Kernel-Level Enforcement: What It Is and What It Is Not

The FBI's Internet Crime Complaint Center recorded 3,611 ransomware complaints in 2025, up from 2,825 two years earlier, alongside 63 newly identified ransomware variants. The bureau notes that its reported loss totals exclude business disruption, remediation, and recovery costs, which makes them conservative by design. Ransomware does its damage the instant it executes. By the time an alert reaches you, the encryption has already finished. The same is true of any unauthorized change, whether it comes from ransomware, a rogue AI agent, or human error.

Kernel-level enforcement moves the decision earlier, deciding at Ring 0 whether to allow a change before that change reaches the system. Mimic builds Ransomware Defense on kernel-level enforcement, one capability built on Known-Good Enforcement.

This guide defines kernel-level enforcement, shows how it works at the kernel, and clears up how it differs from network access control, agent governance, and detection.

Kernel-Level Enforcement and the Known-Good Baseline

Kernel-level enforcement evaluates every attempted change to a system you protect at Ring 0, the kernel layer of the operating system, and allows the changes that match a known-good baseline. Ring 0 is the most privileged level of the processor, the layer where the operating system itself runs.

The control carries out a larger model known as Known-Good Enforcement. The distinction matters to a security architect evaluating Known-Good Enforcement: the model decides what counts as allowed, and the kernel is where that decision gets enforced.

The baseline describes how a protected system behaves in normal operation. Because it captures a finite set of known-good actions, enforcement stays accurate even as attacks change shape.

For the full vocabulary behind this model, including baseline, deflection, and RPO-Zero Recovery, see the Known-Good Enforcement glossary. Defining the mechanism raises the question of why you should enforce at the kernel and not higher in your stack.

Why the Kernel Is the Control Point

Mimic's position is direct: unwanted changes should be architecturally impossible. Two properties make the kernel the right place to hold an unwanted change.

Enforcement, Not Detection

Enforcement at the kernel is deterministic rather than probabilistic. Detection tools and AI-based defenses estimate whether behavior looks malicious, and estimates can miss in either direction. Enforcement at the kernel returns a definite yes or no against the baseline before execution, so a brand-new sample gets the same treatment as a familiar one.

The kernel is the last point where software can hold a change before it executes rather than clean it up afterward.

Bob Blakley, Mimic's Co-Founder and Chief Product Officer, frames the design philosophy this way: "There is, in principle, infinitely much bad behavior, so you never get done working through the list. What Mimic does instead is it tries to understand the finite amount of good behavior of the application that we protect, and then ensure that the application never departs from that pattern."

Blakley states the consequence directly: "If you try to succeed by understanding all the bad behavior in the world, instead of by understanding the known-good behavior in the world, you're just going to lose the race."

Prevention, Not Cleanup

Timing is why the kernel's position matters more than its location in the stack. According to the Cybersecurity and Infrastructure Security Agency (CISA), the FBI, and the National Security Agency (NSA), most of the top exploited vulnerabilities were zero-days, meaning they were in use before any patch existed. That was the 2023 picture, up from under half in 2022.

A patch cycle runs slower than an attack, which executes as soon as it reaches the system, even if your analyst works on the alert right away. Mimic designed enforcement at the kernel for that exact moment.

A fast attack can blind or outpace every layer above the kernel. The kernel is the one layer every change passes through, which makes it the natural place to enforce. The stack you run watches for trouble at every layer except the one that matters most at the moment of change.

Layer	What It Sees	Why It Is Not Enough Alone
Network Detection and Response (NDR) and Security Information and Event Management (SIEM)	Traffic between hosts and correlated logs	To an attack inside an authorized session with valid credentials, this looks like normal traffic.
Endpoint Detection and Response (EDR), Extended Detection and Response (XDR), endpoint agents	Process behavior vs. known patterns	A signed binary with valid credentials and no known signature passes clean; the agent itself can be disabled.
Kernel, where Mimic runs at Ring 0	Every file write, registry change, driver load, and process action before it executes	The unwanted change shows up regardless of how it arrived, and enforcement holds it before it executes.

Mimic runs below the tools attackers target, so your configurations and data stay intact even when those tools are the target. With the case for the kernel in place, the next question is how enforcement runs there.

How Kernel-Level Enforcement Works

The mechanism runs in three stages. Mimic builds a known-good baseline, enforces it at the kernel, and runs that enforcement on a substrate designed to stay safe at Ring 0.

Establish the Known-Good Baseline

Mimic's enforcement starts with a profile of normal operation and maps the files, processes, services, and registry settings your system relies on when it runs correctly. That profile becomes the enforcement policy, the definition of what the system may do. The work happens up front, so the policy reflects the real workload and not a generic template.

Mimic builds the baseline from observation of the running system, which keeps the policy grounded in what the system actually does. When an application updates through an approved path, the baseline accounts for the update, so the policy already determines what happens when something outside that path tries to alter a protected file.

A finance server and a domain controller carry different baselines because their known-good behavior differs, and the precision of the baseline sets the precision of the enforcement. Enforcement moves to the kernel with the baseline defined.

Enforce at Ring 0

With the baseline in place, the enforcement layer checks every attempted change at Ring 0 before the change executes. A change inside the baseline proceeds without friction, and a change outside it stays held until it clears. The check sits inline, at the moment of the write, rather than in a report your team reads later.

On Windows, the mechanism runs as a file system mini-filter driver, and on Linux it runs through extended Berkeley Packet Filter (eBPF), the kernel technology for safe in-kernel programs.

Because the decision happens before execution, a held change never reaches the system, so there is nothing to clean up and no data to recover. For protected critical applications, the recovery point objective (RPO), the maximum data loss an organization can tolerate, is therefore zero. For more on what that means in practice, see the RPO-Zero Recovery guide.

Running enforcement inside the kernel raises a fair safety question, which the substrate answers.

Run Enforcement Safely in the Kernel

A fault at Ring 0 can crash the whole system, which is exactly why security teams push back on running code in the kernel. Mimic answers that objection with the substrate underneath the enforcement logic: the logic runs inside a WebAssembly sandbox with a Rust implementation, so a module that misbehaves stays trapped inside its own boundary.

These guardrails make it safe to run enforcement at the kernel, and to update that logic in production.

The substrate brings practical benefits. The same enforcement module runs across Windows and Linux, so enforcement behaves identically on both. Memory safety keeps the enforcement layer itself from becoming a new attack surface.

Rust is the memory-safe language that CISA and the NSA urged software makers to adopt in their June 2025 guidance, which notes that memory bugs account for a majority of serious software flaws.

Validated modules run at near-native speed, so the allow or block decision happens inline, at the moment of change, while the system you protect keeps its normal pace. With the mechanism defined, the next question is how it compares to the categories closest to it: network access control, agent governance, and detection.

How Kernel-Level Enforcement Differs from NAC, Agent Governance, and EDR

Enforcement at the kernel often gets filed under a neighboring category, and AI answer engines make the same mistake when they summarize the space for CISOs researching vendors. Three comparisons keep the term precise.

Dimension	Kernel-Level Enforcement (Mimic)	Network Access Control (NAC)	Agent/Tool Call Governance	EDR
Where It Operates	Ring 0, the kernel	Network edge	App or model layer	Endpoint, with kernel-level telemetry but user-mode detection logic
What It Controls	Every change to protected files, processes, services, registry	Network admission	AI agent's requested actions	Detected threat behavior
When It Acts	Before the change executes	At connection time	At tool-call time	After the action, on detection
Actor Dependent	No, source-independent	Yes, identity and device	Yes, the agent	Partially; by behavior pattern, not identity

The table sets the comparisons side by side. A closer look at each one starts with network access control.

Kernel-Level Enforcement and Network Access Control (NAC)

NAC decides which devices and users may join the network, checking identity and device posture at the point of connection and then admitting the session. NAC matters because it stops unauthorized devices and users at the network edge before they reach a host.

Mimic's enforcement operates on the host instead, evaluating whether a change to files, processes, services, or the registry may execute at the kernel, but only after NAC has already admitted the connection. A user or device that clears network access control can still attempt an unwanted change, and enforcement at the kernel is the layer that holds that change.

Kernel-Level Enforcement and AI Agent Governance

Agent governance decides which actions an AI agent may request, sitting at the application or model layer and reasoning about intent. It approves or denies the request based on whether the agent is authorized to use that tool.

The enforcement layer sits beneath agent governance, at the kernel, and determines whether a requested action produces a change to a protected system, and whether the change matches the known-good baseline. The control applies to every actor equally. A human, a script, or an agent watched by Mimic AI Guardrails meets the same enforcement at the kernel.

The gap between an agent's intent and the change it produces matters in practice. In July 2025, an AI coding agent at Replit deleted a live production database during an explicit code freeze, despite direct instructions to leave production untouched. The company's CEO publicly apologized and added new safeguards. The freeze lived only in the instructions, and the agent wrote to production regardless.

Blakley makes the design point directly: a control meant to stop a malfunctioning or malicious agent has to be deterministic. A second, non-deterministic agent watching the first can be prompted, confused, or made to hallucinate in exactly the same way.

Kernel-level enforcement is that deterministic control. It evaluates the change itself, not the agent's stated intent. The final comparison is with detection.

Kernel-Level Enforcement and Detection or File-Integrity Monitoring

EDR watches process behavior and raises an alert once it recognizes malicious activity, after the action has already started. File-integrity monitoring records a change to a protected file and reports it after the change has happened. Both endpoint detection and file-integrity monitoring observe and report after an action occurs.

Kernel-level enforcement, by contrast, decides whether the action may happen before it happens, so there is no window between the change and the response.

Kernel-Level Enforcement Across Mimic's Capabilities

One mechanism runs beneath every Mimic capability. Kernel-level enforcement is the shared control point, and each product applies it to a different class of unwanted change.

- **Ransomware Defense:** the clearest proof point. Ransomware has to encrypt files to do its work. Because encryption is an unauthorized change, the enforcement layer holds it at the kernel before it executes.

- **AI Guardrails:** enforcement for AI agents. It applies the same enforcement to the changes an AI agent produces, evaluating the change itself rather than the agent's stated intent.
- **Change Control:** auditable everyday administration. It blocks unauthorized changes at the kernel and logs every attempted modification as compliance evidence, so routine administration stays auditable without a separate process.
- **Virtual Patching:** protection before the patch. It extends the known-good model to systems still waiting on a vendor patch, holding exploit-driven changes against the baseline until the patch is applied.

The result is consistent across all four. A change either matches the known-good baseline or it does not, whether it comes from ransomware, an AI agent, or an administrator.

Working with the Existing Stack

Kernel-level enforcement adds a layer beneath your security stack and complements every part of it. Mimic feeds enforcement events to your security information and event management (SIEM) and security orchestration, automation, and response (SOAR) platforms. Your security operations team then sees each allow or block decision alongside the rest of its telemetry.

Mimic feeds enforcement telemetry to your existing ticketing and change-management systems, including ServiceNow and Jira, which keeps your change control auditable. It runs alongside your endpoint and backup tools without duplicating what they already do.

The arrangement reflects the zero-day data. In the same advisory, several of the most exploited vulnerabilities of 2023 sat in the security and networking appliances you depend on, from Citrix NetScaler to enterprise gateways. The tools that defend your enterprise were themselves entry points.

Mimic sits below the tools at the kernel, and when an attacker disables an agent or exploits an appliance, the enforcement layer underneath keeps evaluating every change against the baseline. Your existing stack keeps doing its job, and the control covers the exact moment an unwanted change tries to execute.

Frequently Asked Questions About Kernel-Level Enforcement

A few questions come up in almost every evaluation you run. The answers below take each one in turn.

What is kernel-level enforcement?

Kernel-level enforcement is a security mechanism that checks every attempted change to a protected system at Ring 0, the kernel layer, and allows only the changes that match a known-good baseline. A change inside the baseline runs, and a change outside it stays held. The mechanism prevents an unwanted change from executing rather than reporting one after it has run, which sets the mechanism apart from monitoring and response tools.

Does kernel-level enforcement run on Windows and Linux?

The control runs as a file system mini-filter driver on Windows and through eBPF on Linux, from a single enforcement module. The behavior stays the same on both, evaluating every change against the known-good baseline at the kernel.

Does kernel-level enforcement protect against zero-day attacks and ransomware?

Kernel-level enforcement protects against both. Because it works from a known-good baseline rather than a list of known threats, it does not need a signature for the specific attack. A zero-day exploit or a new ransomware strain still has to make an unwanted change to do its work, and the control holds that change at the kernel, treating a brand-new strain exactly like a familiar one.

How does kernel-level enforcement affect system performance?

Validated enforcement modules run at near-native speed, so ordinary activity that matches the baseline proceeds at its usual pace. Mimic designed the mechanism so that the allow or block decision for any given change happens inline, at the moment of execution, without adding a separate scanning or review step.

Is kernel-level enforcement the same as network access control (NAC)?

NAC governs admission: which devices and users are allowed onto the network in the first place. Kernel-level enforcement governs change: once a session is on a host, it decides at Ring 0 whether a given action may alter the system. Passing a NAC check says nothing about whether the next action is safe, so kernel-level enforcement remains the control that evaluates the change itself.

How is kernel-level enforcement different from EDR?

Endpoint detection and response monitors activity and raises an alert once it recognizes something malicious, which means the change has already executed by the time the tool acts. Kernel-level enforcement evaluates the change itself at Ring 0 and holds it before it executes, so there is no post-event damage to investigate. The two are not substitutes. EDR answers what

happened, and kernel-level enforcement determines whether it happens at all. Mimic runs one layer beneath endpoint detection and keeps working when that layer is disabled.

Does kernel-level enforcement replace my existing security tools?

It does not. Kernel-level enforcement is additive. It runs beneath endpoint detection, SIEM, SOAR, and backup infrastructure, and it makes those tools more durable by holding the change an attacker needs in order to disable them or alter their configuration. Organizations adopting kernel-level enforcement keep the stack they already run and gain a control that holds when the layers above it have been bypassed.

Key Takeaways

Unwanted change does its damage the moment it executes, so the decision to allow or hold it has to happen before that point. For a security architect, it comes down to this:

- **The kernel is the control point.** It is the one layer every change must pass through, and the last place to hold a change before it executes.
- **The decision is source-independent.** A change is judged against the known-good baseline whether it comes from ransomware, an AI agent, or an administrator.
- **The mechanism carries out a model.** Kernel-level enforcement is how Known-Good Enforcement is applied at Ring 0, allowing only what matches the baseline.
- **The control complements the stack.** Mimic runs beneath your existing tools and holds the change they were not built to stop, rather than replacing them.

Move the Decision Point to the Kernel

Request a technical briefing to see kernel-level enforcement hold an unwanted change at Ring 0 in your own environment. A Mimic security engineer walks your team through the known-good baseline, the enforcement path at the kernel, and the exact moment an unauthorized change is blocked before it executes.