CleanStart

Six Layers of Unbreakable Defence

How CleanStart's overlapping architecture delivers comprehensive, end-to-end protection against the full spectrum of software supply chain attacks.

Build clean. Run secure.

EXECUTIVE SUMMARY

The Attack Surface Has Become Systemic

Software supply chain attacks used to be rare and sophisticated, the kind aimed at defence contractors by nation-state actors. That era is over. These attacks are now the main offensive strategy for a wide and growing set of adversaries, from ransomware syndicates to state-sponsored APT groups.

The numbers point to a structural shift

$60B

Global cost of software supply chain attacks in 2025, rising to $138B by 2031 (Cybersecurity Ventures).

2x

Supply chain attacks doubled year-on-year in 2025, with 877,522 malicious packages detected in open source registries

95%

Of vulnerabilities found in transitive dependencies, not the packages developers explicitly install (Endor Labs)

267

Days is the industry average time to detect a supply chain breach. SolarWinds went undetected for 14 months

Scale is only part of the story. The deeper change is in the nature of the attack surface itself. Modern software is no longer built, it is assembled.

The average enterprise container image pulls in hundreds of open source libraries, each with its own dependency tree. Every upstream maintainer account, every package registry, and every CI/CD pipeline is a potential entry point. The trust model that allows organisations to scale has become the attack surface they cannot see.

No single control is sufficient against this threat profile. **CleanStart is architected to provide six independent, overlapping layers of defence**. Together they cover the full range of supply chain attacks: malicious code injected into trusted packages, zero-day exploits aimed at unknown vulnerabilities in production, and the unmeasured security and licensing risk buried in transitive dependencies. Each layer defeats a distinct category of attack. Together, they eliminate the attack surface that adversaries depend on.

This is Zero Trust principle applied to the software supply chain. The principle is simple: never trust, always verify. It already reshaped network and identity security. CleanStart extends it to the artifacts an organisation actually runs.

The industry default is implicit trust in any package that is signed, named, or popular. CleanStart replaces that with verified trust. A component earns its place only when its identity, intent, and integrity have been verified. Verified trust is what carries an organisation forward on its Zero Trust journey, from the network and identity layers down to the container and its contents.

> **Bad actors proved software supply chain attacks are now their most reliable weapon. Isolated incidents became an integrated, industrialised campaign."**
>
> — Sonatype, 2026 State of the Software Supply Chain Report

Six Layers of Unbreakable Defence

How CleanStart's Layers Work Together

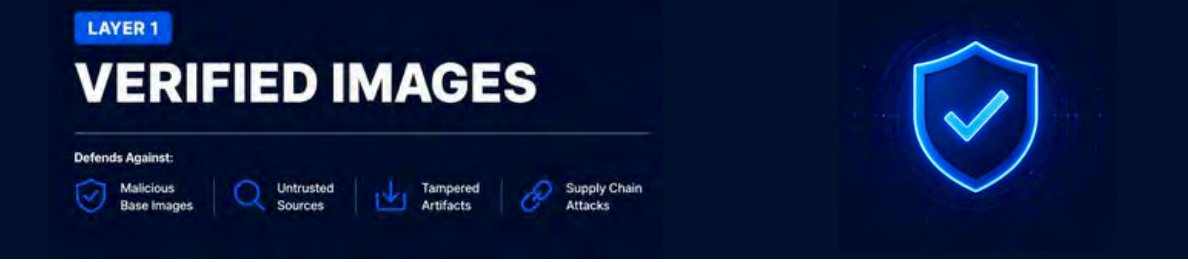
Most security solutions address one dimension of the supply chain threat. A scanner detects known CVEs. A SIEM monitors runtime behaviour. A WAF filters inbound traffic. Each is a point solution for a point problem.

When a new attack category appears, point solutions fail quietly. A novel injection technique, a zero-day in an unmaintained library, or a backdoor planted two years before discovery can all slip past them.

CleanStart's architecture is different because it is structural, not detective. Each of the six layers modifies what is physically present in the container: what code runs, who is trusted to have authored it, what tools exist, what the filesystem allows, what dependencies are present, and what remains unknown.

When a layer removes something an attack needs, that whole category of attack fails. It fails every time, whether or not a scanner has ever seen the signature.

LAYER	THREAT DEFEATED	MECHANISM
Layer 1 **Verified Images**	Malicious code, backdoors, trojanised updates	Maintainer & source verification (intent, identity, integrity); cryptographic provenance; source-build
Layer 2 **Hardened Images**	Known CVEs, unpatched libraries, inherited base-layer risk	Zero known CVEs at publication, 7-day Critical SLA, source compilation
Layer 3 **Hardening**	Misconfiguration, privilege escalation, lateral movement	Build-level hardening, capability minimization, non root default, CIS Benchmark as auditable baseline
Layer 4 **Shell-less & Read-only**	Zero-day exploits, RCE weaponisation, persistence	No shell, no package manager, immutable filesystem; memory-safe (Rust) init and core utilities
Layer 5 **Minimal Dependencies**	Bloated dependency surface, merged library vulnerabilities	Only required components present; aggregate vulnerability surface collapsed before scanning
Layer 6 **Full Dependency Visibility**	Transitive/hidden vulnerabilities, dependency confusion, unknown licensing liability	SBOM-driven full-tree analysis; every component known, versioned, vuln-checked, and licence identified



The Attack: Trust as a Weapon

The defining trait of modern supply chain attacks is that they arrive through trusted channels.

The 2024 XZ Utils backdoor (CVE-2024-3094, CVSS 10.0) is the clearest case. An attacker spent two years building the trust of an overworked maintainer, submitting helpful patches and building rapport, before gaining commit rights to a compression library used in millions of Linux systems. The backdoor was designed to allow unauthenticated remote code execution through SSH on every major Linux distribution.

In 2025, attackers compromised maintainer accounts for 18 widely used npm packages, including chalk and debug. Those packages were downloaded over 2.6 billion times per week. The attackers injected a crypto wallet stealer that only activated in browser contexts.

The malicious version was semantically valid, signed by an authenticated account, and distributed over HTTPS. Traditional signature verification passed. The attack propagated to thousands of repositories and CI/CD pipelines before detection.

More recently, on March 31, 2026, the popular HTTP client Axios was attacked via a compromised maintainer account. Two newly published npm packages reached out to command-and-control infrastructure. Researchers attributed the attack to the North Korean state actor Sapphire Sleet. The industry takes 267 days on average to detect such a breach. Socket, using behavioural analysis, caught the Axios compromise within 6 minutes of publication.

THE CORE PROBLEM

Your package manager, your CI/CD pipeline, and your container registry all operate on a trust model. They verify that a package was signed by its maintainer. They do not confirm that the maintainer was not compromised, and they do not confirm that the published code is free of a malicious payload.
When the trusted source becomes the attack vector, verification of the source signature is insufficient. Verification of the people behind the source, and of the content itself, is required.

CleanStart's Defence: Verified People, Verified Provenance, Verified Content

CleanStart does not consume open source packages from public registries and trust the source. Every component in a CleanStart image is compiled from verified source code in a hermetically sealed build environment. That environment has no network access during compilation and no way to pull in unexpected dependencies, and it produces a cryptographically attested build artifact that records exactly what went in and what came out.

The deeper problem is blind trust in the open source world. Pulling a package from an upstream registry and compiling it from source does not solve the supply chain problem. It only moves the trust boundary one step earlier, from a pre-built binary to the source code and the people who wrote and committed it.
Open source is, by default, trust extended to maintainers and committers an organisation has never met and cannot vouch for.

Source compilation is necessary, but it is not sufficient. Being precise about why is what separates this layer from generic build-from-source claims.

Compiling from verified source in a hermetically sealed environment eliminates one specific vector: the injection of undeclared payload into a pre-built binary. If you never consume a pre-compiled artifact, you cannot inherit a backdoor hidden inside one.

But compiling from source is still an act of trust in the people who wrote and committed that source. Those maintainers and committers may be unknown, compromised, or impostors.

XZ Utils proves the point. The malicious logic was present in the distributed source itself: a build-stage macro and binary test fixtures committed by a maintainer the ecosystem had been cultivated to trust. A naive source-build pipeline compiles that backdoor straight into the output. Building from source, by itself, would not have stopped it.

CleanStart therefore treats verification as an active, ongoing discipline applied to the source and its authors before any code is trusted:

Intent

Does the code do only what it claims, or does it contain hidden malicious behaviour? Behavioural and structural analysis examines what the code actually does, regardless of how it is described or who submitted it.

Identity

Is this maintainer or committer who they claim to be? Is the contribution consistent with their established history? Does the chain from published artifact back to source and tag hold without substitution? This catches the compromised or impersonated maintainer that signature verification waves through.

Integrity

Does the artifact cryptographically match the verified source, with no unaccounted-for inclusions, attested end-to-end through the build pipeline?

This capability is built to evolve. Adversary techniques change constantly, so CleanStart's verification backend keeps pace rather than freezing against a fixed signature set.
Every CleanStart image ships with a complete SBOM that attests to the provenance of every component. A downstream consumer can verify mathematically that the image they are running was built from the source they expect, by the build system they trust, with no unaccounted-for inclusions.

ATTACK VECTOR	CLEANSTART DEFENCE
Maintainer account compromised; malicious code injected into trusted package	Identity + intent verification: the maintainer-behaviour change and anomalous contribution are evaluated, not just the signature
Backdoor present in distributed source (XZ Utils model)	Intent analysis evaluates what the source actually does; provenance verification catches source-to-tag substitution
Impostor account publishing under a plausible identity	Identity verification against established maintainer history; inconsistencies flagged before trust is extended
Trojanised CI/CD artifact (Axios 2026 model)	Build provenance chain: every artifact is cryptographically attested from verified source
Self-replicating malware in open source ecosystem (ShaiHulud)	No registry dependency: CleanStart builds from verified source, not public registries

The Attack: The Inherited Vulnerability Surface

Most enterprise container images are assembled from upstream distributions such as Alpine, Debian, Ubuntu, and Red Hat UBI, then layered with application dependencies. A 2024 study by NetRise found that public container images carry between 50 and 600 CVEs before a single line of application code is added. These are inherited vulnerabilities from the base operating system and its bundled libraries.

This creates a structural problem that scanning alone cannot solve. A scanner identifies CVEs present in an image. The remediation path is to wait for the upstream distribution to patch the dependency, pull a new base image, rebuild, and redeploy. Across a large image catalog this is operationally intensive; for complex or proprietary base layers it may take weeks or months. During that window, every deployed instance carries known, exploitable vulnerabilities.

AI-accelerated discovery tools have compressed the exploitation window dramatically. Per Mandiant's M-Trends 2026 report, the mean time to exploit a disclosed vulnerability has dropped to negative seven days. Exploitation now begins before the patch is public. Exposure is no longer measured in weeks. It is measured in hours.

THE PATCH DEPENDENCY PROBLEM

Standard container security operates on a dependency chain:
Upstream maintainer patches the library → distribution packages the patch → organisation pulls the new base image → CI/CD rebuilds and redeploys.

Each step introduces delay. In an AI-accelerated threat environment, the total lag across this chain routinely exceeds the exploitation window. The organisation is perpetually behind.

CleanStart's Defence: Zero Known CVEs at Source, Structured SLA

CleanStart images are not assembled from upstream distributions. They are compiled directly from verified source code, selecting only the components the application requires .There are no bundled tools, no unnecessary libraries, and no inherited packages that serve no function in the target workload.

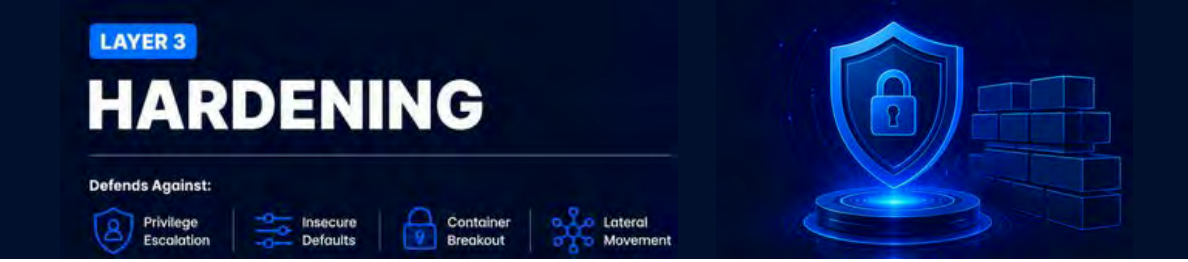
This is the distroless principle taken to its logical extreme: if a component is not required for the application to function, it is not present in the image.

The result is a structural reduction in the CVE surface before any scanning occurs. CleanStart images carry zero known CVEs at the point of publication. When new CVEs are disclosed that affect components present in CleanStart images, the remediation clock starts immediately against a published SLA:

- Critical vulnerabilities — 7 days to patched image publication
- High / Medium / Low — 14 days to patched image publication

This SLA is contractual and tracked against every published image across all 19,200+ variants (spanning versions, architectures, FIPS configurations, and environment targets). CleanStart's structured SLA converts patch management from an unpredictable operational burden into a predictable, contractually-backed commitment.

ATTACK VECTOR	CLEANSTART DEFENCE
CVE disclosed in base layer OS library; organisation waits on upstream patch	Source-compiled images: no inherited base-layer CVEs; zero known CVEs at publication
Known vulnerability exploited before enterprise patch cycle completes	7-day Critical SLA / 14-day High-Medium-Low SLA: contractual, tracked, predictable
Scanner-driven triage overwhelms security team (10,000+ CVEs typical)	Structural surface reduction: only required components present; CVE count near zero
Old vulnerable image version left running in production	19,200+ variants maintained; SLA applies across all architectures and configurations



The Attack: The Misconfiguration Gap

Security research consistently finds that misconfiguration, not novel exploitation, is the main vector for container and cloud-native breaches. Containers running as root when they need not. Writable filesystems where read-only suffices. Processes get capabilities they never use. Ports stay open to services that could be turned off.

These are not sophisticated vulnerabilities. They are configuration choices made at build time that widen the attack surface for no reason.

REAL MISCONFIGURATION ATTACK PATTERNS

Containers running as root: If the container is compromised, the attacker inherits root-equivalent access on the host, enabling container breakout via kernel exploits.

Misconfigured /tmp as executable: The attacker writes a payload to /tmp and executes it, bypassing application-layer controls.

Unrestricted SSH access + weak password policy: A trivial brute-force entry point that enables lateral movement across the environment.

Excessive Linux capabilities granted: A process with CAP_SYS_ADMIN can perform a wide range of privileged operations, converting a limited exploit into system control.

CleanStart's Defence: Hardening at Source, Measured Against CIS

Hardening at CleanStart is a defence-in-depth posture applied at build time. It is not a single checklist, and it is not CIS compliance on its own. CIS is the externally auditable baseline. It is one pillar of the hardening, not the whole of it.

Because hardening is architected from source rather than assessed and remediated afterward, it is structural and permanent. It does not drift between scans. It cannot be overridden by accident in a deployment configuration. And it does not depend on an operations team following a checklist correctly every time.

The hardening posture is built from source and includes:

- **Build-level hardening**
  Compiler and linker hardening flags, fixed at compile time so they cannot drift.
- **Non-root process execution**
  Enforced at image build time, not left to deployment configuration.
- **No unnecessary Linux capabilities**
  Minimum privilege for the workload.

Strict file permission settings
Aligned to CIS Level 2; filesystem mount points configured read-only where application operation does not require writability.

Removal of all tooling not required
No shells, no package managers, no debugging utilities. This also strips the remote-management and SSH attack surface from images that do not need it

CIS Benchmark compliance
The consensus-driven Docker and Kubernetes control set serves as the measurable, auditable standard the above is verified against.

Hardening complements the other layers. A container that cannot run as root is harder to break out of even if an exploit fires. A container with no shell and no capabilities cannot be used as a pivot point even if the application is vulnerable. The hardening layer compresses the blast radius of every other attack category.

ATTACK VECTOR	CLEANSTART DEFENCE
Container running as root; exploit grants host-equivalent privilege	Non-root execution enforced at image build time; cannot be overridden at deploy
Privilege escalation via excessive Linux capabilities (CAP_SYS_ADMIN)	Minimal capability set baked in; no unnecessary capabilities present
/tmp executable; attacker writes and executes payload	Filesystem permissions hardened to CIS Level 2; /tmp non-executable
Insecure SSH configuration enables brute-force lateral movement	No SSH daemon present; no remote management tooling in production image
Configuration drift between baseline and production	Hardening is structural, built in at source rather than applied after the build, and measured against the CIS Benchmark

The Attack: The Zero-Day Problem

Layers 1, 2, and 3 address threats that are known in some sense: malicious code that can be identified, CVEs that have been catalogued, misconfigurations that have been documented. Layer 4 addresses the category that cannot be addressed reactively.

These are vulnerabilities that have no CVE yet, exploits no scanner has seen, and zero-days that an attacker, increasingly an AI model, discovers and weaponises faster than defenders can respond.

The 2026 deployment of Claude Mythos Preview by Anthropic showed that AI systems can now autonomously discover zero-day vulnerabilities across major operating systems and web browsers, build working exploit chains, and validate them. The cost is far below a human expert engagement, and the trend is accelerating. With mean time to exploit at negative seven days, the patch cycle cannot close the gap. The application will contain exploitable vulnerabilities before defenders know they exist.

When an AI-generated RCE exploit fires, its success depends entirely on what it finds on the other side of the exploited function. The standard post-exploitation sequence needs four things:

1. A shell to execute commands in.
2. Tools such as curl, wget, or bash to fetch second-stage payloads.
3. Writable filesystem paths to drop malware and establish persistence.
4. The ability to execute the written code.

Remove any one of these primitives and the exploit chain breaks. Remove all of them and the exploit achieves nothing beyond triggering the vulnerable function.

THE EXPLOIT CHAIN CLEANSTART SEVERS

Step 1: AI discovers a zero-day RCE in application code. This step succeeds; CleanStart does not prevent vulnerability discovery.

Step 2: Exploit fires; attempts to spawn reverse shell. BLOCKED: no shell binary exists. clnimg-init is the only entrypoint.

Step 3: Exploit tries to run OS utilities such as curl or wget. BLOCKED: no package manager and no OS tools are present.

Step 4: Exploit attempts to write payload to disk. BLOCKED: the root filesystem is read-only at the kernel level.

Step 5: Exploit attempts to establish persistence. BLOCKED: there no writable paths outside memory-backed application directories.

Result: The vulnerability was triggered. The exploit achieved nothing. Engineering teams patch during normal maintenance windows.

CleanStart's Defence: Shell-less Architecture, Immutable Filesystem, Memory-Safe Core

CleanStart replaces the traditional container entrypoint, usually a shell, with clnimg-init: a statically compiled, hardened init process that launches the application process and nothing else. There is no /bin/sh, no bash, no dash. There is no package manager. There are no standard OS utilities. The execution environment contains precisely what the application requires to function, and nothing an attacker requires to operate.

The root filesystem is mounted read-only. Write access is restricted to explicitly defined, memory-backed paths required for application runtime: temporary files, application logs, runtime state. No path in the container filesystem can receive a dropper, a malware binary, a modified shared library, or a persistence mechanism. When the container restarts, every memory-backed path is reset to its original state.

The few components that remain are themselves hardened against attack. CleanStart's init process (clnimg-init) and core container utilities are implemented in Rust, a memory-safe language . This removes the entire class of memory-corruption vulnerabilities, including buffer overflows, use-after-free, and out-of-bounds access, that has long made C-based system tooling a soft target.
What is present is not only minimal; it is memory-safe.

Taken together with Layer 5, this is a systematic exercise in attack surface reduction. Minimal dependencies remove what need not be present; shell-less, read-only execution removes the attacker's operating environment; and memory-safe Rust hardens what remains.
The cumulative result is a container that is, to a substantial extent, impregnable. There is little left to attack, and what little remains resists the classic exploitation primitives.

The AISI's 2026 evaluation of Claude Mythos Preview reinforces the underlying principle: architectural barriers stop even highly capable AI-driven attackers when the terrain they need to operate in is simply not present. Removing that terrain defeats even the most capable AI-driven exploitation.

ATTACK VECTOR	CLEANSTART DEFENCE
RCE exploit fires; attempts to spawn reverse shell	No shell binary exists; exploit drops into a void
Post-exploitation attempts to download second-stage malware	Minimal capability set baked in; no unnecessary capabilities present
Attacker writes persistence mechanism to disk	Read-only filesystem; kernel blocks all unauthorised writes
Malware modifies shared library to intercept future requests	Filesystem immutable; no library can be overwritten
AI discovers novel zero-day and weaponises it in hours	Zero-day can trigger but cannot be converted into operational impact
Container compromised and used as pivot for lateral movement	No shell, no tools, no writable paths; container is a dead end for the attacker
Memory-corruption exploit (buffer overflow, use-after-free) targets container tooling	Init process and core utilities implemented in memory safe Rust; this vulnerability class is eliminated

LAYER 5

MINIMAL DEPENDENCIES

Defends Against: Bloated Dependency Surface | Merged Library Vulnerabilities | Unnecessary Component Risk

The Attack: The Bloated Image Problem

Layer 2 removes the inherited vulnerability surface of the base operating system. Layer 5 addresses the vulnerability surface of the application's own dependencies. The two are distinct: a zero-CVE base does not help if the application image is then packed with libraries it does not need.

Every dependency present in an image is another library that can carry a vulnerability. The image's total vulnerability surface is the union of the vulnerabilities across every component it contains.
A library included just in case, a utility never called in production, a framework pulled in for one helper function: each merges its entire vulnerability history into the image, expanding the surface an attacker can target and the volume of CVEs a security team must triage, with no corresponding functional benefit.

Conventional container images assembled from general-purpose base distributions include hundreds of components the application never uses. This is not a requirement of containerisation. It is a side effect of assembly rather than compilation. The security team then has to triage CVEs across every bundled component, whether or not the application ever runs the vulnerable code.

CleanStart's Defence: Minimisation as the Default State

CleanStart images include only the components the target workload actually needs.

Because the image is compiled from source with explicit component selection rather than assembled from a general purpose base, minimisation is the default state, not a hardening step bolted on afterward.

The effect is a structural collapse of the aggregate vulnerability surface before any scanning occurs. Fewer libraries mean fewer libraries that can carry CVEs, a smaller exploitable footprint, and less for a security team to track and remediate.

This layer works with Layer 4. Shell-less execution removes the attacker's operating environment, and minimal dependencies reduce the candidate libraries an attacker can probe. The combination makes the container surface, to a large extent, impregnable. There is little left to attack, and what little remains is itself hardened.

ATTACK VECTOR	CLEANSTART DEFENCE
Unused convenience library carries a CVE; expands attack surface for no benefit	Component not present unless required by the workload
Aggregate CVE count across bundled libraries overwhelms security triage	Minimal dependency set collapses the merged vulnerability surface before scanning
Exploit targets a redundant runtime or framework left in the image	Redundant runtimes and frameworks are excluded at build time
Large image surface gives an attacker many candidate libraries to probe	Smaller footprint: fewer components, fewer exploitable targets

The Attack: The 95% Blind Spot, and the Licensing Blind Spot Beside It

Endor Labs research found that 95% of vulnerabilities in modern software sit in transitive (indirect) dependencies. These are packages developers never explicitly chose. They arrived as dependencies of dependencies, sometimes several layers deep in the graph.

A developer installs a logging library. That library depends on a serialisation utility. That utility depends on a compression library from 2014 that is no longer maintained. A CVE is disclosed in the compression library. The developer has no knowledge that this package is even present in their application.

As of 2025, 13% of Log4j downloads were still pulling vulnerable versions. This was not negligence. Log4j was embedded as a transitive dependency inside other libraries inside their applications, invisible to standard dependency manifests. The original Log4Shell discovery triggered a frantic global effort precisely because the blast radius was unmappable without a complete dependency tree.

Dependency confusion attacks exploit this complexity on purpose. An attacker finds internal package names by scraping job postings, GitHub repositories, Docker layer metadata, or error messages in public issue trackers, then registers a public package with an identical name. When a build system pulls from both public and private registries without explicit scope enforcement, it may download the malicious public package instead of the intended internal one. The build succeeds. The malware is signed and deployed.

The unknown transitive dependency is not only a security risk. That same 2014 compression library, pulled in five layers deep and never explicitly chosen, may carry a copyleft or otherwise restrictive licence. A GPL, AGPL, or unknown-licence obligation can propagate legal requirements the organisation never knowingly accepted.

Without a complete inventory, an enterprise cannot know what licences it is shipping, cannot satisfy its own procurement and compliance obligations, and cannot bound its legal exposure. The blind spot cuts two ways: unmeasured vulnerability and unmeasured liability.

THE SCALE OF THE HIDDEN EXPOSURE

In 2025, attackers published 454,648 malicious packages in npm alone. Over 1.2 million total malicious packages have been identified across open source registries.

95% of vulnerabilities reside in transitive dependencies, not the packages developers explicitly install. The total CVE and licence surface of a running container is orders of magnitude larger than what developers track.

Without a complete SBOM covering every transitive dependency, the vulnerability and licensing surface is unmeasurable.

The EU Cyber Resilience Act will require complete dependency chain SBOMs, covering all transitive dependencies, by December 2027, with fines up to €15 million for non-compliance.

CleanStart's Defence: Full Dependency Tree Analysis, Licence Inventory, and SBOM

CleanStart's build process does not stop at the direct dependency list. Every image build performs a complete dependency tree resolution. It traces every package to its root components, identifies every transitive library, and validates every component against known vulnerability databases and malware signatures before it is included in the build.

This process is enabled by the hermetically sealed source-build model. Because CleanStart compiles from source rather than pulling pre-built binaries from registries, it has full visibility into exactly what components are included at each layer of the dependency tree. A pre-built binary may contain components that are not declared in its manifest. Source compilation is explicit by definition.
Every CleanStart image ships with a complete SBOM: a cryptographically signed inventory of every component in the image. It covers all transitive dependencies, their versions, their known vulnerabilities at publication time, their licences, and their provenance chain.

When a new CVE is disclosed in any component at any depth of the dependency tree, CleanStart's monitoring systems identify every affected image variant across the entire catalog within hours. The 7-day Critical / 14-day High-Medium-Low SLA applies to transitive dependencies with the same commitment as direct dependencies.

Because every transitive component's licence is identified and inventoried, an enterprise also knows exactly what it is shipping: there is no hidden copyleft obligation buried five layers deep, no unknown-licence component creating unbounded liability, and no gap between what procurement signed off on and what is actually in production.

ATTACK VECTOR	CLEANSTART DEFENCE
CVE in transitive dependency five layers deep; undetected by standard scanning	Full dependency tree analysis at build time; every transitive package evaluated
Dependency confusion: malicious public package shadows internal private package	Hermetic build environment: no public registry access during compilation
Unmaintained library (2014 vintage) carries unpatched vulnerability	Full tree analysis identifies unmaintained packages; alternatives sourced or patched
Log4j-style hidden blast radius: scope of vulnerability unmappable	Complete SBOM with full transitive graph: every component known and trackable
Restrictive or unknown licence buried in a transitive dependency	Full licence inventory across the transitive tree; no unaccounted-for obligation or liability
Malicious package injected at specific dependency depth to avoid scanning	Source-build model: no pre-built binary inputs; injection at depth is impossible

Why Six Layers, Not One

Each CleanStart layer is independently valuable. Together, they achieve something qualitatively different: comprehensive structural defence against the full software supply chain attack taxonomy, and the licensing liability dimension that pure-security tooling ignores.

An adversary who defeats one layer encounters another. A zero-day that bypasses the CVE scanning finds no shell to execute in. A malicious transitive dependency that evades source verification finds a read-only filesystem that prevents its payload from deploying. A misconfiguration that might enable privilege escalation is eliminated by hardening before the container reaches production.

THE XZ UTILS ATTACK AGAINST A CLEANSTART-PROTECTED WORKLOAD

- Attacker plants backdoor in XZ Utils source over two years. (Attack succeeds at the open source level.)
- Layer 1 intercepts. CleanStart's maintainer-identity and intent verification evaluates the anomalous contribution and the maintainer-behaviour change. The source is not trusted simply because it compiles
- Layer 3 intercepts. If Layer 1 were bypassed, the backdoor attempts SSH remote code execution. The hermetically built image contains no SSH daemon
- Layer 4 intercepts. If the exploit fires against a different vector, it attempts to spawn a shell. No shell exists.
- Layer 4 intercepts again. If execution were somehow achieved, the attacker attempts to write a persistence mechanism. The filesystem is read-only.
- Layer 6 provides independent detection. The transitive dependency carrying XZ Utils is flagged in full tree analysis regardless, and its licence and provenance are inventoried.

At no point does a single failure cascade into a breach. The layers are independent, not sequential gates.

The Full Attack Taxonomy, Mapped to the Stack

ATTACK CLASS	REAL-WORLD EXAMPLE	CLEANSTART LAYER	HOW IT FAILS AGAINST CLEANSTART
Malicious code injection	XZ Utils backdoor 2024; Axios npm 2026	Layer 1: Verified Images	Maintainer-identity + intent verification; hermetic environment blocks build-time injection
Known CVE exploitation	Log4Shell; unpatched base layer libraries	Layer 2: Hardened Images	Zero known CVEs at publication; 7-day Critical SLA closes window before exploitation
Misconfiguration attack	Container as root; writable /tmp	Layer 3: Hardening	Non-root, minimal capabilities, structural hardening; CIS as auditable baseline
Zero-day RCE	AI-discovered zerodays (Mythos Preview)	Layer 4: Shell-less & Read-only	Exploit fires but finds no shell, no tools, no writable paths; chain terminates
Bloated / merged vuln surface	Unused libraries inflating CVE count	Layer 5: Minimal Dependencies	Only required components present; aggregate vulnerability surface collapsed
Transitive dependency CVE	13% Log4j downloads still vulnerable 2025	Layer 6: Full Visibility	Full tree analysis at build; every component at every depth evaluated
Dependency confusion	Internal package shadowed by malicious one	Layer 6: Full Visibility	Hermetic build environment; no public registry access during compilation
Hidden licensing liability	Restrictive licence buried in transitive dep	Layer 6: Full Visibility	Complete licence inventory across the transitive tree

CONCLUSION

From Defence-in-Depth to Structure-as-Defence

The supply chain threat landscape of 2026 has one defining characteristic: attacks arrive through trusted channels faster than defensive tooling can identify them.

Malicious packages are signed and distributed by authenticated accounts. Zero-days are discovered and weaponised before patches exist. Transitive vulnerabilities, and transitive licensing obligations, live undetected for years in libraries no developer explicitly chose. Waiting to scan, detect, and remediate is a strategy designed for an attack timeline that no longer exists.

CleanStart's six-layer architecture is not a detection system. It is a structural transformation of the container environment that removes the conditions attacks require to succeed.

Verified images
Remove the malicious code, and the unverified people behind it, before it reaches the build.

Hardened images
Eliminate the known vulnerability surface.

Hardening
Eliminates the misconfiguration and privilege-escalation gaps.

Shell-less and read-only execution
Removes the exploit's operating environment.

Minimal dependencies
Collapse the application's attack surface.

Full dependency visibility
Eliminates both the hidden vulnerability blind spot and the hidden liability beside it.

The cumulative effect is not merely better security. It is a fundamentally different security posture, one where the outcome of a successful attack at the application layer is contained, logged, and unable to propagate, rather than the starting point of a breach that will go undetected for nine months.

This is the essence of verified trust, and why it is the foundation of a Zero Trust posture that reaches beyond network and identity to the software artifacts themselves.. Nothing runs because it was signed, named, or widely downloaded. It runs because its identity, intent, and integrity have been verified.

End-to-End. Layer by Layer.

Other vendors find the problem. CleanStart eliminates the environment the problem needs to operate in. Six independent layers. One comprehensive defence.

"They find the problem. We eliminate it."

REFERENCES

- Sonatype. '2026 State of the Software Supply Chain.' sonatype.com, 2026. (Malicious packages; Lazarus Group; self-replicating npm malware; 454,600 new packages in 2025.)
- Cybersecurity Ventures. 'Global Costs of Software Supply Chain Attacks.' December 2025. ($60B 2025 estimate; $138B 2031 projection.)
- Prosegur / Cipher. 'Supply Chain Attacks: 2025 Analysis and 2026 Trends.' February 2026. (Attacks doubled; 877,522 malicious packages detected.)
- Endor Labs. '2025 Dependency Research.' (95% of vulnerabilities in transitive dependencies.)
- Microsoft Security Blog. 'Supply Chain Attacks.' April 2026. (Axios / Sapphire Sleet March 2026; Shai-Hulud 2.0.)
- Netlas. 'Supply Chain Attack — How Attackers Weaponize Software Supply Chains.' December 2025. (npm chalk/debug compromise; 18 packages; 2.6B weekly downloads.)
- Cato Networks. 'CVE-2024-3094 and XZ Upstream Supply Chain Attack.' (XZ Utils backdoor; SSH RCE; CVSS 10.0; 2-year social engineering campaign.)
- Mandiant. M-Trends 2026. (Mean time to exploit: negative 7 days.)
- UK AI Security Institute (AISI). 'Our evaluation of Claude Mythos Preview's cyber capabilities.' April 13, 2026.
- Anthropic. 'Project Glasswing.' anthropic.com/glasswing, April 2026.
- Gecko Security. 'Transitive Dependencies Explained.' April 2026. (13% of Log4j downloads in 2025 using vulnerable versions.)
- a16z. 'Et Tu, Agent? Did You Install the Backdoor?' April 2026. (267-day breach detection average; Socket detects Axios in 6 minutes.)
- Appsecsanta / Stingrai Research. 'Supply Chain Attack Statistics 2026.' (1.2M+ malicious packages identified; 454,600+ new in 2025.)
- NetRise. '2024 Container Study.' (Public container images carry 50–600+ CVEs before application code added.)
- CIS. 'CIS Docker and Kubernetes Benchmark.' (Hardening controls; misconfiguration as primary container breach vector.)
- EU Cyber Resilience Act. Requirement for complete dependency chain SBOMs by December 2027; fines up to €15M for non-compliance.