CleanStart

# Retrofitting Security vs Building It In

## The Container Hardening Dilemma



2026.03.24

## Abstract

Container security practices generally follow one of two architectural models. The first approach accepts pre-built distribution packages and attempts to harden them after assembly through vulnerability scanning, configuration adjustments, and runtime controls. The second approach embeds security during compilation by building container components directly from verified source code with hardened compilation parameters.

Retrofit hardening improves security posture relative to unmodified images but cannot modify fundamental properties of pre-compiled binaries. Security-critical compilation decisions—including memory protection flags, control-flow integrity mechanisms, and stack defenses—are embedded in binaries at build time and cannot be introduced through post-build hardening.

Building containers from verified source code establishes security characteristics during compilation. Hardened compiler configurations, hermetic build environments, and reproducible build processes produce binaries whose security properties are intrinsic rather than externally applied.

This architectural distinction determines whether container security remains a reactive operational process or becomes a deterministic property of the build system.


## 1. Container Security as an Architectural Choice

Container security ultimately depends on how software artifacts are produced. Every container image consists of compiled binaries originating from upstream source repositories and distribution packaging systems. The build process therefore represents the primary point at which security properties become embedded in deployed artifacts.

Two architectural approaches dominate container security practices.

The first approach relies on pre-built distribution packages obtained from public repositories such as Debian, Ubuntu, or Alpine. Organizations assemble container images from these packages, then attempt to improve security through scanning, patching, and runtime controls.

The second approach builds container components directly from verified source code within controlled build environments. Security protections are applied during compilation, embedding defensive mechanisms into binary artifacts before they are distributed.

These two models produce fundamentally different security outcomes. Retrofit hardening attempts to improve security after binaries already exist, while source-based compilation establishes security characteristics at the point where binaries are created.

## 2. Limits of Retrofit Hardening

Most container security practices follow a retrofit model. Organizations begin with publicly available base images and apply security improvements during or after image assembly.

Typical hardening processes include:

- Vulnerability scanning to identify known CVEs
- Removal of unnecessary packages and services
- Implementation of non-root execution policies
- Use of multi-stage builds to reduce image size
- Application of runtime security controls such as SELinux or AppArmor

These techniques provide measurable improvements. Removing unnecessary components reduces attack surface, and runtime isolation mechanisms can limit exploitation impact.

However, retrofit hardening cannot alter the security characteristics embedded in pre-compiled binaries.

Distribution packages are compiled by maintainers who must prioritize compatibility across multiple environments. Compilation flags, memory-protection configurations, and

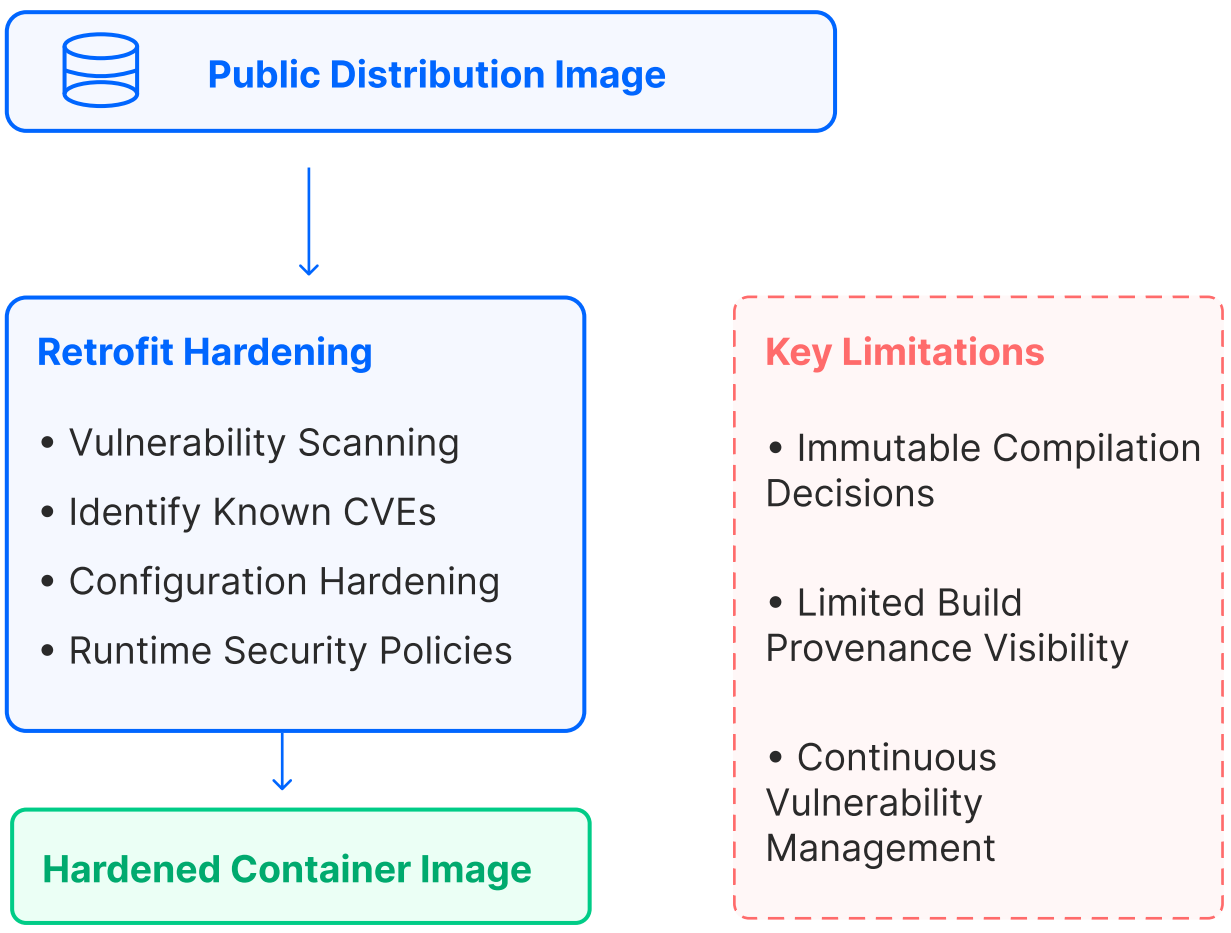
optimization parameters are chosen for general usability rather than context-specific security. Once binaries are produced, these decisions become immutable properties of the software artifact.

A container image may therefore contain binaries that:

- Lack advanced memory safety protections
- Omit modern stack protection mechanisms
- Exclude control-flow integrity safeguards
- Include unnecessary functionality compiled into the binary

These weaknesses cannot be detected through vulnerability scanning because they are not discrete vulnerabilities. They are architectural properties introduced during compilation.

As a result, retrofit hardening improves security posture but cannot fundamentally transform the security characteristics of the underlying software.



## 3. Security Embedded at Compilation

Building container components from verified source code establishes security properties at the point where binaries are created. Instead of accepting pre-compiled binaries, this process retrieves source code directly from upstream repositories and compiles it using controlled toolchains.

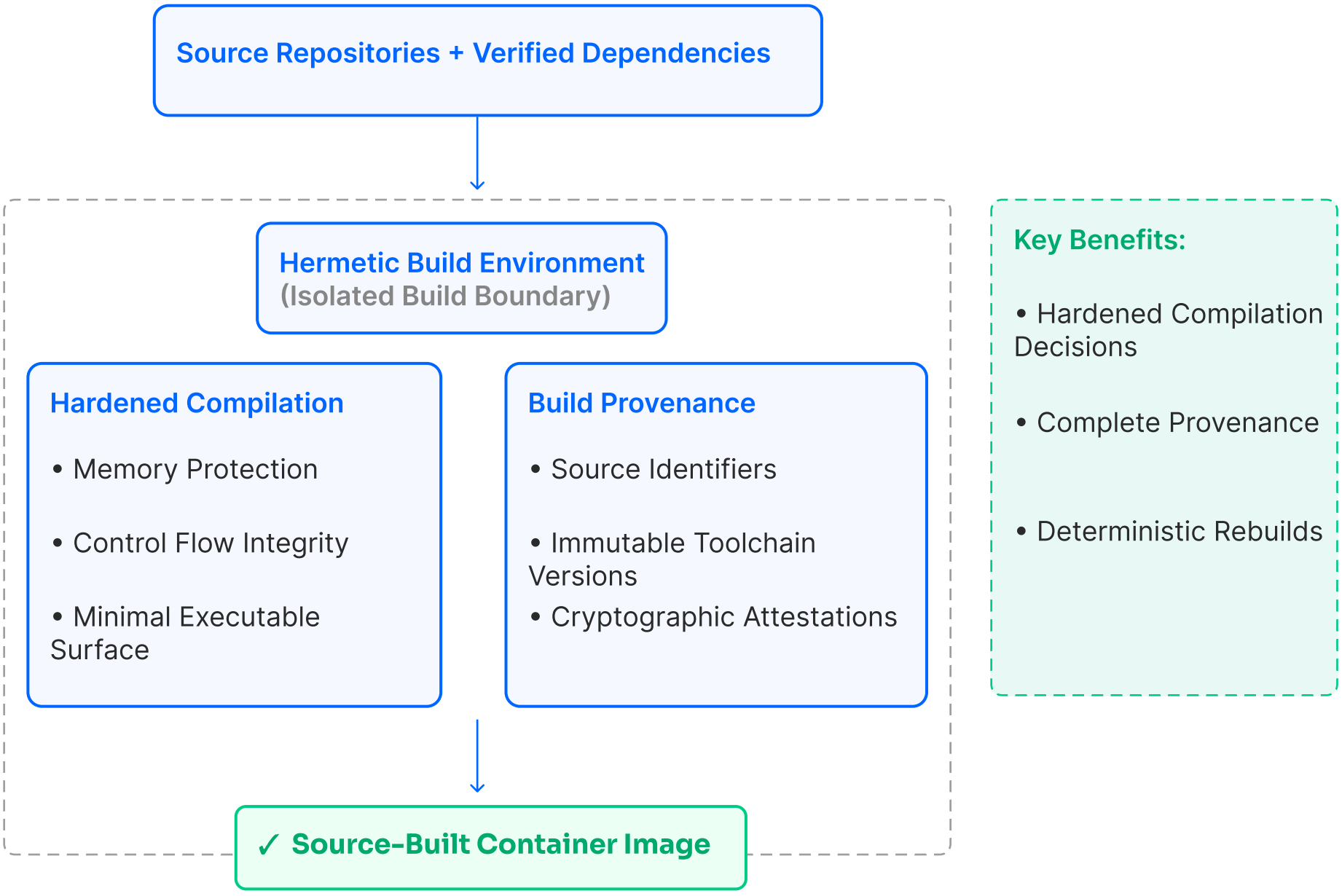
Compilation environments can apply hardened compiler configurations across all components, including:

- Memory protection through FORTIFY_SOURCE
- Stack clash protection mechanisms
- Control-flow integrity (CFI) enforcement
- Position-independent executable generation
- Additional compile-time safety checks

These mechanisms introduce defensive capabilities directly into binary artifacts. Because the protections exist within the compiled code itself, they cannot be bypassed through simple configuration changes or package substitutions.

Source-based compilation also allows organizations to control feature inclusion. Unnecessary modules and optional components can be excluded during compilation, reducing attack surface before artifacts are produced.

The result is a container image whose binaries incorporate security protections as intrinsic characteristics rather than external configuration layers.




## 4. Provenance and Compliance Implications

Supply-chain security frameworks increasingly require verifiable documentation of how software artifacts are produced. Regulatory guidance and security standards emphasize build provenance, reproducibility, and source code traceability.

Building containers from source code enables comprehensive provenance documentation. Each compiled component can be traced to specific upstream repositories and commit identifiers. Build environments can generate cryptographically-signed attestations documenting:

- Source repository origins
- Exact commit versions
- Compilation parameters
- Toolchain versions
- Build timestamps and environments

These records enable verification that artifacts were produced from trusted sources using controlled build procedures.

Frameworks such as SLSA require tamper-evident documentation of build processes and verifiable links between source code and produced artifacts. Hermetic build environments and reproducible compilation support these requirements by ensuring that identical inputs generate identical outputs.

Organizations relying on pre-built distribution packages must instead depend on upstream maintainers to provide equivalent transparency. In many cases, detailed compilation parameters and build environments are not independently verifiable.

Source-based compilation therefore enables stronger supply-chain assurance and simplifies compliance validation.

## 5. Operational Sustainability

The operational implications of these two models diverge significantly over time.

Retrofit hardening requires continuous vulnerability management cycles:

- Scan container images for vulnerabilities
- Analyze scanner results
- Wait for distribution maintainers to release patches
- Rebuild container images with updated packages
- Redeploy affected workloads

This cycle repeats whenever new vulnerabilities are disclosed. Security teams must continually monitor upstream package updates and coordinate remediation activities across production environments.

Source-based build systems enable a different operational model. Automated build pipelines monitor upstream source repositories and security advisories. When vulnerabilities are identified, affected components can be patched directly from upstream sources and recompiled using existing hardened configurations.

This approach allows organizations to regenerate updated container images immediately after patches become available, rather than waiting for distribution package updates. Automated build pipelines can incorporate patch monitoring, compilation, validation testing, and artifact publishing into a continuous process.

Over time, operational effort shifts from manual vulnerability remediation to automated build maintenance.

## 6. CleanStart Implementation

CleanStart implements a source-first container architecture built on a purpose-built container operating system compiled entirely from verified source code.

Every component within CleanStart OS is compiled using hardened compilation configurations that embed security protections directly into binary artifacts. The build pipeline applies consistent memory protection mechanisms, stack defenses, and control-flow integrity protections across all packages.

Compilation occurs within hermetic build environments that isolate build processes from external network inputs. These environments ensure reproducible builds and prevent supply-chain tampering during compilation.

Source verification programs validate upstream repositories through cryptographic signature verification and commit integrity checks. Build processes generate cryptographic attestations documenting the complete compilation pipeline.

Automated build pipelines monitor upstream security advisories and integrate patches as they become available. Updated container images are produced through deterministic rebuilds using the same hardened configurations, enabling rapid security response while preserving reproducibility.

## Conclusion

Container security ultimately depends on how software artifacts are produced. Hardening techniques applied after image assembly can reduce attack surface and improve configuration hygiene, but they cannot alter security characteristics embedded within compiled binaries.

Building containers from verified source code enables security protections to be introduced during compilation, embedding defensive mechanisms directly into binary artifacts. Hermetic build environments, reproducible builds, and cryptographic provenance further strengthen supply-chain assurance.

By establishing security properties at the point of compilation, organizations can shift container security from a reactive operational process to a deterministic build architecture.

## References


NetRise.

Red Hat.

Open-Source Initiative.