WHITEPAPER

# Reducing Day-Zero Threat Exposure on the Edge

Why a smaller trusted base matters more than runtime hardening alone when devices are fielded beyond direct operator control

April 2026

**nova8 Technologies**

nova8.io

# Table of Contents


1. Executive Summary
2. The Edge Security Mistake: Inheriting Server OS Assumptions
3. What a Smaller Trusted Base Actually Means
4. Build-Time vs. Runtime Hardening: Why Removal Beats Disablement
5. Physical Access and USB/Keyboard Threats on Unattended Hardware
6. Kernel Attack Surface Reduction Through Minimal Userspace
7. Immutable Hosts and the Elimination of Persistence Vectors
8. Design and Buying Implications for Edge Platform Teams
9. How nova8 Technologies Aligns with These Principles
10. References

# 1. Executive Summary

Most edge security failures begin with an inherited assumption: that the host operating system should resemble a general-purpose server. Server distributions are designed for environments with interactive administration, persistent network connectivity, and human operators who install, configure, and maintain software over time. At the edge, those assumptions invert. Devices are unattended, physically exposed, and updated remotely if at all.

This whitepaper argues that the most effective approach to edge security is day-zero threat reduction: eliminating unnecessary binaries, mutable paths, and administrative surfaces at build time rather than layering runtime hardening onto a bloated base. A component that was never included in the image cannot be exploited, misconfigured, or used as a pivot point. This structural property is stronger than any runtime control.

The paper examines what a smaller trusted base actually means (beyond image size), why build-time removal provides stronger assurance than runtime disablement, how physical access threats on unattended hardware change the design calculus, and how immutable host architectures eliminate entire classes of persistence and tampering risk. It draws on publicly available guidance from NIST SP 800-123, NIST SP 800-190, DISA STIGs, CIS Benchmarks, and the Linux kernel security documentation. It is intended for security teams, embedded platform owners, and technical buyers evaluating edge platforms for defense, critical infrastructure, and industrial deployments.

## 2. The Edge Security Mistake: Inheriting Server OS Assumptions

The most common security architecture mistake on edge platforms is starting from a general-purpose server operating system and attempting to harden it for unattended deployment. Server distributions are designed for interactive administration: they ship package managers, interactive shells, broad hardware driver sets, debug utilities, documentation, and development libraries. These components exist because a server is expected to have a human operator with console or SSH access who will install, configure, and maintain software over time.

At the edge, those assumptions invert. Devices are deployed into environments where physical access is uncontrolled, network connectivity is intermittent, and human operator presence is rare or impossible. Every binary, service, and writable path that exists on the host but does not participate in the mission is an unnecessary expansion of the attack surface: a potential pivot point for an attacker, a source of CVEs that must be tracked and patched, and a configuration surface that can drift from the intended posture.

The hardening guides published by DISA (STIGs), CIS Benchmarks, and NIST SP 800-123 all begin from the premise of reducing a general-purpose OS to a task-specific configuration. But when the starting point is a full server distribution, hardening becomes a subtractive exercise performed after build time: disabling services, removing packages, applying restrictive policies. Every omission or regression in that process is a latent vulnerability.

A typical server Linux distribution ships with 1,500 to 3,000 packages, hundreds of executable binaries, multiple shell interpreters, a package manager with network access, debug and profiling tools, development headers and libraries, broad hardware driver coverage, and a full init system with dozens of enabled service units. For an edge device whose mission is to run containerized workloads and communicate with a management plane, the vast majority of these components are unnecessary.

## 3. What a Smaller Trusted Base Actually Means

A smaller trusted base is not simply a smaller disk image. Image size is a useful proxy, but the security-relevant metric is the number of executable binaries, writable filesystem paths, running services, available syscall targets, and local privilege escalation vectors present on the production host.

A purpose-built edge OS should contain only what is needed to boot, establish device identity, run containerized workloads, and communicate with its management plane. Every binary beyond that minimum is a potential attack surface. Every writable path is a potential persistence location. Every running service is a potential entry point (see Figure 2).

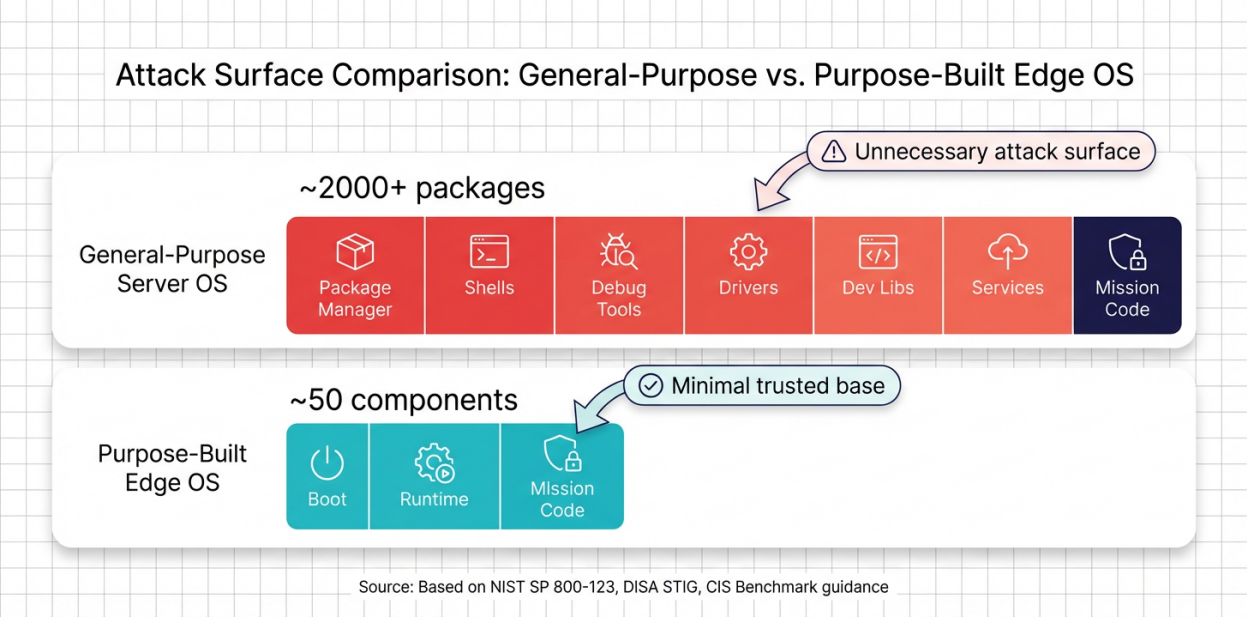


Figure 2: Attack surface comparison between a general-purpose server OS and a purpose-built edge OS.

The Linux kernel's attack surface is itself shaped by the userspace it supports. A host that never runs an X11 server does not need the associated kernel subsystems. A host that does not mount USB mass-storage devices does not need the USB storage driver stack. A host that does not permit interactive logins does not need PAM, login shells, or terminal allocation. Removing these components is not a feature limitation; it is a security property.

- Fewer binaries means fewer CVEs to track, fewer potential SUID/SGID escalation paths, and fewer tools available to an attacker post-compromise.
- Fewer writable paths means fewer locations where an attacker can persist implants, modify configuration, or stage lateral movement tools.

- Fewer running services means fewer listening sockets, fewer IPC surfaces, and fewer components that must be kept current.
- A minimal userspace reduces the effective kernel attack surface by eliminating the drivers and subsystems that unused userspace components would exercise.

# 4. Build-Time vs. Runtime Hardening: Why Removal Beats Disablement

There are two fundamentally different approaches to reducing attack surface: removing components at build time so they are never present in the production image, or disabling components at runtime through configuration, policy, or service management. Both reduce exposure, but they have very different assurance properties.

## Build-Time Removal

Build-time removal is deterministic and verifiable. If a binary is not included in the image build manifest, it cannot be present on any device running that image. There is no configuration to check, no service to verify is disabled, and no risk that a future update or operator action re-enables it. The absence is a structural property of the artifact, not a runtime assertion (see Figure 1).

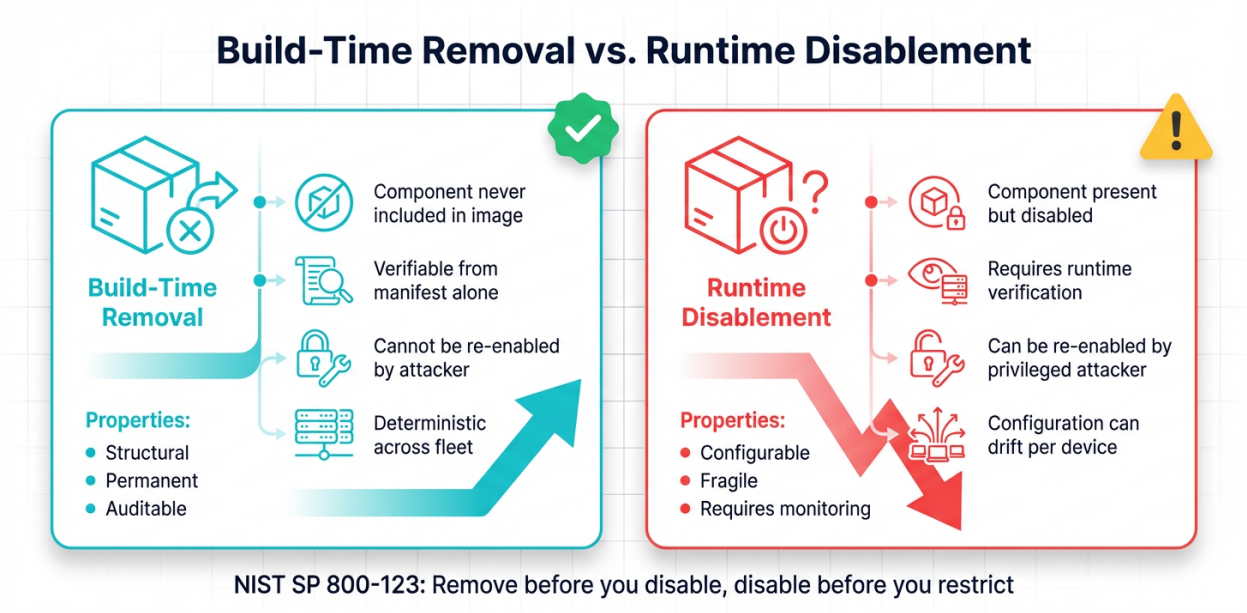


Figure 1: Build-time removal versus runtime disablement approaches to attack surface reduction.

## Runtime Disablement

Runtime disablement is probabilistic and fragile. A service that is disabled via configuration can be re-enabled by a configuration change, a package update, or an operator with sufficient privilege. A binary that is present but not called can still be invoked by an attacker who gains execution. A kernel module that is blacklisted can be force-loaded by root. Every runtime control depends on the assumption that the enforcement mechanism itself is not compromised, an

surface that is harder to maintain on physically exposed, unattended hardware.

The practical recommendation from NIST SP 800-123 (Guide to General Server Security) and the broader hardening literature is clear: remove before you disable, and disable before you restrict. Build-time removal is the strongest form of hardening because it eliminates the component from the trust base entirely.

- Build-time removal is structural and verifiable from the image manifest alone.
- Runtime disablement depends on configuration integrity and can regress under updates or operator error.
- A present-but-disabled binary can still be invoked by an attacker with execution capability.
- Kernel modules that are blacklisted at runtime can be force-loaded by a privileged attacker; modules excluded from the kernel build cannot.

# 5. Physical Access and USB/Keyboard Threats on Unattended Hardware

Edge devices deployed in uncontrolled environments face a threat that data-center servers largely avoid: sustained, unsupervised physical access by potential adversaries. The assumption that physical access equals full compromise is a useful simplification for risk modeling, but it is not inevitable. Design choices can significantly raise the cost and complexity of physical attacks.

## USB-Based Attacks

USB-based attacks are among the most practical physical threats. Malicious USB devices can emulate keyboards (HID injection attacks, such as those demonstrated by USB Rubber Ducky and similar tools), present as network interfaces to redirect traffic, or attempt to exploit USB driver vulnerabilities in the kernel. If the host kernel loads USB HID and mass-storage drivers by default, these attack vectors are available to anyone with momentary physical access.

The countermeasure is architectural: exclude USB HID, USB mass-storage, and other unnecessary USB class drivers from the kernel build entirely. A device that does not include the USB HID driver stack cannot be attacked through keyboard emulation, regardless of what USB device is connected. This is a build-time decision that provides permanent protection without any runtime configuration.

## Boot Media Substitution

Boot media substitution (booting the device from an attacker-controlled USB drive or network source) is another physical threat that must be addressed at the firmware level. UEFI Secure Boot, when properly configured with enrolled keys and disabled fallback boot paths, prevents the device from executing unsigned boot media. Combined with measured boot and TPM-backed attestation, this creates a verifiable chain from power-on to running workloads that is resistant to offline tampering.

The design implication is that physical threat mitigations must be architectural, not administrative. Policies that instruct operators to "disable USB ports" or "lock the enclosure" are useful but insufficient. The platform itself should minimize the kernel driver surface for unused physical interfaces, enforce Secure Boot with properly scoped key enrollment, and ensure that interactive login paths do not exist unless explicitly enabled through a controlled, auditable process.

- USB HID injection attacks require only momentary physical access and bypass software-only access controls.
- Kernel drivers for unused physical interfaces (USB storage, serial, Thunderbolt) should be excluded at build time, not just disabled at runtime.
- UEFI Secure Boot with properly enrolled keys prevents boot media substitution attacks.
- Measured boot with TPM attestation provides remote-verifiable evidence that the device has not been tampered with offline.
- Interactive login paths should not exist on production images unless explicitly enabled through a controlled break-glass process.

# 6. Kernel Attack Surface Reduction Through Minimal Userspace

The Linux kernel's attack surface is not fixed; it is shaped by the userspace components that exercise kernel interfaces. Each kernel subsystem, driver, and module represents code that an attacker might reach through syscalls, device interactions, or network protocols. Reducing the userspace reduces the set of kernel code paths that are reachable in practice.

## Driver and Module Reduction

A general-purpose distribution includes kernel modules for hundreds of hardware devices, filesystem types, network protocols, and subsystems that the edge device will never use. Each loaded module is kernel code running at ring 0 with full system privileges. A vulnerability in any loaded module is a potential kernel compromise.

Modules that are excluded from the kernel build at compile time cannot be loaded at runtime, even by root. This is strictly stronger than module blacklisting, which prevents automatic loading but can be overridden by a privileged user or attacker. For edge devices with fixed hardware configurations, the kernel can be built with only the drivers required for the specific hardware platform, eliminating hundreds of unused driver modules.

## Syscall Surface Reduction

Seccomp (Secure Computing Mode) provides process-level syscall filtering that restricts which kernel system calls a process can invoke. When combined with a minimal userspace that does not require exotic syscalls, seccomp profiles can be tightly scoped to the actual needs of the containerized workloads.

The combination of build-time kernel module reduction and runtime seccomp filtering provides defense in depth at the kernel boundary: unnecessary kernel code is never present, and the remaining kernel interfaces are filtered per process. This layered approach reduces the effective kernel attack surface below what either mechanism achieves alone.

# 7. Immutable Hosts and the Elimination of Persistence Vectors

Traditional Linux hosts provide writable root filesystems, package managers for installing software, and administrative shells for configuration changes. Each of these capabilities is also a persistence vector: an attacker who gains execution on a mutable host can install backdoors, modify configuration, create new user accounts, schedule cron jobs, or replace system binaries.

## Read-Only and RAM-Centric Execution

An immutable host architecture eliminates these persistence vectors by design. When the root filesystem is read-only (or executed from a verified image loaded into RAM), an attacker cannot modify system binaries, install packages, or create persistent backdoor files. Any changes made to in-memory state are lost on reboot, which means the device returns to a known-good state with every restart.

This property transforms the economics of host compromise. On a mutable host, an attacker who achieves execution once can persist indefinitely by modifying the filesystem. On an immutable host, the attacker must re-establish access after every reboot, and the path they used for initial access may not exist in the next boot (if it was an artifact of runtime state rather than the base image).

## Fleet-Wide Vulnerability Management

Immutable, image-based hosts also simplify fleet-wide vulnerability management. When every device in the fleet runs an identical image, vulnerability scanning becomes an image-level concern rather than a per-device audit. A CVE that affects a component in the image is either present in all devices running that image version (and fixed by deploying a new image) or absent from all devices (because the component was never included). There is no device-by-device variation to investigate.

Incident response benefits similarly. When host state is deterministic (defined entirely by the image version), responders can reason about compromise indicators, lateral movement possibilities, and recovery procedures at the image level rather than reconstructing the unique configuration of each affected device.

# 8. Design and Buying Implications for Edge Platform Teams

Teams evaluating edge platforms should ask a specific question: how much of the shipped host actually participates in the mission, and how much exists because it was inherited from a broader Linux distribution? The ratio between mission-relevant components and inherited baggage is a direct indicator of the platform's security posture and operational maintainability.

- Ask for a bill of materials: how many packages, binaries, and running services are present in the production image?
- Compare the image contents to the actual mission requirements; every gap is unnecessary attack surface.
- Evaluate whether the platform's update model is image-based (atomic, verifiable) or package-based (incremental, drift-prone).
- Confirm that physical access mitigations are architectural (build-time driver exclusion, Secure Boot) rather than purely administrative.
- Prefer platforms where compliance evidence is a byproduct of the build and release process, not a post-deployment scanning exercise.
- Verify that the production image contains no debug tools, compilers, or development libraries.
- Assess whether writable host paths are minimized and whether critical filesystem layers are mounted read-only.
- Ask vendors to demonstrate the kernel module inventory on the production image and justify each included module.

The fewer inherited assumptions a platform carries, the easier it is to reason about security posture at fleet scale. When every device runs an identical, minimal, immutable image, vulnerability management becomes an image-level concern rather than a per-device configuration audit. Incident response benefits from deterministic host state. Compliance evidence is produced by the build and release process rather than by runtime scanning tools that may themselves be unavailable at the edge.

## 9. How nova8 Technologies Aligns with These Principles

nova8 Technologies applies publicly documented security principles, including build-time attack surface reduction, immutable host architecture, hardware-rooted boot integrity, and image-based fleet management, in its edge platform. This paper describes the public architectural pattern and evaluation criteria, not product implementation details.

The platform is designed to ship only the components required for the edge mission, minimize writable host paths, enforce boot integrity through UEFI Secure Boot, and deliver updates as atomic, signed image replacements. These properties are consistent with the guidance in NIST SP 800-123, NIST SP 800-190, and the broader hardening literature referenced throughout this paper.

nova8 Technologies is aligning infrastructure and development practices with CMMC Level 2 expectations. The company holds U.S. Provisional Patent Applications 63/897,352, 63/897,609, 63/903,132, 63/903,161, 63/903,164, and 63/903,168 related to edge computing, security architecture, and operational resilience innovations.

For more information, visit nova8.io or contact the team at contact@nova8.io.

# 10. References

1. NIST, "SP 800-123: Guide to General Server Security," July 2008, csrc.nist.gov/pubs/sp/800/123/final

2. NIST, "SP 800-190: Application Container Security Guide," September 2017, csrc.nist.gov/pubs/sp/800/190/final

3. DISA, "Security Technical Implementation Guides (STIGs)," public.cyber.mil/stigs/

4. CIS, "CIS Benchmarks," cisecurity.org/cis-benchmarks

5. Linux Kernel Documentation, "Kernel Self Protection Project," kernsec.org/wiki/index.php/Kernel_Self_Protection_Project

6. UAPI Group, "Unified Kernel Images (UKI) Specification," uapi-group.org/specifications/specs/unified_kernel_image/

7. systemd, "Automatic Boot Assessment," systemd.io/AUTOMATIC_BOOT_ASSESSMENT/

8. Hak5, "USB Rubber Ducky," shop.hak5.org (representative of USB HID injection attack tooling)

9. UEFI Forum, "UEFI Specification," uefi.org/specifications

10. Trusted Computing Group, "TPM 2.0 Library Specification," trustedcomputinggroup.org

11. OCI, "Open Container Initiative Image Format Specification," opencontainers.org

12. NSA, "Commercial National Security Algorithm Suite 2.0," September 2022, media.defense.gov