**NIST Special Publication
NIST SP 800-204D ipd**

Strategies for the Integration of Software Supply Chain Security in DevSecOps CI/CD Pipelines

Initial Public Draft

Ramaswamy Chandramouli
Frederick Kautz
Santiago Torres Arias

NIST | NATIONAL INSTITUTE OF
STANDARDS AND TECHNOLOGY
U.S. DEPARTMENT OF COMMERCE

NIST Special Publication
NIST SP 800-204D ipd

Strategies for the Integration of Software Supply Chain Security in DevSecOps CI/CD Pipelines

Initial Public Draft

Ramaswamy Chandramouli
Computer Security Division
Information Technology Laboratory

Frederick Kautz
TestifySec

Santiago Torres Arias
Electrical and Computer Engineering Department
Purdue University

Authority

NIST Technical Series Policies
Copyright, Use, and Licensing Statements
NIST Technical Series Publication Identifier Syntax

Publication History
Approved by the NIST Editorial Review Board on YYYY-MM-DD [Will be added on final publication]

Author ORCID iDs
Ramaswamy Chandramouli: 0000-0002-7387-5858

37 **Public Comment Period**
38 August 30, 2023 – October 13, 2023

39 **Submit Comments**
40 sp800-204d-comments@nist.gov
41
42 National Institute of Standards and Technology
43 Attn: Computer Security Division, Information Technology Laboratory
44 100 Bureau Drive (Mail Stop 8930) Gaithersburg, MD 20899-8930

45 **All comments are subject to release under the Freedom of Information Act (FOIA).**

46 **Abstract**

47 The predominant application architecture for cloud-native applications consists of multiple
48 microservices with a centralized application infrastructure, such as a service mesh, that provides
49 all application services. This class of applications is generally developed using a flexible and
50 agile software development paradigm called DevSecOps. A salient feature of this paradigm is the
51 use of flow processes called CI/CD pipelines, which initially take the software through various
52 stages (e.g., build, test, package, and deploy) in the form of source code through operations that
53 constitute the software supply chain (SSC). This document outlines strategies for integrating
54 SSC security measures into CI/CD pipelines.

55 **Keywords**

56 actor; artifact; attestation; CI/CD pipeline; package; provenance; repository; SBOM; SDLC;
57 SLSA; software supply chain.

58 **Reports on Computer Systems Technology**

59 The Information Technology Laboratory (ITL) at the National Institute of Standards and
60 Technology (NIST) promotes the U.S. economy and public welfare by providing technical
61 leadership for the Nation's measurement and standards infrastructure. ITL develops tests, test
62 methods, reference data, proof of concept implementations, and technical analyses to advance
63 the development and productive use of information technology. ITL's responsibilities include the
64 development of management, administrative, technical, and physical standards and guidelines for
65 the cost-effective security and privacy of other than national security-related information in
66 federal information systems. The Special Publication 800-series reports on ITL's research,
67 guidelines, and outreach efforts in information system security, and its collaborative activities
68 with industry, government, and academic organizations.

Call for Patent Claims

This public review includes a call for information on essential patent claims (claims whose use would be required for compliance with the guidance or requirements in this Information Technology Laboratory (ITL) draft publication). Such guidance and/or requirements may be directly stated in this ITL Publication or by reference to another publication. This call also includes disclosure, where known, of the existence of pending U.S. or foreign patent applications relating to this ITL draft publication and of any relevant unexpired U.S. or foreign patents.

ITL may require from the patent holder, or a party authorized to make assurances on its behalf, in written or electronic form, either:

a) assurance in the form of a general disclaimer to the effect that such party does not hold and does not currently intend holding any essential patent claim(s); or

b) assurance that a license to such essential patent claim(s) will be made available to applicants desiring to utilize the license for the purpose of complying with the guidance or requirements in this ITL draft publication either:

   i. under reasonable terms and conditions that are demonstrably free of any unfair discrimination; or

   ii. without compensation and under reasonable terms and conditions that are demonstrably free of any unfair discrimination.

Such assurance shall indicate that the patent holder (or third party authorized to make assurances on its behalf) will include in any documents transferring ownership of patents subject to the assurance, provisions sufficient to ensure that the commitments in the assurance are binding on the transferee, and that the transferee will similarly include appropriate provisions in the event of future transfers with the goal of binding each successor-in-interest.

The assurance shall also indicate that it is intended to be binding on successors-in-interest regardless of whether such provisions are included in the relevant transfer documents.

Such statements should be addressed to: sp800-204d-comments@nist.gov

95   **Table of Contents**

138
139

140 **Acknowledgments**

Executive Summary

Cloud-native applications are made up of multiple loosely couple components called microservices. This class of applications is generally developed through an agile software development life cycle (SDLC) paradigm called DevSecOps, which uses flow processes called Continuous Integration/Continuous Delivery (CI/CD) pipelines.

Analyses of recent software attacks and vulnerabilities have led both government and private-sector organizations involved in software development, deployment, and integration to focus on the activities involved in the entire SDLC. These collected activities are called the software supply chain (SSC).

The integrity of these individual operations contributes to the overall security of an SSC, and threats can arise from attack vectors unleashed by malicious actors as well as defects introduced when due diligence practices are not followed during SDLC.

Executive Order (EO) 14028, NIST's Secure Software Development Framework (SSDF)[2], other government initiatives, and industry forums have discussed the security of SSC to enhance the security of all deployed software. This document focuses on actionable measures to integrate the various building blocks of SSC security assurance into CI/CD pipelines to prepare organizations to address SSC security in the development and deployment of their cloud-native applications.

Building a robust SSC security edifice requires various artifacts, such as a software bill of materials (SBOM) and frameworks for the attestation of software components. Since the specification of these artifacts, their mandatory constituents, and the requirements that processes using them must satisfy are continually evolving through projects in government organizations and various industry forums, they are beyond the scope of this document.

1. Introduction

Cloud-native applications consist of multiple loosely coupled services or microservices and are deployed and run using an integrated application service infrastructure called a service mesh. The applications are developed through an agile software development life cycle (SDLC) paradigm called DevSecOps, which uses flow processes called Continuous Integration/ Continuous Delivery (CI/CD) pipelines. The service mesh provides numerous runtime security measures through mechanisms for assigning unique service identities for microservices and policy enforcement through proxies. However, sophisticated attacks on software have been carried out through the stealthy introduction of attack vectors during various activities in the SDLC, which collectively constitute the software supply chain (SSC). Thus, in the context of cloud-native applications, SSC security assurance measures must be integrated into CI/CD pipelines.

1.1. Purpose

This document outlines strategies for integrating SSC security assurance measures into CI/CD pipelines. The overall goal is to ensure that the CI/CD pipeline activities that take source code through the build, test, package, and deployment stages are not compromised.

1.2. Scope

SSC security assurance measures use various artifacts, such as a software bill of materials (SBOM) and frameworks for the attestation of software components. The specification of these artifacts, their mandatory constituents, and the requirements that processes using them must satisfy are continually evolving through projects in government organizations and various industry forums and are therefore beyond the scope of this document. Rather, this document focuses on actionable measures to integrate various building blocks for SSC security assurance into CI/CD pipelines to enhance the preparedness of organizations to address SSC security in the development and deployment of their cloud-native applications.

1.3. Target Audience

This document is intended for a broad group of practitioners in the software industry, including site reliability engineers, software engineers, project and product managers, and security architects and engineers.

1.4. Relationship to Other NIST Documents

This document is part of the NIST Special Publication (SP) 800-204 series of publications, which offer guidance on providing security assurance for cloud-native applications that are developed and deployed using the DevSecOps SDLC paradigm with CI/CD pipelines. SP 800-204C [1] discussed DevSecOps, which is an agile software development paradigm for cloud-native applications that focuses on the various types of code involved in microservices-based applications that are supported by a service mesh infrastructure. SP 800-218 [2] provided a

203 comprehensive list of high-level practices and tasks for providing SSC security under the Secure
204 Software Development Framework (SSDF) based on the directives Executive Order (EO) 14028
205 [3]. Other documents in the SP 800-204 series have outlined the mechanisms for enforcing
206 various types of access controls for inter-service calls in the microservices environment during
207 runtime.

208 This document presents strategies for integrating SSC security into CI/CD pipelines through the
209 identification of workflow tasks that can meet the goals of the various high-level practices
210 outlined in the SSDF. Since the SSDF is application architecture and the SDLC paradigm is
211 agnostic, not all practices and tasks outlined in the SSDF may be applicable in the context of
212 cloud-native applications developed using the DevSecOPs SDLC paradigm. Hence, this
213 document maps the SSC security integration strategies for CI/CD pipelines to the high-level
214 practices in the SSDF.

1.5.   Document Structure

216 This document is organized as follows:

217 • Section 2 presents a series of definitions for modelling and understanding software
218   supply chains and their compromises.

219 • Section 3 provides a broad understanding of common risk factors and potential mitigation
220   measures with a particular focus on the software developer environment.

221 • Section 4 provides the background for CI/CD pipelines, the broad security goals of the
222   processes involved, and the entities that need to be trusted.

223 • Section 5 outlines strategies for integrating SSC security assurance measures into CI/CD
224   pipelines.

225 • Section 6 provides a summary and conclusions.

226 • Appendix A provides a mapping of the SSC security integration strategies for CI/CD
227   pipelines to the SSDF's high-level practices.

228 • Appendix B provides a justification for the omission of certain measures related to SSDF
229   practices in this document.

2. Software Supply Chain (SSC) — Definition and Model

2.1. Definition

Most activities in the SSC strongly affect the resulting software product. As such, the security of each individual activity is paramount for the security of the end result. This includes not only the integrity of the activities themselves but also the assurance that all activities were carried out and — conversely — that no unauthorized activities were injected into the chain.

While software composition (e.g., dependency management) is under the purview of software supply chain activities, other often overlooked activities are central to the software supply chain. This includes writing source code; building, packaging, and delivering an application; and repackaging and containerization.

In order to carry out an SSC attack, an attacker needs to subvert, remove, or introduce a step within the SSC to maliciously modify the resulting software product. In practice, attackers often target the activities mentioned above to implant backdoors and subsequently compromise a target or exfiltrate sensitive information once the application is delivered.

SSC security should also account for discovering and tracking software defects rather than simply mitigating attackers. This can be achieved by sharing a software bill of materials (SBOM) with end users who can build inventories of software components to identify and address any vulnerabilities or defects in the software.

2.2. Economics of Security

SSC attacks have two fundamental properties that make them appealing to attackers. First, they allow attackers to infiltrate highly regulated environments through less secure but legitimate channels. Second, due to the highly interconnected nature of supply chains, they allow for widespread damage in a short period of time.

Attacks that target highly regulated environments often allow motivated attackers to identify weak spots in the chain. In the case of SOLORIGATE [4], for example, attackers identified a single point of compromise that delivered software to multiple government agencies. Such attacks are also stealthy because they typically propagate through legitimate channels, such as software updates, which allows for widespread damage to users of the target software. Since attackers typically seek this avenue to obtain short-term benefits, widespread attacks of this nature often rely on the use of private crypto miners and crypto jackers. This is evidenced in the prevalence of these vectors existing in breadth-first approaches, such as typo and combosquatting attacks. Regardless of the motivations of the attackers, both vectors highlight the possibility of incredible impact when carried out successfully.

2.3. Governance Model

Due to the distributed nature of an SSC, multiple practices, developer cultures, security and quality expectations, and legislative frameworks exist. As a consequence, there is no unified governance model, and these distinct models often overlap.

267 **2.4. SSC Model**

268 At a high-level, an SSC is a collection of steps that create, transform, and assess the quality of
269 software artifacts. These steps are often carried out by different actors who use and consume
270 artifacts to produce new artifacts. For example, a build step uses a series of artifacts as tools
271 (e.g., a compiler and a linker) and consumes artifacts (i.e., source code) to produce a new artifact
272 (i.e., the compiled binary).

273 Without a loss of generality, this same definition can be applied to other actions, such as writing
274 code, packaging an application inside of a container, and performing quality assurance. This
275 definition also encompasses more activities than are colloquially considered. That is, it includes
276 elements of secure software development, secure build systems, and dependency management.

277 While this simplified model can accommodate multiple activities, mitigations and attacks may
278 surface in different, nuanced ways for each activity.

279
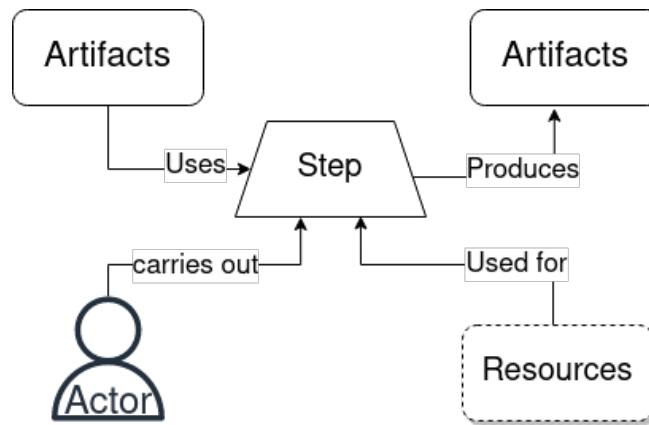
280

281 **Fig. 1.** Interaction between the different elements of a software supply chain step

282 **2.4.1. Software Supply Chain Defects**

283 Much like software defects (i.e., bugs), defective artifacts can propagate throughout an SSC and
284 affect its security posture. A noteworthy example of such a defect is that of Log4Shell [5], where
285 a vulnerability in a highly used software artifact allowed attackers to compromise a large number
286 of targets with very little effort. While the line between a defect and an attack is often blurred in
287 the SSC context, the guiding principle is that of intent — that is, whether or not the upstream
288 actor intended for that defect to be exploited.

289 **2.4.2. Software Supply Chain Attacks**

290 In contrast to defects, an SSC attack is when a malicious party tampers with steps, artifacts, or
291 actors within the chain to compromise the consumers of a software artifact down the line.

292 Explicitly, an SSC attack is a three-stage process:

293　　1. **Artifact, step, or actor compromise:** An attacker compromises an element of the SSC
294　　　　to modify an artifact or information of such.

295   2. **Propagation:** The attack propagates throughout the chain.

296   3. **Exploitation:** The attacker exploits the target to achieve their goals (e.g., exfiltration of
297   data, cryptojacking).

3. SSC Security — Risk Factors and Mitigation Measures

This section considers the various risk factors that are applicable to the software development environment and the mitigation measures that can counter those risks.

3.1. Risk Factors in an SSC

The risk factors in an SSC are discussed under the following topics:

- Developer Environment
- Threat Actors
- Attack Vectors
- Attack Targets (Assets)
- Types of Exploits

3.1.1. Developer Environment

Developer workstations and their environments are at risk of compromise and present a fundamental risk to the security of an SSC. The first and best line of defense is to not implicitly trust the developer workstation. Mature SDLC processes accept code and assets into their software configuration management (SCM) mainline and versions branches only after code reviews and scanners are in place. Furthermore, if the developer is working on proprietary software with sensitive IP, additional measures must be put in place to protect the confidentiality of the source code and related material (e.g., architecture diagrams, documentation).

3.1.2. Threat Actors

Threat actors generally come in two types:

- External attackers who seek privileged access to an SSC
- Disgruntled employees or contractors who perpetuate insider threats

External attackers may include foreign adversaries, criminal organizations, and cyber-activists who target an SSC for various reasons, such as espionage or sabotage. Internal attackers pose a significant risk, as they may have insider access to sensitive information — often using legitimate access rights — that allow them to launch attacks or steal confidential information. Additionally, both categories of threat actors may use a variety of techniques to compromise the software development environment and steal or manipulate software, such as phishing, malware, social engineering, and physical access. Therefore, companies should be aware of these risks and take appropriate measures to secure their SSC.

3.1.3. Attack Vectors

Attack vectors in an SSC include:

- Malware

- Social engineering

- Network-based attacks

- Physical attacks

Attack vectors can originate from various sources, including malware attacks on developer workstations, social engineering attacks that target developers, network-based attacks that target the development environment, and physical attacks on the hardware or networks used by developers. These different attack vectors require distinct countermeasures, including endpoint protection software, network security controls, access control policies, and physical security measures. Companies should identify potential risks and vulnerabilities, assess their security posture, and implement appropriate defensive measures to mitigate threats to their software development environment.

3.1.4. Attack Targets (Assets)

The assets targeted under an SSC include:

- Source code

- Credentials

- Sensitive data

A software developer's workstation typically contains various assets, including source code, credentials, and access to sensitive information, such as personally identifiable information (PII), protected health information (PHI), intellectual property (IP), and proprietary information. These assets should be protected, as they are valuable to attackers who may attempt to steal or compromise them. Companies should identify critical assets and implement controls to protect them from unauthorized access, such as access controls, multi-factor authentication, encryption, and data loss prevention (DLP) measures.

3.1.5. Types of Exploits

Exploits in the context of attack vectors and targeted assets in an SSC environment typically include:

- Injection of vulnerabilities or malware into an SSC

- Stolen credentials that grant access to other systems

- Sensitive data leaked

- Injection of malicious code into repositories

- Lack of code integrity in public repositories

362 Threat actors may seek to compromise various components of the software development process,
363 including source code, testing environments, development tools, and build pipelines. They may
364 introduce vulnerabilities, malware, or stolen credentials to gain access to other systems or
365 compromise sensitive data. Such threats can result in financial losses, reputational damage, and
366 legal consequences.

367 To inject malicious code into repositories, attackers may perform an operation called "forking"
368 in GitHub. This operation allows the attacker to copy some repository and make modifications
369 freely outside of the original project. The attacker then initiates a pull request — a request to
370 merge the forked project with the original project. If the project maintainer accepts the request
371 without reviewing the changes and determining them to be suitable, they will merge them into
372 the original project, thus introducing malicious code into the repository.

373 Not all code is written from scratch. When open-source code is used, an artifact or package is
374 often pulled from a repository based on the reputation of the developer or the repository.
375 However, there is no guarantee that pulled code is the same software that the developer authored
376 and checked into their source code repository. The following actions could have potentially
377 occurred, resulting in a lack of assurance or an inability to trust the code:

378 • The source code could have been modified.

379 • Vulnerabilities could have been introduced due to an insecure build system.

380 • Checks, such as scanning and various types of tests (e.g., static, dynamic, or interactive),
381   may have been bypassed in the CI/CD process.

3.2. Mitigation Measures

383 The following generic mitigation measures are applicable to the entire SDLC but are particularly
384 relevant to an SSC:

385 • Patch management

386 • Access control

387 • Malware protection

388 • Secure SDLC

389 • Data protection

390 • Physical security

391 • Audit and monitoring

392 • Adherence to applicable security standards (e.g., regulatory requirements)

393 Organizations can implement various controls to mitigate risks to their software development
394 environment, including regular patch management, access control, malware protection, secure
395 development life cycle (SDLC) practices, data protection measures, physical security controls,
396 and auditing and monitoring tools. They should regularly assess their security posture, identify
397 potential weaknesses and vulnerabilities, and implement appropriate defensive measures to
398 address them. Organizations should also ensure that their software development environment

399   remains compliant with various security standards, such as the OWASP Top Ten, SP 800-53,
400   HIPAA, and PCI DSS.

401   Overall, a secure software development environment can reduce the likelihood of security
402   incidents and ensure the confidentiality, integrity, and availability of software assets and
403   systems. It is crucial to assess security risks and implement appropriate defensive measures to
404   ensure a secure software development environment. The choice of a mitigation approach will
405   depend on the organization's customized threat model. However, all developer systems should
406   meet a minimum baseline for security to ensure that the operating system and applications are
407   kept up to date with the latest security patches, that individual and unshared user accounts are
408   adequately protected, and that proper access controls are enforced when interacting with SCM.

3.2.1. Baseline Security

410   Independent and open-source developers will need to follow best practices to protect their own
411   systems. Government and enterprise environments should establish and adhere to a well-defined
412   security policy that meets regulatory requirements and industry best practices. Since the
413   development of such a policy is out of scope for this document, readers should refer to SP 800-
414   53r5 (Revision 5) [6] for a more complete treatment of this topic.

415   An important responsibility of the developer is to download, evaluate, and integrate open-source
416   components into their projects. There has been a significant increase in malware deployed
417   through software repositories with typo-squatting, compromised repositories, or – in some
418   scenarios — malicious actors legally acquiring repositories.

3.2.2. Controls for Interacting With SCMs

420   Developers also use their workstations to create, edit, and test source code. This process requires
421   developers to pull source code from the SCM, modify the source code, and submit changes
422   (patches) back to the SCM. The proposed changes should adhere to the SDLC processes defined
423   by the organization. Pull access to the software depends on the policies of the software project in
424   question (e.g., open-source projects typically allow anyone to pull, replicate, modify, and share
425   the source code with minimal or copyleft restrictions). Proprietary software vendors often
426   enforce strict rules that describe who is allowed to access the source code and under what
427   conditions. In all cases, write access to the SCM should be considered a high risk and tightly
428   controlled. A mature SDLC process allows developers to propose patches to the SCM, but
429   another developer should perform a code review before the patch is merged. Code analysis tools
430   should be implemented to catch common mistakes, but care should be taken to not inundate the
431   developers with too many false positives to prevent alert fatigue.

432

433 4. CI/CD Pipelines — Background, Security Goals, and Entities to be Trusted

434 DevSecOps is an agile paradigm used for the development and deployment of cloud-native
435 applications. This paradigm consists of a series of stages that takes code from variously sourced
436 repositories (e.g., first-party or in-house, third parties or open-source/commercial) to perform
437 tasks or activities, such as building, packaging, testing, and deploying. The build process is based
438 on application logic-driven dependencies and generates builds using many individual source-
439 code artifacts that are stored in build repositories. The build artifacts are tested and used to
440 generate packages. The generated package artifacts are then stored in designated repositories and
441 scanned before being deployed in testing or production environments. These stages and the
442 various tasks performed at each stage are collectively called CI/CD pipelines. In other words,
443 CI/CD pipelines use processes called workflows to transform source code to deployable
444 packages in production environments. There are several platforms that support these workflows
445 (e.g., GitHub Actions workflows, GitLab Runners, Buildcloud, etc.). A common approach for
446 SSC security in all of these workflows is to generate as much provenance data as possible.

447 From the above description of CI/CD pipelines and associated activities, one can identify the set
448 of security assurance measures that need to be added to those activities:

449 • Internal SSC security practices that are applied during the development and deployment
450   of first party software

451 • Security practices that are applied with respect to the procurement, integration, and
452   deployment of open-source and commercial software modules.

453 Not all artifacts involved are composed of entities developed in-house (i.e., first party). Some
454 components may involve third-party sources.

455 4.1. Broad Security Goals for CI/CD Pipelines

456 There are two security goals in the application of SSC security measures or practices in CI/CD
457 pipelines:

458 1. Incorporate a range of defensive measures to ensure that attackers cannot tamper with
459   software production processes or introduce malicious software updates (e.g., secure
460   platform for build process).

461 2. Ensure the integrity of the CI/CD pipeline artifacts (e.g., repositories) and activities
462   through role definitions and authorizations for actors.

463 The most common approach to security assurance measures for CI/CD pipelines is the
464 introduction of security measures into the CI/CD platform, which allows developers to automate
465 their build, test, and deployment pipelines. There are many open-source CI/CD platforms, such
466 as GitHub Actions.

467 4.2. Entities That Need Trust in CI/CD Pipelines — Artifacts and Repositories

468 Zero trust architectures focus on protecting assets and resources, such as services, the entire
469 application, and hardware systems (e.g., servers). The entities that access these assets — such as
470 users, services, and other servers — are not inherently trusted. Trust needs to be established

471 through the verification of credentials that these entities present through a process called
472 authentication. Based on this authentication, appropriate permissions or access rights are
473 assigned to those entities based on enterprise business policies.

474 In contrast, an SSC focuses on ensuring the integrity of artifacts and the repositories where they
475 are stored because artifacts that travel through various repositories ultimately become the final
476 product. This integrity assurance results in trust.

477 **Table 1** gives examples of entities (i.e., artifacts and repositories) that need to be trusted in
478 typical CI/CD pipelines [7].

479   **Table 1.** Entities that need to be trusted in typical CI/CD pipelines.

Artifact	Repository
First-party code — source code or binary	SCM
Third-party code — open source or commercial	Artifact managers for language, container, etc.
Builds	Build repository
Packages	Package repository

480

481 **5. Integrating SSC Security Into CI/CD Pipelines**

482 In order to outline the strategies for integrating SSC security into CI/CD pipelines, it is necessary
483 to take a closer look at the workflows in each of the two pipelines (i.e., CI pipelines and CD
484 pipelines) and understand their overall security goals.

485 The prerequisites to activating CI/CD pipelines are:

486   • Define the roles for the various actors that operate the various CI/CD pipelines (e.g.,
487   application updaters, package managers, deployment specialists, etc.).

488   • Identify the granular authorizations to perform various tasks, such as generating and
489   committing code to SCMs, generating builds and packages, and checking various
490   artifacts (e.g., builds and packages) into and out of the repositories.

491   • The entire CI/CD pipeline must be automated through the deployment of appropriate
492   tools. The driver tools for CI and CD pipelines are at a higher level, and they invoke a
493   sequence of function-specific tools, such as those for code checkouts from repositories,
494   edits and compilation, code commits, and testing (e.g., SAST, DAST and SAC testers).

495   • CI/CD pipeline activities and associated security requirements are defined for the
496   development and deployment of application code as well as:

497   o Infrastructure as code, which contains details about the deployment platform.

498   o Policy as code and configuration code, which specify runtime settings (e.g.,
499   YAML files)

500 **5.1. Securing Workflows in CI Pipelines**

501 The workflows in the CI pipeline mainly consist of build operations, push/pull operations on
502 repositories (both public and private), software updates, and code commits.

503 The overall security goals for the framework used for securely running CI pipelines include:

504   • The capability to support both cloud-native and legacy software development
505   environments.

506   • Standard compliant evidence structures, such as metadata and digital signatures

507   • Support for multiple hardware and software platforms

508   • Support for infrastructures for generating the evidence.

509 The following subsections consider the SSC security tasks for the various workflows in CI.

510 **5.1.1. Secure Build**

511 The following tasks are required to obtain SSC security assurance in the build process:

512   • Specify policies regarding the build, including (a) the use of a secure isolated platform
513   for performing the build, (b) the tools that will be used to perform the build, and (c) the
514   authentication/authorization required for the developers performing the build process.

515   • Enforce those build policies using an agent or some other means and a policy
516   enforcement engine.

517   • Ensure the concurrent generation of evidence for build attestation to demonstrate
518   compliance with secure build processes during the time of software delivery.

519 A common technique for facilitating the second task is to wrap commands from a CI tool with
520 capabilities to gather evidence and ultimately create an evidence trail of the entire SDLC [8]. The
521 evidence gathered consists of the hash of the final build artifact, files, libraries, and other
522 materials used in the artifacts and all events. This is then signed using a secure PKI distribution
523 system to become the attestation, which provides verifiable proof of the quality of the software to
524 consumers and enables them to verify the quality of that artifact independently from the producer
525 of the software. In this context, the artifact is the build generated by a series of CI process steps.
526 The attestation for a build consists of the following components [9]:

527   1. Environment Attestation: Environment attestation pertains to the inventory of the system
528   at the time when the CI process happens. It generally refers to the platform on which the
529   build process is run. The components of the platform (e.g., compiler, interpreter, etc.)
530   must be hardened, isolated, and secure.

531   2. Process Attestation: Process attestation pertains to the computer programs that
532   transformed the original source code or materials into an artifact (e.g., compilers,
533   packaging tools, etc.) and/or the programs that performed testing on that software (i.e.,
534   code testing tool).

535   3. Materials Attestation: Materials attestation pertains to any raw data and can include
536   configuration, source code, and other data.

537   4. Artifacts Attestation: An artifact is the result or outcome of a CI process. For example, if
538   the CI process step involves running a compiler (e.g., GCC) on a source code written in
539   C, the artifact that will result is an executable binary of that source code. If the step
540   involves running a static application security testing (SAST) tool on the same source
541   code, the artifact that will result will be the "Scan Result." The step that generated it can
542   be a final or intermediate step. An attestation pertaining to this newly generated product
543   falls under the category of artifacts attestation.

544 The signed evidence (i.e., attestation) must be stored securely in a server and can then be used to
545 evaluate policy compliance. A policy is a signed document that encodes the requirements for an
546 artifact to be validated. The policy may include checks as to whether each of the functionaries
547 involved in the CI process has used the right keys to generate the attestations, the required
548 attestations are found, and the methodology to evaluate the attestation against its associated
549 metadata has also been specified. The policy enables the verifiers to trace the compliance status
550 of the artifact at any point during its life cycle.

551 The above capabilities collectively provide the following assurances:

552   • The software was built by authorized persons using the authorized tools (e.g., machines
553   for each step) in the correct sequence of steps.

554   • There is no evidence of potential tampering or malicious activity.

5.1.2. Secure Pull-Push Operations on Repositories

The first SSC security task is to secure source code development practices. In the context of CI/CD pipelines, code resides in repositories, is extracted by authorized developers using a PULL operation, is modified, and is then put back into the repositories using a PUSH operation. To authorize these PULL-PUSH operations, two forms of checks are required:

1. The type of authentication required for developers authorized to perform the PULL-PUSH operations. The request made by the developer must be consistent with their role (e.g., application updater, package manager, etc.).

2. The integrity of the code in the repository can be trusted such that it can be used for further updates.

The various mechanisms for ensuring the trustworthiness of the code in the repository are:

- **PULL-PUSH_REQ-1:** The project maintainer should run automated checks on all artifacts covered in the pull request, such as unit tests, linters, integrity tests, security checks, and more.

- **PULL-PUSH-REQ-2:** Running CI pipelines using external tools (e.g., Jenkins) should be performed only when confidence is established in the trustworthiness of the source-code origin.

- **PULL-PUSH-REQ-3:** The repository or source-code management system (e.g., GitHub) should have built-in protection that incorporates a delay in CI workflow runs until they are approved by a maintainer with write access. This built-in protection should go into effect when an outside contributor submits a pull request to a public repository. The setting for this protection should be at the strictest level, such as "Require approval for all outside collaborators" [10].

- **PULL-PUSH_REQ-4:** If there are no native built-in protections available in the source-code management system, then external security tools with the following features are required:

  o Functionality to evaluate and enhance the security posture of the SCM systems with or without a policy (e.g., OPA) to assess the security settings of the SCM account and generate a status report with actionable recommendations

  o Functionality to enhance the security of the source-code management system (e.g., GitHub, GitLab) by detecting and remediating misconfigurations, security vulnerabilities, and compliance issues

5.1.3. Integrity of Evidence Generation During Software Updates

One important process in an SSC is the software update process, which is typically carried out by a special class of software development tool called software update systems. Ensuring the security of these software update systems plays a critical role in the overall security of an SSC. Threats to software update systems mainly target the evidence generation process so as to erase the trail of updates and prevent the ability to determine whether the updates were legitimate or not.

594   There are several types of software update systems [11]:

595   • Package managers that are responsible for all of the software that is installed on a system

596   • Application updaters that are only responsible for individual installed applications

597   • Software library managers that install software that adds functionality, such as plugins or
598   programming language libraries

599   The primary task performed by a software update system is to identify the files that are needed
600   for a given update ticket and download those files that are trusted. At first glance, it may appear
601   that the only checks needed for establishing trust in downloaded files are the various integrity
602   and authenticity checks performed by verifying the signatures on the metadata associated with
603   individual files or the package. However, the very process of signature generation may be
604   vulnerable to known attacks, so software update systems require many other security measures
605   related to signatures generation and verification.

606   The evolving framework for providing security for software update systems has incorporated
607   many of these required security measures into its specification and prescribed some others for
608   future specifications. A framework is a set of libraries, file formats, and utilities that can be used
609   to secure new and existing software update systems. The following are some of the consensus
610   goals for the framework:

611   • The framework for software update systems should provide protection against all known
612   attacks on the tasks performed by the software update systems, such as metadata (hash)
613   generation, the signing process, the management of signing keys, the integrity of the
614   authority performing the signing, key validation, and signature verification.

615   • The framework for software update systems should provide a means to minimize the
616   impact of key compromise. To do so, it must support roles with multiple keys and
617   threshold or quorum trust (with the exception of minimally trusted roles designed to use a
618   single key). The compromise of roles that use highly vulnerable keys should have
619   minimal impact. Therefore, online keys (i.e., keys used in an automated fashion) must not
620   be used for any role that clients ultimately trust for files they may install [11].

621   • The framework must be flexible enough to meet the needs of a wide variety of software
622   update systems.

623   • The framework must be easy to integrate with software update systems.

624   **5.1.4. Secure Code Commits**

625   Appropriate forms of testing should be performed before code commits, and the following
626   requirements must be met:

627   • Both SAST and DAST tools used in CI/CD pipelines must provide coverage for different
628   language systems used in cloud-native applications.

629   • If open-source modules and libraries are used, dependencies must be detected using
630   appropriate SCA tools, and the security conditions they should meet for their inclusion must
631   also be tested.

632 An SSC security measure required during code commits is the prevention of secrets getting into
633 the committed code. This is enabled by a scanning operation for secrets and results in a feature
634 called push protection [12]. This feature should satisfy the following requirements:

635 • **COMMIT-REQ-1:** If the committed code has an embedded secret, there should be a
636   feature to generate an alert that contains information on the secret type (e.g., personal
637   access token) and location, as well as the methodology to remediate the exposure.

638 • **COMMIT-REQ-2:** Push protection features should be enabled for all repositories
639   assigned to an administrator [13].

640 ## 5.2. Securing Workflows in CD Pipelines

641 Supply chain security measures also apply to controls during the CD process. The following are
642 some due diligence measures that should be used during CD. These due diligence measures can
643 be implemented by defining verification policies for allowing or disallowing an artifact for
644 deployment.

645 • **DEPLOY-REQ-1:** A key deploy time control that can be used is based on build
646   information. If a secure build environment and associated process have been established,
647   it should be possible to specify that the artifact (i.e., container image) being deployed
648   must have been generated by that build process in order to be cleared for deployment.

649   **DEPLOY_REQ-2:** Another deploy time control is to check for evidence that the
650   container image was scanned for vulnerabilities and attested vulnerability findings. This
651   technique enables DevOps teams to implement a proactive container security posture by
652   ensuring that only verified containers are admitted into the environment and remain
653   trusted during runtime [14]. Specifically, it should be possible to allow or block image
654   deployment based on organization-defined policies.

655   The tasks to be performed include:

656   o As soon as a container image is built, it should be scanned for vulnerabilities even
657   before it is pushed to a registry. The early scanning feature can also be built in as part
658   of the local workflows.

659   o There should be tools to manage container images and language packages. The
660   common repository over which both of these activities can be performed should
661   support native artifact protocols, and the tools used should be capable of integration
662   with CD tools, thus making all activities an integral part of automated CD pipelines.

663 • **DEPLOY-REQ-3:** For code that is already in the repository and ready to be deployed, a
664   security scanning sub-feature should be invoked to detect the presence of secrets in the
665   code, such as keys and access tokens.

666 • **DEPLOY-REQ-4:** Before merging pull requests, it should be possible to view the details
667   of any vulnerable versions through a form of dependency review [15].

668 ### 5.2.1. Secure CD Pipeline — Case Study (GitOps)

669 All operations during and after a build in the CI/CD pipeline involve interacting with a central
670 repository (usually GIT). Bitbucket, GitHub, and GitLab are some examples of GIT
671 repositories. The operations are collectively called GitOps and consist of commits, forking, and
672 pull and push requests. In other words, GitOps is an automated deployment process facilitated by
673 open-source tools, such as Argo CD and Flux. GitOps is carried out for both infrastructure code
674 and application code. The usage of GitOps covers the following [16]:

675 • Managing infrastructure as code

676 • Managing and applying cluster configurations

677 • Automating the deployment of containerized applications and their configurations to
678 distributed systems.

679 The following SSC security tasks are to be applied with respect to creating configuration data
680 prior to deployment, capturing all data pertaining to a particular release, modifying software
681 during runtime, and performing monitoring operations:

682 • **GitOps-REQ-1:** The process should rely on automation rather than manual operations.
683 For example, manually configuring hundreds of YAML files to roll back a deployment
684 on a cluster in a Git should be avoided.

685 • **GitOps-REQ-2:** Package managers that facilitate GitOps should preserve all data on the
686 packages that were released, including version numbers of all modules, all associated
687 configuration files, and other metadata as appropriate for the software operational
688 environment.

689 • **GitOps-REQ-3:** Another situation that should be avoided is manually applying changes
690 directly into the nodes with a kubectl edit during runtime. For example, security issues
691 discovered in running applications will need to be remediated in the build process rather
692 than an administrator making changes directly in the cluster. This is to ensure that Git
693 commits remain the single source of truth for what runs in the cluster.

694 • **GitOps-REQ-4:** (Monitoring and Remediation for Drift) — Since the Git repository
695 contains the application definitions and configuration as code, it should be pulled
696 automatically and compared with the specified state of these configurations. For any
697 configurations that deviate from their specified state, the following actions may be
698 performed:

699 o Administrators can choose to automatically resync configurations to the defined state.

700 o Notifications should be sent regarding the differences, and manual remediation
701 should be performed.

702 ### 5.3. SSC Security for CI/CD Pipelines — Implementation Strategy

703 The extensive set of steps needed for SSC security cannot be implemented all at once in the
704 SDLC of all enterprises without a great deal of disruption to underlying business processes and

705 operational costs. Rather, solutions that provide SSC security can be broadly classified into two
706 types [17]:

1. Solutions that ensure SSC security through the following features associated with each
   task in the DevSecOps pipelines:

   a. Verifying that the software is built correctly by ensuring tamper-proof build
      pipelines, such as by providing verified visibility into the dependencies and steps
      used in the build [18]

   b. Including features for the specification of checklists for each step of the delivery
      pipeline to provide guidance for implementation and to check and enforce
      controls for complying with checklists

2. Solutions that ensure integrity and provenance through digital signatures and attestations

716 **6. Summary and Conclusions**

717 This document provided an overview of strategies for integrating SSC security assurance
718 measures with various workflows associated with CI/CD pipelines, which is a methodology in
719 the DevSecOps paradigm that is widely used for the development and deployment of cloud-
720 native applications. However, no recommendations were provided with respect to the specific
721 artifacts and frameworks associated with SSC security, such as SBOMs, code signing, and
722 attestation. This is due to the fact that specifications and the standards associated with them are
723 still evolving as part of projects in government institutions and industry forums.

724

References

[1] Chandramouli R (2022) Implementation of DevSecOps for a Microservices-based Application with Service Mesh. (National Institute of Standards and Technology, Gaithersburg, MD), NIST Special Publication (SP) NIST SP 800-204C. https://doi.org/10.6028/NIST.SP.800-204C

[2] Souppaya M, Scarfone K, Dodson D (2022) Secure Software Development Framework (SSDF) Version 1.1: Recommendations for Mitigating the Risk of Software Vulnerabilities. (National Institute of Standards and Technology, Gaithersburg, MD), NIST Special Publication (SP) NIST SP 800-218. https://doi.org/10.6028/NIST.SP.800-218

[3] EO14028 (2021) *Improving the Nation's Cybersecurity*. Available at https://www.federalregister.gov/documents/2021/05/17/2021-10460/improving-the-nations-cybersecurity

[4] Goud N (2021) *What is Solarigate.* Available at https://www.cybersecurity-insiders.com/what-is-solorigate/

[5] Berger A (2023) *What is Log4Shell ?* Available at https://www.dynatrace.com/news/blog/what-is-log4shell/#:~:text=Log4Shell%20is%20a%20software%20vulnerability,logging%20error%20messages%20in%20applications

[6] Joint Task Force (2020) Security and Privacy Controls for Information Systems and Organizations. (National Institute of Standards and Technology, Gaithersburg, MD), NIST Special Publication (SP) 800-53, Rev. 5. Includes updates as of December 10, 2020. https://doi.org/10.6028/NIST.SP.800-53r5

[7] Lorenc D (2021) *Zero Trust Supply Chain Security*. Available at https://dlorenc.medium.com/zero-trust-supply-chain-security-e3fb8b6973b8

[8] Testify/Witness (2023) *Witness – Secure Your Supply Chain*. Available at https://github.com/testifysec/witness/

[9] Kennedy C (2021) *What is a Software Supply Chain Attestation – and Why do I need it?* Available at https://www.testifysec.com/blog/what-is-a-supply-chain-attestation/

[10] Gelb Y (2023) *Mass Scanning of Popular GitHub Repos for CI Misconfiguration*. Available at https://medium.com/checkmarx-security/mass-scanning-of-popular-github-repos-for-ci-misconfiguration-cd36ad6be788

[11] TUF V1.0.31 (2022) *The Update Framework Specification*. Available at https://theupdateframework.github.io/specification/latest/

[12] Malik Z, Sulakian M (2023) *Push Protection is generally available*. Available at https://github.blog/2023-05-09-push-protection-is-generally-available-and-free-for-all-public-repositories/?utm_source=thenewstack&utm_medium=website&utm_content=inline-mention&utm_campaign=platform

[13] GitHub Docs (2023) *Enabling Security Features for Multiple Repositories*. Available at https://docs.github.com/en/enterprise-cloud@latest/code-security/security-overview/enabling-security-features-for-multiple-repositories

[14] Cloud Build (2023) *Securing Image Deployments to Cloud Run and GKE*. Available at https://cloud.google.com/build/docs/securing-builds/secure-deployments-to-run-gke

768   [15] GitHub Docs (2023) *About dependency review*. Available at
769   https://docs.github.com/en/enterprise-cloud@latest/code-security/supply-chain-
770   security/understanding-your-software-supply-chain/about-dependency-review
771   [16] Williams A (2021) *A Blueprint for Supply Chain Security*. Published by Newstack
772   [17] Crane D (2023) *Five Stages for A Secure Software Supply Chain*. Available at
773   https://danacrane.medium.com/five-stages-for-a-secure-software-supply-chain-
774   f8420841cc3a
775   [18] CyRise (2023) *Supply Chain Security with Ensignia*. Available at
776   https://medium.com/@cyrise/supply-chain-security-with-ensignia-483c1d872639
777

778 **Appendix A. Mapping of Recommended Security Tasks in CI/CD Pipelines to**
779 **Recommended High-Level Practices in SSDF**

780 **Table 2.** Mapping of recommended CI/CD pipeline security tasks to SSDF practices

Section	Recommended Security Tasks in CI/CD Pipeline	Recommended High-Level Practice in SSDF
5.1.1 Secure Build — Policies for Build Process and Mechanisms to Enforce Policies **5.2 Securing Workflows in CD Pipelines**	• Specify policies regarding the build. The policies include (a) the use of secure isolated platform for performing the build, (b) the tools that will be used to perform the build, and (c) the authentication/authorization required for developers performing the build process. • Enforce those build policies using an agent or some other means and a policy enforcement engine. **DEPLOY-REQ-1:** A key deploy time control that can be used is based on build information. If a secure build environment and associated process have been established, it should be possible to specify that the artifact (i.e., container image) being deployed must have been generated by that build process in order to be allowed to be cleared for deployment. **DEPLOY_REQ-2:** Another deploy time control is to check for evidence that the container image was scanned for vulnerabilities and attested vulnerability findings. This technique enables DevOps teams to implement a proactive container security posture by ensuring that only verified containers are admitted into the environment and remain trusted during runtime [14]. Specifically, it should be possible to allow or block image deployment based on organization-defined policies. The tasks to be performed include: As soon as a container image is built, it should be scanned for vulnerabilities even before it is pushed to a registry. The early scanning feature can also be built in as part of the local workflows. There should be tools to manage container images and language packages. The common repository over which both of these activities can be performed should support native artifact protocols, and the tools used should be capable of integration with CD tools, thus making all activities an integral part of automated CD pipelines.	**Define Security Requirements for Software Development (PO.1):** Ensure that the security requirements for software development are known at all times so that they can be taken into account throughout the SDLC and duplication of effort can be minimized. This includes requirements from internal sources (e.g., the organization's policies, business objectives, and risk management strategy) and external sources (e.g., applicable laws and regulations).

Section	Recommended Security Tasks in CI/CD Pipeline	Recommended High-Level Practice in SSDF
	• **DEPLOY-REQ-3:** For code that is already in the repository and ready to be deployed, a security scanning sub-feature should be invoked to detect the presence of secrets in the code, such as keys and access tokens. • **DEPLOY-REQ-4:** Before merging pull requests, it should be possible to view the details of any vulnerable versions through a form of dependency review.	
5 Integrating SSC Security in CI/CD Pipelines	The prerequisites for activating CI/CD pipelines are: • Define roles for various actors operating the various CI/CD pipelines (e.g., application updaters, package managers, deployment specialists, etc.) • Identify the granular authorizations to perform various tasks, such as generating and committing code to SCMs, generating builds and packages, and checking various artifacts (e.g., builds and packages) into and out of the repositories.	**Implement Roles and Responsibilities (PO.2):** Ensure that everyone inside and outside of the organization involved in the SDLC is prepared to perform their SDLC-related roles and responsibilities throughout the SDLC.
5 Integrating SSC Security in CI/CD Pipelines	A prerequisite for activating CI/CD pipelines is: • The entire CI/CD pipeline must be automated through the deployment of appropriate tools. The driver tools for CI and CD pipelines are at a higher level, and they invoke a sequence of function-specific tools, such as those for code checkouts from repositories, edits and compilation, code commits, and testing (e.g., SAST, DAST and SAC testers).	**Implement Supporting Toolchains (PO.3):** Use automation to reduce human effort and improve the accuracy, reproducibility, usability, and comprehensiveness of security practices throughout the SDLC, as well as provide a way to document and demonstrate the use of these practices. Toolchains and tools may be used at different levels of the organization, such as organization-wide or project-specific, and may address a particular part of the SDLC, like a build pipeline.
5.1.4 Secure Code Commits	A prerequisite operation before code commits is appropriate forms of testing. The following requirements must be met: • Both SAST and DAST tools used in CI/CD pipelines must provide coverage for different language systems used in cloud-native applications. • If open-source modules and libraries are used, dependencies must be detected using appropriate SCA tools, and the security conditions they should meet for their inclusion must also be tested.	**Define and Use Criteria for Software Security Checks (PO.4):** Help ensure that the software resulting from the SDLC meets the organization's expectations by defining and using criteria for checking the software's security during development.

Section	Recommended Security Tasks in CI/CD Pipeline	Recommended High-Level Practice in SSDF
5.1.1 Secure Build Policies for Build Process and Mechanisms for Enforcement of Policies	**Already covered under meeting requirements for PO.1.** In addition: 1. Environment Attestation: Environment attestation pertains to the inventory of the system when the CI process happens. It generally refers to the platform on which the build process is run. This platform must be hardened, isolated, and secure.	**Implement and Maintain Secure Environments for Software Development (PO.5):** Ensure that all components of the environments for software development are strongly protected from internal and external threats to prevent the environments or the software in them from being compromised. Examples of environments for software development include development, build, test, and distribution environments.
5.1.2 Secure PULL-PUSH Operations on Repositories	All forms of code used in SDLC reside in repositories. Code is extracted from these repositories by authorized developers using a PULL operation, modified, and then put back into the repositories using a PUSH operation. To authorize these PULL-PUSH operations, two forms of checks are required. 1. The type of authentication required for developers authorized to perform the PULL-PUSH operations. The request made by the developer must be consistent with their role (e.g., application updater, package manager, etc.). 2. The integrity of the code in the repository can be trusted such that it can be used for further updates.	**Protect All Forms of Code From Unauthorized Access and Tampering (PS.1):** Help prevent unauthorized changes to code, both inadvertent and intentional, that could circumvent or negate the intended security characteristics of the software. For code that is not intended to be publicly accessible, this helps prevent theft and may make it more difficult or time-consuming for attackers to find vulnerabilities in the software.
5.1.3 Integrity of Evidence Generation During Software Updates (To provide the assurance to acquirers that the software they get is legitimate, steps are taken to protect the integrity of evidence generation tasks)	1. The framework for software update systems should provide protection against all known attacks on the tasks performed by the software update systems, such as metadata (hash) generation, the signing process, the management of signing keys, the integrity of the authority performing the signing, key validation, and signature verification. 2. The framework for software update systems should provide a means to minimize the impact of key compromise. To do so, it must support roles with multiple keys and threshold or quorum trust (with the exception of minimally trusted roles designed to use a single key). The compromise of roles that use highly vulnerable keys should have minimal impact. Therefore, online keys (i.e., keys used in an automated fashion) must not be used for any role that clients ultimately trust for files they may install [11].	**Provide a Mechanism for Verifying Software Release Integrity (PS.2):** Help software acquirers ensure that the software they acquire is legitimate and has not been tampered with.

Section	Recommended Security Tasks in CI/CD Pipeline	Recommended High-Level Practice in SSDF
	3. The framework must be flexible enough to meet the needs of a wide variety of software update systems. 4. The framework must be easy to integrate with software update systems.	
5.2.1 Secure CD Pipeline — Case Study (GitOps)	The following SSC security tasks are to be applied with respect to creating configuration data prior to deployment, capturing all data pertaining to a particular release, modifying software during runtime, and performing monitoring operations: • **GitOps-REQ-2:** Package managers that facilitate GitOps should preserve all data on the packages that were released, including version numbers of all modules, all associated configuration files, and other metadata as appropriate for the software operational environment.	**Archive and Protect Each Software Release (PS.3):** Preserve software releases in order to help identify, analyze, and eliminate vulnerabilities discovered in the software after release.
5.1.2 Secure PULL-PUSH Operations on Repositories (Implements secure coding and build processes to improve security through various checks during PULL-PUSH operations)	• **PULL-PUSH_REQ-1:** The project maintainer should run automated checks on all artifacts covered in the pull request, such as unit tests, linters, integrity tests, security checks, and more. • **PULL-PUSH-REQ-2:** Running CI pipelines using external tools (e.g., Jenkins) should be performed only when confidence is established in the trustworthiness of the source-code origin. • **PULL-PUSH-REQ-3:** The repository or source-code management system (e.g., GitHub) should have built-in protection that incorporates a delay in CI workflow runs until they are approved by a maintainer with write access. This built-in protection should go into effect when an outside contributor submits a pull request to a public repository. The setting for this protection should be at the strictest level, such as "Require approval for all outside collaborators" [10] . • **PULL-PUSH_REQ-4:** If there are no native built-in protections available in the source-code management system, then external security tools with the following features are required: ○ Functionality to evaluate and enhance the security posture of the SCM systems with or without a policy (e.g., OPA) to assess the security settings of the SCM account and generate a status report with actionable recommendations.	**Create Source Code by Adhering to Secure Coding Practices (PW.5):** Decrease the number of security vulnerabilities in the software and reduce costs by minimizing vulnerabilities introduced during source code creation that meet or exceed organization-defined vulnerability severity criteria.

Section	Recommended Security Tasks in CI/CD Pipeline	Recommended High-Level Practice in SSDF
	o Functionality to enhance the security of the source-code management system (e.g., GitHub, GitLab) by detecting and remediating misconfigurations, security vulnerabilities, and compliance issues.	
5.1.1 Secure Build (Addresses the requirements for PW.6 through security requirements for the build platform)	Environment Attestation: Environment attestation pertains to the inventory of the system at the time when the CI process happens. It generally refers to the platform on which the build process is run. This platform components (e.g., compiler, interpreter, etc.) must be hardened, isolated, and secure.	**Configure the Compilation, Interpreter, and Build Processes to Improve Executable Security (PW.6):** Decrease the number of security vulnerabilities in the software and reduce costs by eliminating vulnerabilities before testing occurs.
5.1.4 Secure Code Commits	A prerequisite operation before code commits is appropriate forms of testing. The following requirements must be met: • Both SAST and DAST tools used in CI/CD pipelines must provide coverage for the different language systems used in cloud-native applications. • If open-source modules and libraries are used, dependencies must be detected using appropriate SCA tools, and the security conditions they should meet for their inclusion must also be tested.	**Test Executable Code to Identify Vulnerabilities and Verify Compliance With Security Requirements (PW.8):** Identify vulnerabilities so that they can be corrected before the software is released. Using automated methods lowers the effort and resources needed to detect vulnerabilities and improves traceability and repeatability. Executable code includes binaries, directly executed bytecode and source code, and any other form of code that an organization deems executable.
5 Integrating SSC Security into CI/CD Pipelines	CI/CD pipeline activities and associated security requirements are defined for the development and deployment of application code as well as: • Infrastructure as code, which contains details about the deployment platform • Policy as code and configuration code, which specify runtime settings (e.g., YAML files)	**Configure Software to Have Secure Settings by Default (PW.9):** Help improve the security of the software at the time of installation to reduce the likelihood of the software being deployed with weak security settings, thus putting it at greater risk of compromise.

781

782

783 **Appendix B. Justification for the Omission of Certain Measures Related to SSDF**
784 **Practices in This Document**

785 **Table 3.** Justification for the omission of certain SSDF practices

SSDF Practice	Justification for Omission
Produce Well-Secured Software (PW) PW1 through PW4, PW7	These practices pertain to secure software design, review of the design, and software reuse. CI/CD pipelines focus on setting up the environment for secure development and deployment and not software design per se.
Respond to Vulnerabilities (RV) RV1 through RV3	Vulnerability management strategies are at the organization policy level and are not specific to CI/CD pipelines.

786