CleanStart

Dependency Management & Attack Surface Reduction

Reducing risk & surface through architecture - first isolation

Abstract

Modern software environments rely on thousands of open-source components, each introducing new layers of risk. The dependency explosion has transformed containers from lightweight application packages into complex, interconnected systems that are difficult to verify and secure. **CleanStart Platform** reduces this risk by isolating dependencies, minimizing the attack surface, and compiling all components from verified source. This paper explains how dependency management and attack surface reduction form the foundation of modern container security.

Introduction

Containers revolutionized how applications are built and deployed by enabling consistency and portability. However, this consistency often hides a deeper problem: an uncontrolled web of software dependencies. Each package, library, and sub-dependency introduces potential vulnerabilities and uncertain provenance.

As organizations scale, this dependency complexity becomes unmanageable. Teams are forced to choose between developer convenience and security assurance. **CleanStart Platform** eliminates this trade-off by ensuring that every dependency included in the container foundation is isolated, verified, and rebuilt from trusted source code under a controlled supply chain.

The Dependency Explosion

In today's development ecosystem, an average enterprise container can include hundreds of dependencies, many of which are indirect or nested within other packages. Studies show that up to 90 percent of container image content originates from open-source software maintained outside the organization's visibility or control.

Traditional vulnerability scanning identifies risks after integration, often too late in the development cycle. Furthermore, most scanners only detect known Common Vulnerabilities and Exposures (CVEs) without verifying whether the package's origin or integrity can be trusted.

The result is an invisible and growing attack surface. A single outdated library can compromise an entire production environment, even if the primary application code is secure.

The True Cost of Dependencies

Unmanaged dependencies not only increase exposure to vulnerabilities but also impose measurable operational and compliance costs. Each dependency adds potential licensing requirements, compatibility concerns, and additional testing overhead.

When organizations rely on prebuilt binaries or public registries, they inherit both the dependencies and the trust model of unknown third parties. These packages may be compiled with insecure flags, linked against outdated libraries, or modified without verifiable provenance.

Over time, dependency sprawl leads to inconsistent builds, untraceable vulnerabilities, and failed compliance audits. As supply chain regulations such as SLSA and Executive Order 14028 enforcement expand, dependency transparency has shifted from a best practice to a requirement.

Attack Surface Reduction as a Security Principle

Attack surface reduction (ASR) is the practice of eliminating unnecessary components, interfaces, and privileges to minimize potential entry points for exploitation. In container security, ASR requires building environments that contain only the essential runtime elements—no unused libraries, tools, or packages.

CleanStart Platform operationalizes this principle through its compile-from-source, hermetic build process. Each image is constructed from verified source code in an isolated environment, ensuring that no external packages, network dependencies, or unverified binaries are introduced. This removes untrusted intermediaries from the software supply chain and guarantees that each dependency included is intentional, verified, and accounted for.

The CleanStart Way

CleanStart Platform enforces dependency integrity at three layers of control:

Hermetic Builds: The build environment is fully sealed, preventing network access during compilation and ensuring reproducible results.

Source-Level Verification: Every dependency is compiled directly from verified upstream source, eliminating the risks of tampered precompiled binaries.

Isolation and Minimalism: Only explicitly declared dependencies are included in the final container image, removing hidden transitive inclusions that increase attack surface.

Architecture of Secure Dependencies

(Build-to-Run Pathway)

Application Container

Compiled with CleanStart toolchain

Application-Specific Dependencies

- Located in `/usr/share/[app]/lib`
- Private namespace isolation
- RPATH: `$ORIGIN/.../lib/private`
- Invisible to system scanners

CleanStart Foundation

- Custom glibc (hardened, optimized)
- FIPS 140-3 cryptographic modules
- Minimal system files only

Hermetic Build Environment

Complete Linux namespace isolation

Zero network access

Cryptographic attestation

Enhanced SBOM Generation

- Complete dependency inventory
- Isolation context for each component
- Cryptographic signatures and attestations

Architecture Overview

Figure 1. Architecture of Secure Dependencies (Build-to-Run Pathway)

CleanStart Platform enforces dependency integrity from source to runtime through isolated builds, verified provenance, and minimal execution layers.

This approach results in containers that are smaller, faster, and more secure by design. Each layer of the container has a defined purpose and cryptographically verifiable lineage, making it impossible for unvetted components to enter production workloads.

Quantifiable Benefits

Organizations adopting **CleanStart Platform** for dependency management achieve measurable results across their software lifecycle:

Drastic Vulnerability Reduction: Containers show near-zero known CVEs in production due to controlled dependency inclusion.

Improved Performance: Leaner images reduce boot times, resource usage, and patching overhead.

Simplified Compliance: Automatic SBOM (Software Bill of Materials) generation and verifiable provenance streamline audit preparation.

Faster Response to Vulnerabilities: Because dependencies are rebuilt from known sources, patched updates can be generated in hours, not weeks.

Predictable Builds: Reproducibility ensures that test, staging, and production environments are cryptographically identical.

This alignment between security, performance, and transparency allows organizations to shift from reactive vulnerability management to proactive assurance.

Integration into Developer Workflows

CleanStart Platform integrates seamlessly into existing DevOps pipelines without imposing additional overhead on developers. Developers continue to build and deploy as usual, but now on a foundation that ensures dependency trust.

Build artifacts are automatically signed and traced using in-toto attestations and SLSA Level 4 provenance records. This provides both developers and auditors with continuous evidence that every build component originated from a validated source and was compiled under secure, repeatable conditions.

By embedding security into the build process rather than treating it as a post-deployment concern, **CleanStart Platform** aligns developer agility with enterprise compliance.

Conclusion

Dependency management is no longer a developer convenience issue; it is a foundational security requirement. As containers become the standard execution unit for enterprise software, unverified dependencies represent one of the largest unaddressed risks in modern infrastructure.

CleanStart Platform transforms dependency management into a verifiable process by rebuilding all dependencies from trusted source, enforcing hermetic isolation, and minimizing the attack surface. This architectural approach ensures that container environments are not only efficient but also inherently secure.

By integrating dependency integrity at the foundation, organizations move closer to a future where security is an outcome of design, not an afterthought of defense.

References

NIST

(2024)

Software Supply Chain Security Guidance (NIST SP 800-218)

SLSA Framework

(2025)

Supply Chain Levels for Software Artifacts

Gartner Research

(2023)

Modernizing Container Security for the Enterprise

Coalfire

(2025)

Dependency Risk in Cloud-Native Architectures

Red Hat

(2024)

Building Trust Through Minimal Base Images

End of Document