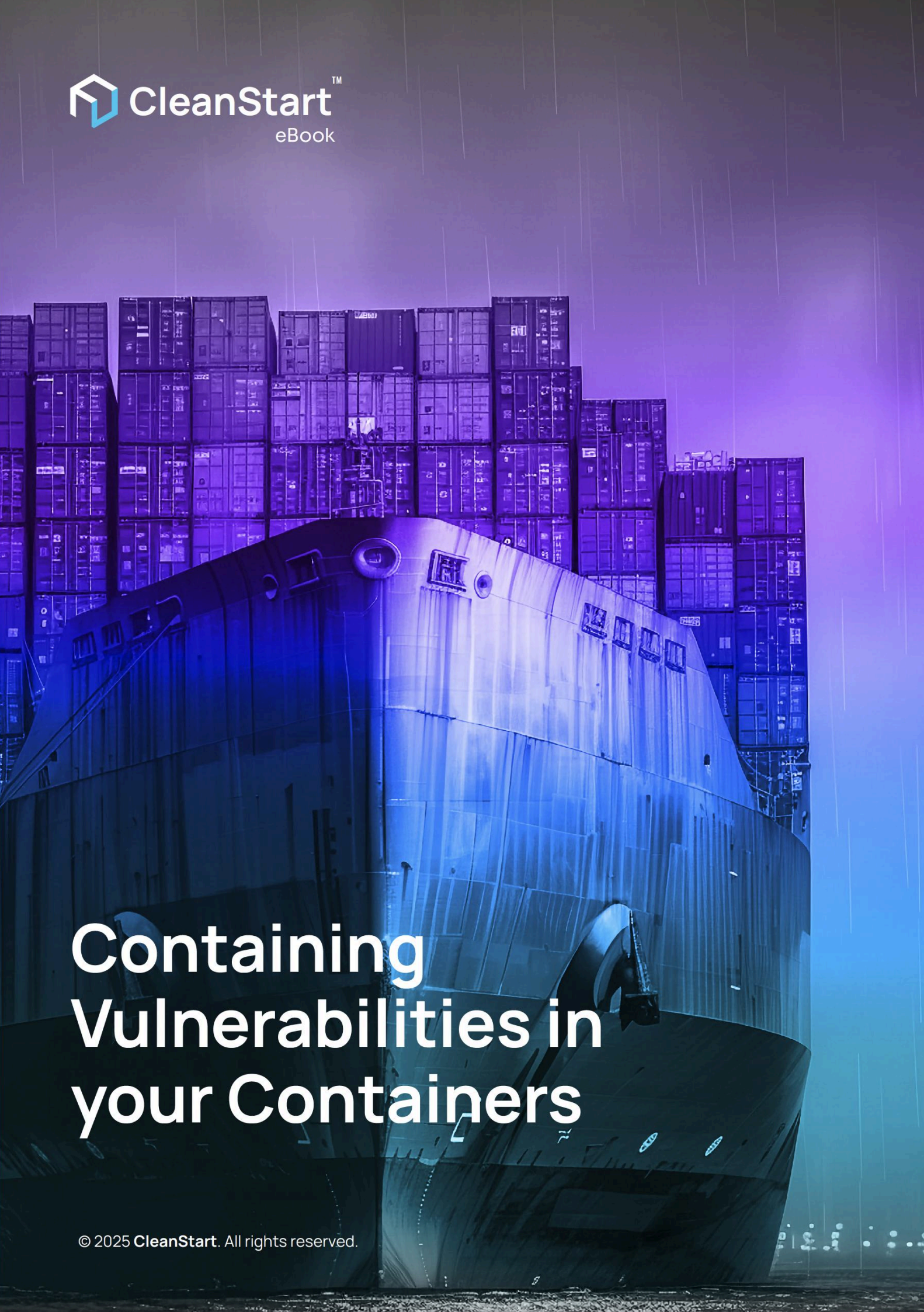
CleanStart™
eBook



Containing Vulnerabilities in your Containers

They likely never worked with container images. Or had to inspect a Kubernetes YAML file at 3 AM. In today's complex environments, knowing what's inside your containers is harder than it should be.

And that's the problem.

Containerization brought immutable infrastructure, predictable environments, and scale. It simplified how software is built and shipped. But it also changed something fundamental: What we ship is no longer obvious. Risk is no longer isolated. It is packaged and replicated across environments.

Along the way, a few assumptions took hold:

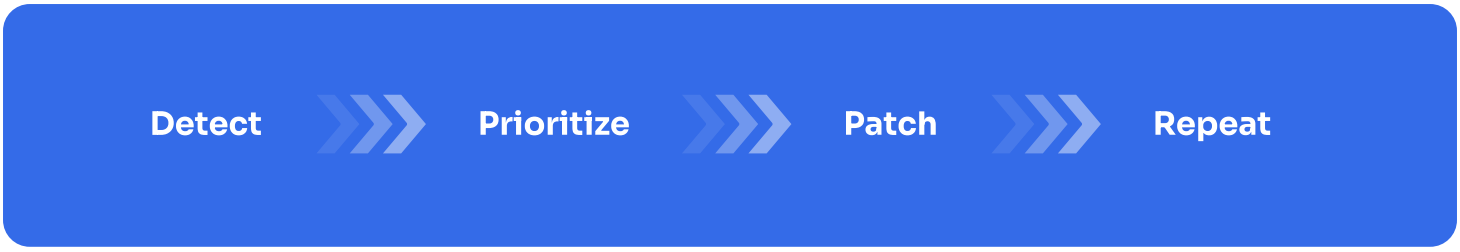
- If it's isolated, it must be secure
- If it's lightweight, it must be low risk
- If it runs in a container, it must be controlled

None of this is inherently true.

Most security efforts focus on:

- Scanning images
- Monitoring runtime
- Responding to vulnerabilities

But all of these happen after the container already exists. By then, the dependencies are included. The attack surface is defined. The risk is already shipped.



At scale, that cycle breaks. The real gap is not visibility. It is control over what goes into containers in the first place.

This guide outlines 7 common container security risks. But more importantly, it highlights a pattern:

Most teams try to secure containers after they are built. The real opportunity lies in how they are built.

#1 Security Risk
Lack of image visibility

Simplicity is good, but also dangerous. As container adoption grows, so does the attack surface. Most incidents still start with something simple: an image pulled from a public registry Public registries are a flea market. You might find something useful. You might also inherit vulnerabilities you didn't ask for. Security teams often discover this late. Not in pipelines, but through incidents or unexpected behavior.

So teams turn to scanning and this is where a false sense of control sets in.

A container is like Schrödinger's cat. You don't know what's inside until you scan it. Scanning tells you what is inside an image. It does not control how it got there. By the time a scan runs, the image is already built and the attack surface already defined.

Modern tools generate SBOMs and flag CVEs. But they all rely on the same assumption: That visibility after the fact is enough.

It isn't. The result is a pattern most teams know too well

- Large volumes of vulnerabilities
- Limited prioritization
- Little reduction in actual risk

SBOMs improve transparency, but they don't answer a more important question: Why does this component exist at all?

Real visibility starts earlier, at build time. The question shifts from "what's inside this image?" to "what should be inside this image?"

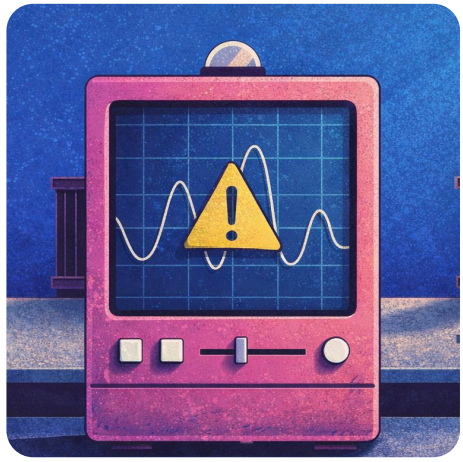
This means explicit control over dependencies, minimal and purpose-built images, and the elimination of unnecessary components before they ever reach a registry.

The most effective teams don't scan to understand containers. They design them so there is little to understand.

US Executive Order 14028 signed by President Biden on May 21, 2021 emphasized, among other things, enhancing software supply chain security and transparency, thus making the SBOM generation a bare minimum for any container security tool.

#2 Security Risk
Letting Vulnerabilities Foster

Different people have different hobbies. Some read books, some collect stamps, and some discover vulnerabilities in production. Because that's where many teams still find them.



Production incidents are expensive and avoidable. Yet one pattern repeats: "We'll patch it in production." Developers ship fast. Security teams catch up. Production becomes the vulnerability scanner. An expensive one.

The Solution: Shift-Left Security

Shifting left means integrating scanning and policy checks into CI/CD to catch issues earlier. It's the right direction, but often misunderstood. Detection moves left. Risk does not.

By the time code reaches CI/CD:

- Dependencies are already chosen
- Base images are already inherited
- Containers are already assembled

You are still evaluating risk after it exists.

This creates a faster loop:

Detect → Fix → Rebuilt → Repeat

But it doesn't reduce the source of the problem.

The real shift happens earlier. Before the image is built:

- Deliberate dependency selection
- Controlled build process
- Exclusion of unnecessary components

Because if risk is introduced at build time, scanning earlier won't remove it.

Recent industry reports indicate that over 50-60% of container-related breaches involved known, patchable CVEs.

#3 Security Risk
Runtime Risks

Runtime security is a challenge. But not as challenging as explaining why your containers were running cryptocurrency miners. Because by then, it's already too late.

A container may start clean, but it doesn't stay that way. Runtime attacks and lateral movement are expected in environments with gaps. Attackers don't need zero-days, they need visibility gaps and time. And they usually get both.

Runtime alerts exist. But in practice, they are ignored, filtered, or buried in noise until something forces attention:

- A billing spike
- Suspicious traffic
- A breach

The Solution: Continuous Runtime Security Monitoring

Runtime monitoring provides visibility into behavior and anomalies in real time. But it operates under a constraint:

It detects what is already happening.

It does not prevent what should never have been possible.

If a container includes unnecessary tools or excessive privileges, monitoring can alert you, but it cannot remove them. More components mean more behavior, more noise, and a higher chance of missed signals.

The most effective way to reduce runtime risk is not just monitoring. It is reducing what the container is capable of doing in the first place.

A 2023 Devo SOC Performance Report found that 83% of SOC analysts admitted to ignoring alerts they believed were false positives.

#4 Security Risk
Image vulnerability proliferation

Same vulnerabilities, different containers. At scale, this becomes systemic risk.

Container sprawl replicates everything, including vulnerabilities. If a base image is vulnerable, every derived container inherits it. Across teams and environments, one weak foundation becomes widespread exposure. At scale, predictability works against you.

The Practiced Solution: Golden Image

Golden images standardize base images that are:

- Pre-approved
- Hardened
- Regularly patched

They improve consistency and reduce repeated exposure. But they have limits.

Golden images are still images. They can include unnecessary packages and inherited dependencies, and over time, they accumulate more than they eliminate.

Standardization reduces variation but it does not guarantee minimalism. Vulnerabilities persist, just more consistently.

The real goal is not just standardization. It is minimization. Every extra component is potential attack surface

> You can't secure and ship what's already vulnerable

A 2023 Devo SOC Performance Report found that 83% of SOC analysts admitted to ignoring alerts they believed were false positives.

#5 Security Risk
Hardcoded credentials

Secrets in container images are not secrets. They are exposure.

API keys, tokens, and passwords are still commonly found in images. Not due to sophisticated attacks, but because they are easy to discover. It's not a breach It's a discovery.

The issue is not awareness. It is how containers are built. When secrets are introduced at build time, they become part of the image:

The issue is not awareness. It is how containers are built. When secrets are introduced at build time, they become part of the image:

- Replicated across environments
- Stored in registries
- Hard to fully remove

The problem is not where they are used. It is where they exist

The Solution: Secrets Management Integration

The best practices are clear:

- Inject at runtime
- Rotate frequently
- Scan for exposure

> You can't secure what's in plain sight

Modern systems manage secrets dynamically, keeping them out of image layers. This is necessary, but it still assumes secrets may enter the system.

The better approach is to ensure they never do. Because once a secret is baked into an image, removal is far harder than prevention.

In 2024, researchers at Aqua Security found over 21,000 exposed secrets in publicly available containers - including credentials for cloud root accounts, CI pipelines, and internal databases.

#6 Security Risk
Speed versus Security

Adoption is one of security's biggest challenges, not because tools don't exist, but because they get in the way.

Every new security tool brings its own friction: new configurations, new alerts, and new workflows.
Multiply that across scanners, policy engines, runtime monitors, and compliance validators, and the overhead compounds quickly.

Each tool may solve a specific problem, but together they create a new one:

- More context switching between tools and dashboards
- More false positives competing for attention
- Optimizing your website for search engines with effective SEO strategies
- More steps between writing code and shipping it

When security slows delivery, developers work around controls. And trust erodes

The Solution: Integrated, Native Security

Security should not be bolted onto the workflow. It should be woven into it.

- No new dashboards to learn
- No new alerts to triage
- No additional scans delaying deployments

Security decisions are made once at the foundation level, and every container built on that foundation inherits them automatically.

But even native security has limits if the underlying artifacts are complex.

Resentment grows when it takes 45 minutes to scan a single image - with your developers waiting to deploy.

#7 Security Risk
Lack of Automated Compliance

Compliance today is continuous scrutiny. The challenge is not just meeting requirements. It is proving them consistently.

Most frameworks weren't designed for containers. Teams are left interpreting controls and defining evidence.

The result is manual effort:

- Spreadsheets
- Screenshots
- Last-minute audits

Pipelines slow. Deployments stall.

The Solution: Built-in Compliance

Compliance must be enforced, not validated later.

- Policies in pipelines
- Non-compliant artifacts blocked
- Evidence generated automatically

If it cannot be validated programmatically, it won't scale.

> You can't comply if you're not secure

Automation helps but complexity remains a problem. If systems are complex, proving compliance is complex.

The best way to simplify compliance is not just automation. It is simplifying what is being checked. Minimal, predictable containers make compliance a byproduct, not a burden.

As former US Deputy Attorney General Paul McNulty once helpfully explained: "If you think compliance is expensive, try non-compliance."

#8 Rethinking Container Security

Across visibility gaps, delayed fixes, runtime threats, and compliance pressure, one pattern keeps repeating: The problem is rarely detection. It is timing.

Most container security strategies operate after the container already exists. Scan it, monitor it, patch it, prove it. By then, the dependencies are already included, the attack surface is already defined, and the risk is already in motion.

This is why the same issues keep resurfacing:

- The same vulnerabilities across multiple images
- The same alerts buried under noise
- The same compliance evidence reconstructed after the fact

Different symptoms. Same underlying problem.

Security is not a scanning problem. It is a composition problem.

Containers are not just deployed. They are assembled. Every component in a container is a decision and every unnecessary inclusion expands the attack surface that needs to be managed later. Most teams focus on managing what's already there. Fewer focus on controlling what gets there in the first place.

A different approach

This is where a new approach to container security is emerging.

Instead of:	The focus shifts to:
Detecting vulnerabilities after build	Controlling dependencies at build time
Monitoring behavior at runtime	Minimizing what goes into each container
Managing alerts and exceptions	Ensuring only what is necessary is included

➜ Rethinking Container Security

When containers are minimal, intentional, and verifiable, there is less to scan, less to monitor, and less to explain during audits. Security improves not because more tools are added, but because less risk is introduced.

Putting this into practice

This is what that approach looks like in practice.

Instead of layering security on top, CleanStart focuses on how containers are constructed in the first place.

By controlling what goes into an image:

- Unnecessary components are eliminated
- Vulnerabilities are reduced at the source
- The resulting containers are minimal, predictable, and easier to secure

This shifts security from reactive to foundational.

The shift ahead

As container environments continue to scale, the limits of reactive security are becoming more visible

The next phase of container security will not be defined by better detection.

It will be defined by better control.