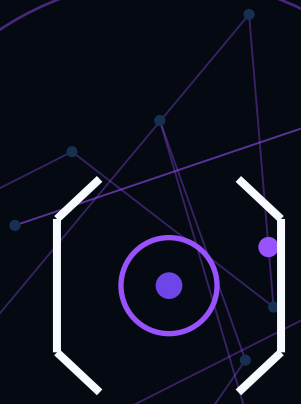


ENGINEERING STANDARDS LIBRARY / PLATFORM ENGINEERING

DevSecOps, Release Engineering, and Supply Chain Standard

Reproducible builds, artifact provenance, controlled promotion, software supply-chain assurance, and evidence-bearing releases.

PERMANENT DOCUMENT ID

MYTHOS-PLT-0001

PUBLICATION

Candidate Standard
Version 1.0.0-candidate

ASSURANCE PROFILE

Standard / Advanced
High Assurance

PERMANENT PUBLICATION IDENTITY

DevSecOps, Release Engineering, and Supply Chain Standard

DOCUMENT ID
MYTHOS-PLT-0001

VERSION
1.0.0-candidate

STATUS
Candidate Standard

PUBLISHER
Mythos Systems

PUBLICATION DATE
2026-07-24

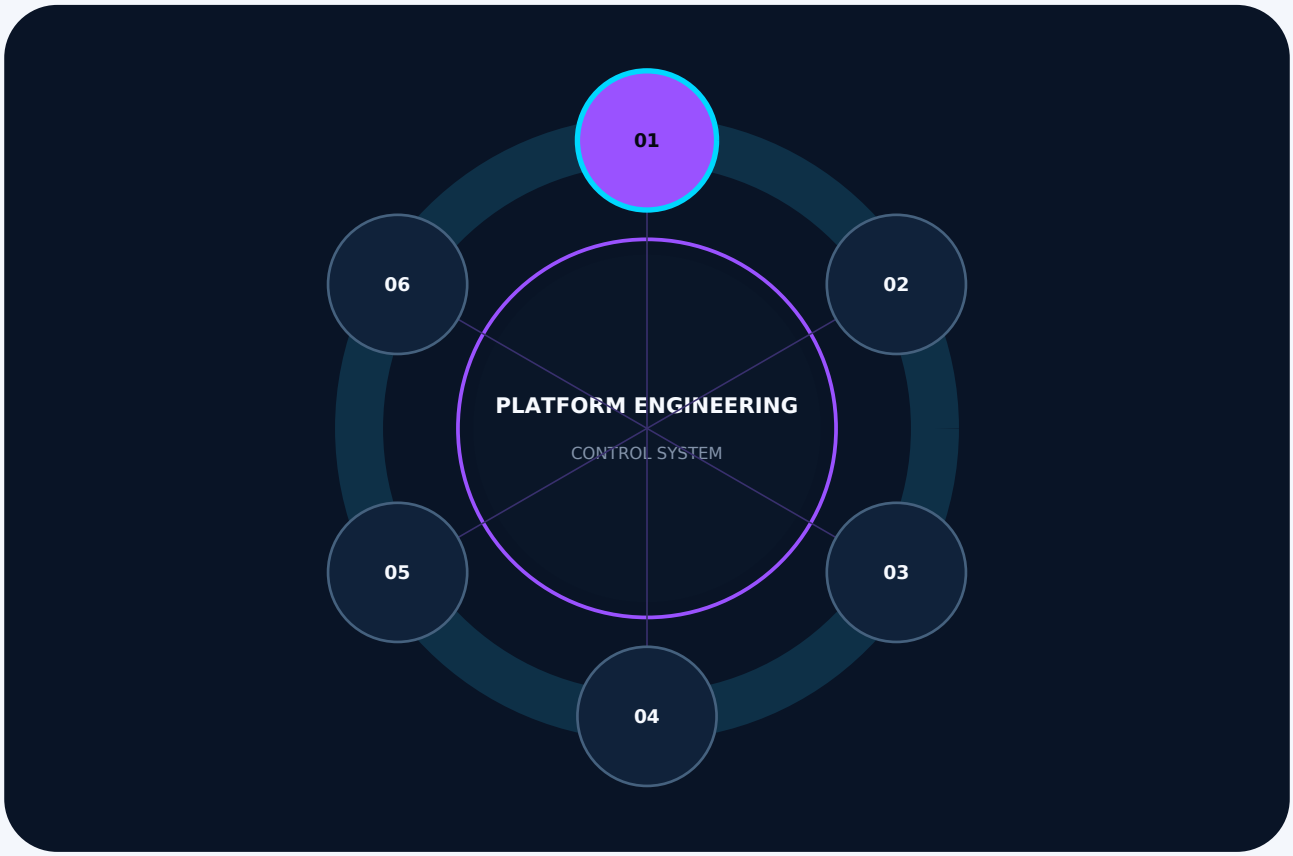
SCHEDULED REVIEW
2027-01-24

12	6	4	1
ATOMIC CRITERIA	ASSURANCE VIEWS	IMPLEMENTATION PROFILES	PERMANENT IDENTITY

Publication doctrine. This publication establishes durable guidance or requirements for its defined scope. Interpretation must preserve the permanent document identity, recorded evidence, controlled exceptions, and formal revision history.

DOMAIN CONTROL SYSTEM

MYTHOS-PLT-0001 inside the platform engineering standard constellation



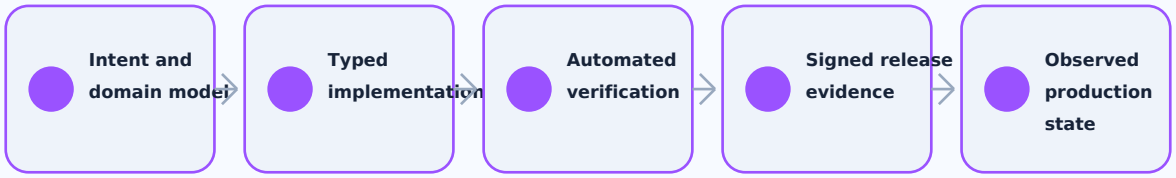
Connected assurance	Architecture, implementation, operations, evidence, and lifecycle are evaluated as one controlled system.
Specialization rule	Companion standards may add precision, but they cannot silently weaken this publication's requirements.

EXECUTIVE POSITION

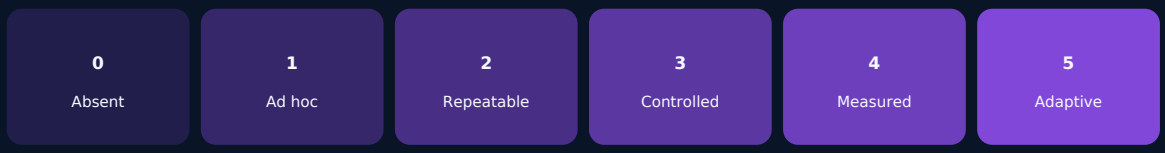
DevSecOps, Release Engineering, and Supply Chain Standard

The architecture of software delivery has failed.

Evidence-bearing delivery path



Assurance maturity spectrum



CONTENTS

Publication navigation

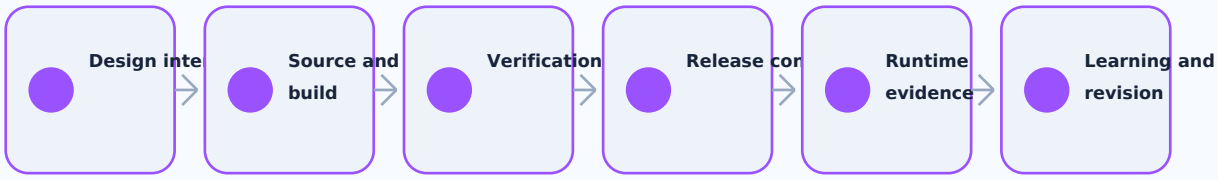
CONTROL TOPOLOGY

Assurance model and evaluation surface

The standard is assessed as an evidence-bearing system. Capability scores do not replace mandatory gates, implementation proof, or operational validation.

Capability family	Evaluation nodes
Build Isolation and Runner Security (Weight: 20%)	1
Provenance and SLSA Compliance (Weight: 20%)	1
Dependency Management and Confusion (Weight: 15%)	1
SBOM/AIBOM Generation and Verification (Weight: 15%)	1
Artifact Signing and Transparency (Weight: 15%)	1
Shift-Left Security and Policy Gates (Weight: 15%)	1
Suite Composition	2
Additional Evaluation Dimension	4

Lifecycle control loop



EVALUATION INDEX

Criteria and evidence map

#	Capability family	Criterion
1	Build Isolation and Runner Security (Weight: 20%)	Build Environment Integrity
2	Provenance and SLSA Compliance (Weight: 20%)	Artifact Integrity
3	Dependency Management and Confusion (Weight: 15%)	Resolution Security
4	SBOM/AIBOM Generation and Verification (Weight: 15%)	Software Transparency
5	Artifact Signing and Transparency (Weight: 15%)	Trust Verification
6	Shift-Left Security and Policy Gates (Weight: 15%)	Continuous Enforcement
7	Suite Composition	MSCS-Isolation
8	Suite Composition	MSCS-Provenance
9	Additional Evaluation Dimension	Supply Chain Dimensions
10	Additional Evaluation Dimension	The DevSecOps Doctrine
11	Additional Evaluation Dimension	Scoring Model
12	Additional Evaluation Dimension	Suite Composition

NORMATIVE PUBLICATION

Requirements, evaluation methodology, and implementation guidance

The following body preserves the substantive source standard while normalizing publication status, identity, navigation, and visual presentation.

Executive Mandate

The architecture of software delivery has failed.

Organizations build critical infrastructure using CI/CD pipelines that pull arbitrary dependencies from the public internet at build time. They use long-lived, static credentials on shared build runners. They treat the build server as a trusted environment, and when a sophisticated attacker (e.g., SolarWinds, `xz-utils`) injects malicious code during the compilation phase, the resulting artifacts are signed and deployed blindly across the enterprise.

This standard exists because:

1. The Build Server is the Crown Jewel: If an attacker compromises the CI/CD runner, they compromise every artifact it produces. Build environments MUST be ephemeral, isolated (hermetic), and possess zero outbound internet access during the compilation step.
2. Provenance Must Be Cryptographic, Not Implicit: Trusting a binary because "it came from our Jenkins server" is a fatal flaw. Every artifact MUST carry cryptographically signed, in-toto attestations detailing exactly how it was built, what sources were used, and what dependencies were resolved (SLSA Level 3+).
3. Public Registries are Hostile Territory: Package managers that fall back to public registries (npm, PyPI) for internal package names are vulnerable to Dependency Confusion attacks. Builds MUST resolve dependencies exclusively from verified, internal, private mirrors.
4. Security is Not a Final Gate: Scanning a compiled binary for CVEs right before deployment is too late. Security and policy enforcement MUST be shifted left, occurring at the PR stage and continuously during the build process, blocking non-compliant code from ever reaching the registry.
5. AI Models are Supply Chain Artifacts: Downloading unverified `.gguf` or `.safetensors` files from HuggingFace is the equivalent of running `curl | bash`. AI model weights MUST be treated as untrusted binaries, requiring SBOMs (AIBOMs), vulnerability scanning, and cryptographic signing.

This document establishes the Mythos Supply Chain Standard (MSCS), a comprehensive, evidence-based methodology for evaluating DevSecOps and release engineering pipelines against hermetic isolation, provenance verification, and zero-trust dependency management.

Scope and Applicability

In Scope

This standard covers the evaluation of:

1. CI/CD pipelines and release engineering (GitHub Actions, GitLab CI, Jenkins, Tekton)
1. Supply-chain Levels for Software Artifacts (SLSA) framework compliance
1. Software Bill of Materials (SBOM) and AI Bill of Materials (AIBOM) generation
1. Cryptographic artifact signing and transparency logs (Sigstore, Cosign, Rekor)
1. Hermetic builds and dependency confusion prevention
1. Ephemeral runner architecture and OIDC federation (zero static secrets)

Out of Scope

1. General infrastructure-as-code (covered in MYTHOS-PLT-0002)
1. General container runtime security (covered in MYTHOS-SEC-0012)
1. Developer identity and passkeys (covered in MYTHOS-ID-0001)

Audience

1. DevSecOps and platform engineers building CI/CD infrastructure
1. Security architects evaluating supply chain threats
1. Release engineers managing artifact registries
1. Open-source maintainers securing public packages
1. Standards bodies seeking a reference supply chain methodology

Definitions and Taxonomy

Supply Chain Dimensions

Dimension	Definition
SLSA (Supply-chain Levels for Software Artifacts)	A security framework providing a graduated set of requirements ensuring the integrity of software artifacts throughout the software supply chain.
Hermetic Build	A build process that is completely isolated from the internet and external state, fetching source code and dependencies only from verified, internal, cached mirrors.
Provenance Attestation	A cryptographically signed metadata file (in-toto format) generated by the build platform, attesting to the exact source, build parameters, and dependencies used to create an artifact.
Dependency Confusion	A supply chain attack where a malicious package with the same name as an internal private package is published to a public registry, tricking the build into downloading the malicious code.
SBOM/AIBOM	Software/AI Bill of Materials. A formal, machine-readable inventory of software components, dependencies, and (for AI) model weights, training data provenance, and licenses.

The DevSecOps Doctrine

> A build server with internet access is a compromised server. A binary without provenance is a Trojan horse. A public package is a stranger. If the dependency resolution is not isolated, the supply chain is poisoned.

Evaluation Methodology

Scoring Model

Each capability is scored from 0 to 5.

Score	Meaning
0	Fails basic task; shared runners, static secrets, internet access during build, no SBOMs
1	Basic CI/CD with automated tests, but no supply chain controls
2	Functional scanning, but relies on implicit trust of the build server
3	Solid production performance; hermetic builds, SLSA L3, Sigstore signing, internal registry mirrors
4	Strong performance; reproducible builds, policy-as-code gates, AIBOMs for AI models
5	Best-in-class; sets the standard for mathematically verified, zero-trust software delivery

Weighting

Category	Weight	Rationale
Build Isolation & Runner Security	20%	Shared runners with internet access are the primary attack vector
Provenance & SLSA Compliance	20%	Implicit trust in binaries allows compromised artifacts to deploy
Dependency Management & Confusion	15%	Public registry fallbacks guarantee eventual code injection
SBOM/AIBOM Generation & Verification	15%	You cannot secure what you do not inventory
Artifact Signing & Transparency	15%	Unsigned artifacts can be tampered with in the registry
Shift-Left Security & Policy Gates	15%	Scanning at deployment is too late; must block at PR/Build

Total: 100%

A system MUST score at least 3.0 in Build Isolation and Provenance to be considered for production use.

Evaluation Categories

Build Isolation and Runner Security (Weight: 20%)

Build Environment Integrity

Is the CI/CD pipeline treated as a highly privileged, untrusted environment?

Score	Criteria
0	Shared, persistent build runners; long-lived SSH keys; outbound internet access
1	Private runners, but run as root with internet access
2	Ephemeral runners spun up per job, but still possess outbound internet access
3	Hermetic builds: runners have zero outbound internet; dependencies fetched from internal mirrors
4	Ephemeral runners with OIDC federation (zero static secrets); strict seccomp/AppArmor profiles
5	Perfect isolation: formally verified build boundaries, zero network egress capability, cryptographic attestation of runner OS state

Provenance and SLSA Compliance (Weight: 20%)

Artifact Integrity

Can the organization cryptographically prove how, where, and from what an artifact was built?

Score	Criteria
0	No provenance; binaries trusted based on "which Jenkins job built it"
1	Basic build logs retained, but easily forgeable
2	SLSA Level 1-2: Provenance generated, but not cryptographically signed
3	SLSA Level 3: Signed in-toto provenance bound to the artifact; build platform isolates jobs
4	SLSA Level 4: Reproducible builds verified by two independent platforms; provenance strictly enforced at deployment
5	Perfect integrity: formally verified build pipeline, zero ability to deploy artifacts without valid provenance, mathematical proof of source-to-binary equivalence

Dependency Management and Confusion (Weight: 15%)

Resolution Security

Is the build system protected against public registry poisoning?

Score	Criteria
0	Build pulls dependencies directly from public internet (npm, PyPI)
1	Private registry used, but falls back to public if package not found
2	Public fallback disabled for internal namespaces, but no vulnerability scanning
3	Strict scoped registries: all builds resolve *only* from internal, cached mirrors; zero public access

Score	Criteria
4	Continuous vulnerability scanning of the internal mirror; auto-quarantine of compromised packages
5	Perfect management: formally verified dependency graphs, zero typosquatting potential, cryptographic hashing of all resolved dependencies

SBOM/AIBOM Generation and Verification (Weight: 15%)

Software Transparency

Is there a machine-readable inventory of every component, including AI models?

Score	Criteria
0	No inventory; dependencies tracked manually in READMEs
1	Basic SBOM generated for release artifacts, but not verified
2	SBOM (CycloneDX/SPDX) generated in CI; stored alongside artifact
3	AIBOM generated for AI models (tracking weights, training data licenses); SBOMs verified at deployment gate
4	Dependency drift detection: build fails if SBOM changes without PR approval
5	Perfect transparency: formally verified component inventory, zero undocumented dependencies, cryptographic binding of SBOM to artifact hash

Artifact Signing and Transparency (Weight: 15%)

Trust Verification

Are artifacts cryptographically signed and logged publicly?

Score	Criteria
0	Unsigned artifacts deployed directly to production
1	Artifacts signed with long-lived GPG keys
2	Keyless signing via Sigstore/Cosign using OIDC identity
3	Transparency log (Rekor) integration for all signed artifacts
4	Runtime (Kubernetes/Host) refuses to execute artifacts lacking a valid Rekor entry
5	Perfect trust: formally verified signature propagation, zero ability to deploy unsigned code, cryptographic proof of audit trail

Shift-Left Security and Policy Gates (Weight: 15%)

Continuous Enforcement

Is security a blocking gate throughout the development lifecycle?

Score	Criteria
0	Security scanned manually right before release
1	Basic SAST/DAST running in CI, but non-blocking
2	Scanning blocks PRs on critical vulnerabilities

Score	Criteria
3	Policy-as-code (OPA) evaluating SBOMs, licenses, and SLSA provenance in CI
4	Developer IDE integration preventing insecure code from ever being committed
5	Perfect enforcement: formally verified policy bounds, zero ability to bypass security gates, mathematical proof of release readiness

The Mythos Supply Chain Suite (MSCS)

Traditional DevOps benchmarks measure build throughput (builds per minute). The MSCS evaluates hermetic isolation, provenance verification, and dependency confusion resistance.

Suite Composition

MSCS-Isolation

1. Egress Test: 100+ tests attempting to `curl` external endpoints during the build script, verifying the hermetic runner blocks the connection.
1. Secret Persistence Test: 50+ tests checking for static credentials in the runner environment, verifying OIDC federation is strictly enforced.

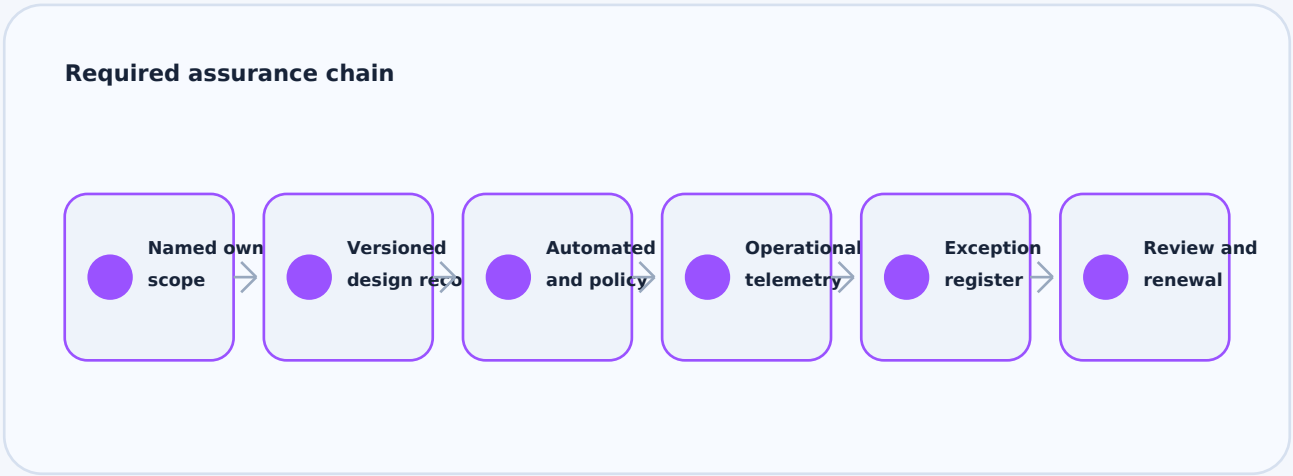
MSCS-Provenance

1. Tampered Artifact Deployment: 100+ tests attempting to deploy an artifact with invalid or missing SLSA provenance, verifying the Kubernetes admission controller blocks it.
1. Dependency Confusion Test: 50+ tests publishing a malicious package with an internal name to a public registry, verifying the build resolver ignores it.

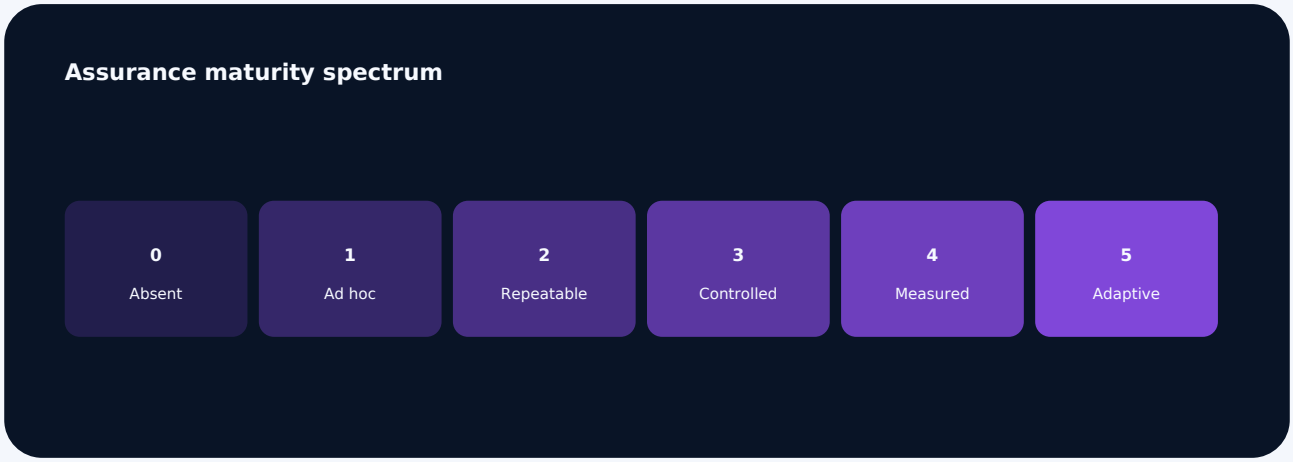
Execution Protocol

ASSURANCE AND ADOPTION

Implementation evidence and review gates



A passing score is not sufficient without current evidence. High-assurance adoption requires traceable design decisions, reproducible builds, independently reviewable tests, controlled secrets, runtime observability, failure exercises, and a dated exception register.



Current authority register

Authority	Version / snapshot	Applicability	Review by
Kubernetes Project: Kubernetes Documentation	v1.36 current documentation	High	2026-10-22
SLSA Project: Supply-chain Levels for Software Artifacts Specification	1.2 current specification snapshot	Moderate	2026-10-22
SPDX Project: SPDX Specification	3.0.1 stable; 3.1-RC1 under review	High	2026-10-22
OWASP CycloneDX: CycloneDX Bill of Materials Standard	1.7	High	2027-01-24
OpenTelemetry Project: OpenTelemetry Documentation and Specification	Rolling specification snapshot; profiles public alpha and declarative configuration stable in 2026	High	2026-10-22
Cloud Native Computing Foundation: Cloud Native Landscape and Project Documentation	Rolling catalog snapshot	Moderate	2026-08-23

Research limitation. Official documentation establishes published interfaces and declared behavior. It does not prove the performance, security, interoperability, cost, or operational suitability of a specific implementation. Those claims require controlled testing and retained evidence.