities in remote management tools and illustrated the widespread impact a compromised IT management solution can have on an organization's clients and network security.
- **Dependency Confusion Attacks (2021):** A researcher demonstrated how "dependency confusion" could be exploited by registering internal package names on public repositories such as npm, PyPI, and RubyGems. In several cases, companies' build systems mistakenly

downloaded these public packages, some containing malicious code, instead of the intended internal versions. Companies like Microsoft, Apple, and Tesla were among those affected. This attack brought attention to the security risks in third-party dependencies and internal package management, prompting companies to adopt stricter policies around dependency management and package sourcing.

- **Log4Shell Vulnerability in Log4j (2021):** The Log4Shell vulnerability, discovered in the Log4j logging library, exposed a critical flaw that allowed attackers to execute arbitrary code by logging a crafted payload. Given Log4j's widespread use in web applications, servers, and enterprise software, the vulnerability sparked a global response to patch and protect affected systems. Organizations such as Amazon, Microsoft, and Apple were impacted. This event highlighted the importance of securing open-source dependencies and maintaining rigorous vulnerability management processes for widely-used software components.
- **3CX VoIP Supply Chain Attack (2023):** In 2023, attackers targeted the build process of the 3CX desktop application, a widely-used VoIP and communication tool, injecting malware into the application. The compromised software was then distributed to 3CX customers, which included multiple organizations. This attack underscored the risks of insufficient build process security, emphasizing the need for integrity controls around software builds, even in trusted applications used for secure communication.

3. Source Code Management Systems

3.1. Overview

Source Code Management (SCM) systems are central to modern software development, allowing organizations to efficiently manage, store, and deploy their code. Due to the critical nature of the data they hold, SCM systems are high-value targets for cyberattacks. Breaches can lead to intellectual property theft, loss of competitive edge, and compromised software integrity. This section outlines essential security practices for SCM [10] systems, with recommendations aligned with industry standards such as ISO 27001 [8] and the NIST framework. A layered security approach—incorporating network security, identity management, data classification, and continuous monitoring—ensures comprehensive protection and compliance with regulatory requirements.

3.2. Security Recommendations and Considerations for SCM Systems

This section focuses on tailored security practices and controls that address the unique vulnerabilities and operational needs of SCM systems, supplementing the broader recommendations provided in Section 2.2.

Access Control and Authentication

- Implement Multi-Factor Authentication (MFA) for all users to prevent unauthorized access to source code systems.
- Automate account management to flag and remove accounts that no longer require access. Enforce SSH key rotation and regular password updates to secure access further.

Webhook Security

- Secure webhooks with authentication tokens and IP allowlists to ensure only authorized systems can trigger events and receive data from SCM.

Branching and Repository Controls

- Enforce branch protection rules, such as code review requirements or prohibiting direct commits to main branches, to safeguard critical branches.
- Set repositories to private by default, enforce strict access control policies, and monitor for unauthorized changes.

Pull Request Reviews

- Require multiple reviewers for pull requests, enforce approval workflows, and enable code signing to ensure that only validated code is merged.

Pre-Commit Hooks and Standards

- Use pre-commit hooks to enforce coding standards, prevent sensitive data exposure, and perform security scans before commits are finalized.

Role-Based Access Control (RBAC)

- Restrict repository access based on user roles, granting permissions only as necessary for each developer's responsibilities.

Commit Signing

- Require developers to sign commits with GPG keys, ensuring that only verified and trusted code authors contribute to the codebase.

Automated Sensitive Data Scanning

- Integrate tools that automatically scan repositories for sensitive data (e.g., API keys, credentials) to prevent accidental exposure.

Branch Policies and Lifecycle Management

- Enforce naming conventions and lifecycle policies on branches to organize code management and prevent unauthorized changes in protected branches.

Automated Vulnerability Scanning

- Schedule regular scans on repositories to identify vulnerabilities, outdated dependencies, and code weaknesses, ideally integrating these scans into the CI pipeline.

Audit Logging and Monitoring

- Enable and review audit logs to track actions like merges, branch deletions, or permission changes, and to identify

unusual or unauthorized activity.

Pull Request Templates

- Use pull request templates to guide developers in following security guidelines, including describing code changes and verifying tests.

Inactive Account Management

- Regularly review and disable inactive accounts with SCM access to reduce the risk of unauthorized access through unused accounts.

By implementing the minimum security expectations outlined above, organizations can prevent unauthorized access, ensure compliance with industry standards, and protect their source code from tampering, thereby safeguarding their intellectual property.

4. Artifact Storage Systems

4.1. Overview

Artifact storage systems are essential for managing the storage and distribution of software artifacts, such as container images, libraries, and build packages, throughout the development and deployment lifecycle. Due to the critical nature of these artifacts, protecting artifact storage is crucial to ensure that only secure, verified, and untampered artifacts reach production environments. This section provides specific security recommendations aligned with best practices to address the unique risks associated with artifact storage.

4.2. Security Recommendations and Considerations for Artifact Storage Systems

This section focuses on tailored security practices and controls that address the unique vulnerabilities and operational needs of Artifact Storage systems, supplementing the broader recommendations provided in Section 2.2.

Integrity Checks and Artifact Signing

- Use hashing and digital signing to verify artifact integrity before storage and deployment. Only publish signed artifacts to the artifact storage service to verify the integrity of artifacts before deployment.
- Implement integrity validation mechanisms to automatically flag artifacts with mismatched signatures or hashes.

Dependency and Version Management

- Maintain a clear versioning strategy for artifacts, allowing easy rollbacks in case a compromised artifact is detected. Implement a policy to archive or remove outdated or vulnerable artifacts from storage.
- Source third-party dependencies from approved repositories, and validate dependency hashes and security assessments before use.

Automated Vulnerability Scanning

- Schedule regular vulnerability scans on artifacts stored in the artifact storage system, notifying artifact owners of scan results and creating tasks for remediation.
- Ensure automated scans detect vulnerabilities before storage and deployment to prevent the distribution of vulnerable artifacts.

Webhook and API Security

- Secure webhooks and APIs with authentication tokens, IP allowlisting, and rate limiting to ensure only authorized systems can interact with artifact storage.
- Regularly review and rotate service account passwords, API keys, and disable unused keys to reduce exposure to unauthorized access.

Configuration Management and Hardening

- Follow vendor-provided hardening guidelines (e.g., for Artifactory, Nexus Repository Manager) to secure configurations, disable unnecessary services, and reduce the attack surface.
- Document all configuration changes and perform regular audits to ensure security best practices are upheld.
- Only privileged users must be able to make configuration changes to the artifact storage systems and supporting components.

Documentation and Ownership

- Document all data types stored in the artifact storage system, including metadata about artifact types (e.g., container images, Helm charts) and ownership information to ensure accountability.
- Enforce ownership policies for all artifacts, ensuring clear tracking and accountability within the artifact storage system.

Retention and Cleanup Policies

- Establish and enforce retention policies to automatically archive or delete obsolete artifacts, reducing clutter and minimizing the storage of potentially vulnerable artifacts.
- Prune artifacts identified as vulnerable within seven days of last use and implement rollback strategies for deleted third-party dependencies.

Exception Tracking and Break Glass Scenarios

- Track any exceptions granted to teams for artifact management, ensuring they are documented and limited to designated teams or members.
- Document break glass scenarios allowing direct uploads to the artifact storage system, and require multi-factor authentication to approve and track these instances.

Incident Response and Backup Protocols

- Ensure backup systems are configured securely to protect artifact data, with encryption for all stored backups, and regularly test restore procedures to ensure data integrity.

By implementing these security recommendations, organizations can protect the integrity and confidentiality of artifacts, reducing the risk of compromised artifacts reaching production and enhancing the overall security of the software supply chain.

5. Continuous Integration (CI) Systems

5.1. Overview

Continuous Integration (CI) systems are vital for automating the processes of building, testing, and preparing code for deployment. They streamline development workflows and allow for rapid integration of new code. However, CI systems handle sensitive code and configuration data, making them prime targets for attackers seeking to inject malicious code or exploit insecure configurations. This section outlines key security practices for CI [10] systems, focusing on tailored recommendations to protect the CI pipeline's integrity.

5.2. Security Recommendations and Considerations for CI Systems

This section focuses on tailored security practices and controls that address the unique vulnerabilities and operational needs of Continuous Integration systems, supplementing the broader recommendations provided in Section 2.2.

Build Environment Security and Isolation

- Secure CI build environments by isolating them from production and other critical environments, ensuring that compromised builds do not affect other systems.
- Use containerization or virtualization to sandbox builds, reducing the risk of lateral movement if a build environment is compromised.

Access Control and Authentication

- Apply Role-Based Access Control (RBAC) to limit permissions to essential personnel, and enforce Multi-Factor Authentication (MFA) for access to CI systems.
- Automate user access reviews and deactivate accounts that no longer need access, ensuring that only necessary users can modify CI configurations and build pipelines.

Source Code and Artifact Integrity

- Implement code signing and artifact signing to ensure that only verified and trusted code artifacts pass through the CI pipeline.
- Use hashing and other integrity checks to detect unauthorized modifications in build artifacts before and after CI processes.

Dependency and Version Management

- Regularly scan dependencies within CI builds for vulnerabilities, and configure dependency validation to ensure only approved libraries are used.
- Maintain a clear version control strategy, allowing easy rollbacks if a vulnerability or issue is discovered in a dependency.

Automated Security and Vulnerability Scanning

- Integrate security scanning tools (e.g., static code analysis, dependency checks) into the CI pipeline to automatically detect vulnerabilities before code progresses to deployment stages.
- Conduct vulnerability assessments on build artifacts to ensure compliance with organizational security policies, and alert artifact owners if issues are detected.

Logging, Monitoring, and Audit Trails

- Enable comprehensive logging for all build and access activities, including build failures, modifications, and deletions. Store logs securely for audit purposes.
- Use centralized monitoring solutions to aggregate and review CI logs, allowing for real-time alerts on anomalies or unauthorized actions within the CI environment.

Configuration Management and Hardening

- Harden CI tools by following vendor-provided security guidelines, and disable unused plugins, integrations, and default configurations that could introduce vulnerabilities.
- Regularly audit CI configurations to ensure that hardening practices remain intact and that security configurations align with organizational policies.

Secrets Management

- Store all credentials, tokens, and API keys in an approved secrets management solution (e.g., HashiCorp Vault, AWS Secrets Manager) rather than embedding them directly in CI scripts or code.
- Rotate secrets regularly and restrict access to secrets only to the CI processes that require them, reducing exposure in case of a compromise.

Webhook and API Security

- Secure webhooks with authentication tokens and IP allowlisting to ensure that only authorized external systems can trigger CI processes.
- Periodically review and rotate API keys and disable any unused keys to reduce exposure to unauthorized CI triggers or integrations.

Pipeline Access Reviews and Expired Resource Cleanup

- Regularly review pipeline access rights and permissions to confirm that only authorized users and services can interact with the CI pipeline.
- Implement cleanup processes for expired or obsolete build resources to prevent accumulation of untracked, potentially vulnerable artifacts or dependencies within the CI system.

Incident Response and Backup Protocols

- Develop an incident response plan specific to CI environments, detailing steps for identifying, isolating, and recovering from security incidents within the CI pipeline.
- Ensure regular backups of CI configurations and scripts, securely stored and encrypted, with periodic tests of restore capabilities to validate backup integrity.

By following these security recommendations, organizations can significantly reduce the risks associated with CI systems. Implementing these tailored security measures ensures that CI pipelines maintain their integrity, protect sensitive data, and deliver secure and reliable code to subsequent stages in

the software delivery lifecycle.

6. Continuous Deployment (CD) Systems

6.1. Overview

Continuous Deployment (CD) systems automate the release of software applications by deploying artifacts generated from Continuous Integration (CI) processes into production environments. CD systems enable rapid and reliable software delivery, but they also present unique security challenges. If compromised, a CD system could result in unauthorized deployments, configuration drift, and the propagation of malicious artifacts. This section outlines security practices tailored to CD [10] systems to ensure the integrity, security, and stability of deployments.

6.2. Security Recommendations and Considerations for CD Systems

Environment Isolation and Segmentation

- Isolate CD environments from production environments using network segmentation to prevent lateral movement in the event of a compromise.
- Separate staging, testing, and production environments, and control access to production to reduce the risk of unauthorized modifications during deployment.

Pipeline Integrity and Deployment Verification

- Validate the integrity of deployment artifacts using digital signatures or hashing to ensure that only verified artifacts are deployed into production.
- Implement automated verification steps in the CD pipeline, such as checksum validation and artifact signing, to confirm artifact integrity before deployment.

Configuration Management and Drift Prevention

- Regularly review and audit deployment configurations to prevent unauthorized changes and configuration drift, which could introduce vulnerabilities or misalignments in production.
- Use configuration management tools (e.g., Ansible, Puppet, Spinnaker) to automate and enforce consistent configurations across environments, minimizing the risk of manual configuration errors.

Automated Vulnerability Scanning and Compliance Checks

- Integrate automated security scans in the CD pipeline to check for vulnerabilities in deployment artifacts before they reach production. Ensure that compliance checks are also enforced to meet industry and organizational standards.
- Schedule periodic vulnerability assessments on deployed artifacts and configurations in production environments to catch any overlooked issues.

Logging, Monitoring, and Audit Trails

- Enable logging for all deployment activities, including artifact deployment, configuration changes, and user access to the CD system. Store logs securely and monitor them for anomalies.
- Use centralized monitoring solutions to aggregate and analyze CD logs in real time, allowing for prompt response to unauthorized deployment attempts or suspicious activity.

Secrets Management

- Store all credentials, API keys, and tokens required by CD processes in an approved secrets management solution (e.g., HashiCorp Vault, AWS Secrets Manager), and avoid embedding secrets directly in deployment scripts.
- Rotate secrets regularly and restrict access to deployment credentials to only those processes or individuals that require them.

Rollback and Recovery Procedures

- Establish automated rollback mechanisms to quickly revert deployments if issues are detected post-deployment. Define rollback strategies for specific failure scenarios, such as artifact corruption or deployment errors.
- Regularly test rollback and recovery procedures to ensure that they work effectively and that deployment teams are trained in their use.

Deployment Approval and Release Gates

- Require approvals from designated stakeholders before deploying to production, ensuring that critical deployments are reviewed and vetted.
- Set up release gates for high-risk deployments, including automated testing, compliance verification, and manual checks where necessary to increase deployment confidence.

Deployment Freeze Windows

- Define deployment freeze windows during critical times (e.g., holiday seasons or high-traffic periods) to prevent unintended disruptions in production.
- Implement controls to block non-essential deployments during freeze periods unless explicitly authorized through a documented process.

Webhook and API Security

- Secure webhooks and APIs that trigger deployments with authentication tokens and IP allowlists to prevent unauthorized deployment actions.
- Periodically rotate API tokens and disable unused webhooks to reduce exposure to unauthorized triggers or attacks.

By implementing these security recommendations, organizations can better safeguard their CD systems from unauthorized access, configuration drift, and deployment of compromised artifacts. Following these practices ensures

that CD pipelines maintain their integrity, and that deployments remain secure and reliable in production environments.

7. Additional Systems and Process

While core systems such as Source Code Management (SCM), Continuous Integration (CI), Continuous Deployment (CD), and artifact storage are crucial, there are additional systems and processes within the software supply chain that play equally important roles in maintaining security, resilience, and compliance. These additional systems [10] involve dependency management, third-party risk management, infrastructure-as-code (IaC) management, testing frameworks, and incident response processes, all of which help reduce security vulnerabilities and streamline software delivery.

7.1. Dependency and Third-Party Library Management

Software development increasingly relies on third-party libraries, open-source components, and external dependencies. These third-party components introduce potential vulnerabilities into the supply chain, as attackers can exploit dependencies to introduce malicious code or leverage outdated components with known vulnerabilities.

Security Recommendations:

- Implement a Software Bill of Materials (SBOM) to document all dependencies used within the software, facilitating vulnerability tracking and compliance.
- Enforce regular dependency audits and vulnerability scanning for all third-party libraries to ensure components remain secure and up-to-date.
- Use trusted repositories for dependencies and limit the use of unverified or less popular third-party libraries.
- Adopt automated dependency management tools to detect and notify teams about outdated or vulnerable dependencies.

7.2. Infrastructure-as-Code (IaC) and Configuration Management

Infrastructure-as-Code (IaC) is a key process in modern DevOps practices, enabling organizations to define and manage infrastructure configurations through code. However, misconfigurations in IaC can introduce vulnerabilities across deployment environments and lead to security risks in production.

Security Recommendations:

- Apply static analysis and security checks on IaC scripts to identify misconfigurations before deployment.
- Use role-based access control (RBAC) and policy-as-code tools (e.g., Open Policy Agent) to enforce security policies and control who can make changes to infrastructure.
- Regularly review and update IaC templates to reflect changes in security standards and infrastructure requirements.
- Monitor IaC deployments and configurations to detect and resolve configuration drift in real time.

7.3. Secrets Management

Secrets management is essential for protecting sensitive credentials, tokens, and API keys used throughout the software supply chain. Poor handling of secrets can lead to unauthorized access to critical systems and data.

Security Recommendations:

- Store secrets in secure secrets management solutions (e.g., HashiCorp Vault, AWS Secrets Manager) rather than embedding them in code or configuration files.
- Enforce multi-factor authentication (MFA) for access to secrets management tools.
- Regularly rotate and audit secrets, ensuring that they remain current and securely managed.
- Implement logging and monitoring for all access and modifications to secrets to detect suspicious activities.

7.4. Testing and Quality Assurance (QA) Processes

Testing and QA processes are critical to identifying security flaws and ensuring software reliability before deployment. This includes both functional testing and security testing, which are essential to maintaining a secure software supply chain.

Security Recommendations:

- Integrate security testing tools (e.g., static and dynamic analysis tools) into CI/CD pipelines to identify vulnerabilities early in the development lifecycle.
- Use containerized or isolated environments for testing to prevent accidental exposure of sensitive data or configurations.
- Conduct regular penetration testing on staging and production environments to identify potential vulnerabilities and misconfigurations.
- Implement automated regression testing to ensure that updates do not introduce new vulnerabilities or functional errors.

7.5. Monitoring and Incident Response

Effective monitoring and incident response are essential components of a resilient software supply chain, allowing teams to detect and respond to threats before they can escalate.

Security Recommendations:

- Establish real-time monitoring across all supply chain systems (SCM, CI/CD, artifact storage, etc.) to detect anomalies or suspicious activities.
- Create an incident response plan tailored to the software supply chain, with clearly defined roles, responsibilities, and procedures.
- Regularly conduct incident response drills and tabletop exercises to ensure teams are prepared to respond quickly and effectively to security events.
- Implement log aggregation and analysis solutions (e.g., Splunk, ELK Stack) to centralize logs from all systems

for efficient incident investigation and response.

7.6. Compliance and Regulatory Adherence

As software supply chains increasingly come under scrutiny, organizations must ensure that their processes and systems adhere to a growing body of regulatory standards and industry best practices. Adherence to standards such as GDPR, HIPAA, SOC 2, and ISO/IEC 27001 is crucial, especially in highly regulated sectors like healthcare, finance, and government.

Non-compliance not only risks regulatory penalties but can also result in reputational damage and legal liabilities if security lapses compromise sensitive information..

Following are few relevant standards and regulations for Software supply chain security:

- The Center for Internet Security (CIS) Benchmark for Supply Chain Security
- Supply Chain Levels for Software Artifacts (SLSA) standard
- The National Institute of Standards and Technology (NIST) published
  - Secure Software Development Framework (SSDF) 1.1
  - NIST SP 800-161 - Cybersecurity Supply Chain Risk Management Practices for Systems and Organizations
- Internet Engineering Task Force (IETF) compiled The Supply Chain Integrity, Transparency, and Trust (SCITT) standard
- Executive Order 14028
- FDA-2021-D-1158-Cybersecurity in Medical Devices
- PCI DSS v4 - Requirement 6: Develop and Maintain Secure Systems and Software
- European Union's Cyber Resilience Act
- European Union's Digital Operational Resilience for the Financial Sector (DORA)

Ensuring that all components of the software supply chain meet these requirements helps prevent regulatory penalties and fosters trust with customers.

Security Recommendations:

- Each organization must comply with its respective regulations and standards based on the industry sector they fall in. For e.g. Healthcare sector organizations must adhere to HIPAA regulations, Finance industry handling payment card information (PCI) must comply with PCI DSS.
- Conduct regular audits of supply chain processes to ensure compliance with applicable industry standards and regulatory requirements.
- Implement data governance policies to ensure data handling within SCM, CI/CD, and artifact storage systems meet legal and regulatory standards.
- Use policy-as-code frameworks (e.g., Open Policy Agent) to automate compliance checks within CI/CD pipelines. Policy-as-code allows organizations to enforce data protection standards and access controls in real-time, ensuring that each deployment adheres to regulatory requirements without manual intervention.
- Maintain a compliance dashboard to track and report on the compliance status of different components within the software supply chain.
- Stay updated with changes in regulations and incorporate them into security practices and supply chain policies.

Non-compliance with these regulations can lead to severe legal and financial repercussions. For example, GDPR violations can result in fines up to €20 million or 4% of global annual revenue, whichever is higher. Additionally, security lapses that expose sensitive data may lead to lawsuits, loss of customer trust, and damage to an organization's reputation. Proactively implementing regulatory-aligned security practices helps avoid these risks, strengthens trust with clients and partners, and enhances the organization's overall security posture.

By integrating compliance-driven security practices into their software supply chains, organizations can better manage regulatory risks, reduce legal liabilities, and build customer confidence in their data protection and privacy measures.

8. Conclusions

The security of the software supply chain [9] is essential for ensuring that digital products remain trustworthy, compliant, and resilient against evolving threats. This paper highlights the primary vulnerabilities across core systems—SCM, CI, CD, and artifact storage—and provides practical, standards-based recommendations to mitigate these risks. By implementing the outlined security controls, organizations can significantly reduce their exposure to supply chain attacks, safeguard intellectual property, and strengthen their overall software delivery process. Continuous vigilance and adherence to best practices are key to maintaining a secure and resilient supply chain.

REFERENCES

[1] Source Code Leakage - https://www.usenimbus.com/post/source-code-leak-what-it-is-and-5-high-profile-examples.

[2] Codecov breach - exposure of information stored in CI environments https://www.zdnet.com/article/codecov-breach-impacted-hundreds-of-customer-networks/.

[3] Heroku breach from 2022 - Improper integrations with SCM https://blog.heroku.com/april-2022-incident-review.

[4] National Institute of Standards and Technology (NIST). (2015). *Supply Chain Risk Management Practices for Federal Information Systems and Organizations* (Special Publication 800-161). NIST. Retrieved from https://csrc.nist.gov/publications/detail/sp/800-161/rev-1/final.

[5] National Institute of Standards and Technology (NIST)- AES https://www.nist.gov/publications/advanced-encryption-standard-aes.

[6] OWASP Foundation. (2020). *OWASP Software Assurance Maturity Model (SAMM), Version 2.0*. OWASP. Retrieved from https://owaspsamm.org/.

[7] International Organization for Standardization. (2013). *ISO/IEC 27001: Information Security Management*. ISO. Retrieved from https://www.iso.org/standard/54534.html.

[8] Cybersecurity & Infrastructure Security Agency (CISA). (2022). *Securing the Software Supply Chain for Developers*. CISA. Retrieved from https://cisa.gov/publication/securing-software-supply-chain-developers.

[9] Sonatype. (2022). *2022 State of the Software Supply Chain*. Sonatype. Retrieved from https://www.sonatype.com/resources/white-paper/state-of-the-software-supply-chain-2022.

[10] Microsoft Security Blog. (2020, March 12). *Security in DevOps: A Security Practitioner's Guide*. Microsoft. Retrieved from https://www.microsoft.com/security/blog/2020/03/12/security-in-devops/.

[11] Software Supply Chain Attacks

- SolarWinds (2020) FireEye. (2020, December 13). *Highly Evasive Attacker Leverages SolarWinds Supply Chain to Compromise Multiple Global Victims with SUNBURST Backdoor*. FireEye Threat Research. Retrieved from https://www.fireeye.com/blog/threat-research/2020/12/evasive-attacker-leverages-solarwinds-supply-chain-compromises-with-sunburst-backdoor.html.
- Kaseya VSA (2021) CISA. (2021, July 4). *Compromise of Kaseya VSA Platform and Associated Ransomware Attack*. Cybersecurity & Infrastructure Security Agency (CISA). Retrieved from https://us-cert.cisa.gov/ncas/current-activity/2021/07/04/compromise-kaseya-vsa-platform-and-associated-ransomware-attack.
- Dependency Confusion Attacks (2021) Birsan, A. (2021, February 10). *Dependency Confusion: How I Hacked Into Apple, Microsoft and Dozens of Other Companies*. Medium. Retrieved from https://medium.com/@alex.birsan/dependency-confusion-4a5d60fec610.
- Log4Shell Vulnerability in Log4j (2021) Apache Software Foundation. (2021, December 10). *Log4j Security Vulnerability CVE-2021-44228*. Apache Logging Services. Retrieved from https://logging.apache.org/log4j/2.x/security.html.
- 3CX VoIP Supply Chain Attack (2023) CrowdStrike. (2023, March 29). *CrowdStrike Investigates Supply Chain Attack Involving 3CX Desktop App*. CrowdStrike Blog. Retrieved from https://www.crowdstrike.com/blog/crowdstrike-investigates-supply-chain-attack-involving-3cx-desktop-app/.