MIMIC

INFORMATIONAL GUIDE · ENTERPRISE SECURITY ARCHITECTURE

Known-Good Enforcement

Why an architecture that allows only what is authorized holds when detection cannot keep pace.

A technical and executive reference · Mimic Networks, Inc. · 2026

Table of Contents

Section 01

The Enforcement Model

EXECUTIVE SUMMARY

The Architecture Question

Most security architectures are built to recognize threats. They watch for what is known to be bad, and they act when they see it. Known-good enforcement is built the other way around. It establishes what a system looks like in its authorized state, then allows only the changes that match. **These are not two versions of the same model.** They are different answers to the same problem, and they behave differently the moment a threat is novel. A detection model has to know the attack to stop it. An enforcement model does not, because every attack that lands still requires an unauthorized change, and the change is what gets evaluated.

This guide explains what known-good enforcement is, how it works at the kernel level, and why a threat landscape moving at machine speed makes it the necessary response rather than one option among several. It is written to be evaluated, not believed.

THE LIMIT OF KNOWN-BAD

The Problem With Known-Bad

Detection has been the dominant security model for two decades, and against a large class of threats it works. The limitation it now faces is not a matter of product quality or budget. It is architectural, and it is worth stating precisely.

Detection works by matching activity against known threat patterns: signatures, rules, behavioral indicators an analyst has catalogued as malicious. The model assumes the defender can know enough, fast enough, to describe what attackers are doing before they do it. That assumption held while attacker discovery moved at human pace. It stops holding under three conditions that now define the landscape.

- **AI-assisted discovery outpaces the rulebook.** Enterprise patch cycles run 30 to 45 days. AI-assisted vulnerability discovery produces working exploits in hours. The exposure window is no longer the tail of the cycle. It is the entire cycle, and it opens before an analyst can study the attack and write the rule that would catch it.
- **Authorized identities do unauthorized things.** An AI agent with valid credentials can execute a damaging change through a legitimate process path. There is no prior threat pattern to match, because in isolation the action is not malicious.
- **Novel ransomware runs through approved tools.** A variant that executes through signed binaries and sanctioned utilities leaves no signature to detect. It looks like normal administration until the encryption starts.

WARNING · THE BUILT-IN EXPOSURE WINDOW

A model that has to know the attack in advance carries a built-in exposure window: the gap between when an attack is possible and when a rule exists to catch it. That window was tolerable when discovery was slow.

As attacker discovery velocity accelerates, the window stops being a footnote and becomes the operational problem. You are not defending against what you can describe. You are exposed to everything you cannot describe yet.

DEFINITION

What Known-Good Enforcement Is

Known-good enforcement is a definitional shift, not a feature. Before the mechanism, the definition, written so it stands on its own.

> **Known-good enforcement is a security model that establishes a verified baseline of a system's authorized state, every approved file, process, service, and configuration, then evaluates each attempted change against that baseline at the kernel level before it executes. Authorized changes proceed. Unauthorized changes do not, regardless of the threat behind them.**
>
> DEFINITION · KNOWN-GOOD ENFORCEMENT

The known-good baseline

The baseline is the verified record of a system in its authorized operational state: the files and directories, the processes and services, and the registry keys and settings the system uses in normal, approved operation. It is established by profiling the environment during onboarding, and it is maintained as the environment changes through authorized change. The baseline is not a policy someone writes from assumptions. It is a measured record of how the system actually runs when it is running correctly.

Enforcement at the kernel

Every change to a protected component is evaluated at the kernel before it executes. The question is not whether the change matches a known threat. It is whether the change was authorized against the known-good baseline. Authorized changes proceed without interruption. Unauthorized changes do not execute, whether they came from ransomware, an AI agent, a misconfiguration, or an exploit no one has catalogued yet. The evaluation is indifferent to the source. It resolves on one property of the change: authorized, or not.

Why the kernel layer is the distinction

Enforcement at the kernel, Ring 0, operates below the application layer, below EDR, below any tool that runs in user space. That position matters for a specific reason. Modern attacks target the security tools themselves, disabling agents, altering configurations, tampering with logs. A control that runs alongside those tools in user space can be reached and disabled by the same techniques. A control at the kernel evaluates the change before any user-space process, including the attacker's, can interfere. This is architecturally distinct from application-layer allowlisting, network controls, and detection tools that all operate above the kernel.

DISAMBIGUATION

Two Questions, Three Models

The difference between detection and enforcement is not a difference of degree. The two models ask different questions, and the question determines what each one can still answer when a threat is novel.

The Detection Question	The Enforcement Question
“Does this look like something bad?”	“Was this authorized?”
Requires knowing the attack in advance.	Requires no knowledge of the attack.
Carries an exposure window between when an attack becomes possible and when a rule exists to catch it.	Carries no exposure window, because the evaluation never depended on recognizing the attack.
Runs above the kernel, alongside the tools an attacker targets first.	Runs at the kernel, below where user-space techniques can reach it.

Application allowlisting is the model most often confused with known-good enforcement. The distinction is what each one governs, and where it runs.

Model	What It Evaluates	What It Must Know In Advance
Detection	Activity, after it has occurred, matched against catalogued threat patterns and behavioral indicators.	The attack. A pattern must exist before the activity can be recognized.
Application allowlisting	Which applications are permitted to launch, evaluated in user space at launch time.	The approved application inventory. An allowed application making an unauthorized change is not evaluated again.
Known-good enforcement	Every attempted change to files, registry, services, and configuration, evaluated at the kernel before it executes.	Nothing about the attack. Only what the system looks like when it is authorized.

THE MECHANISM

How Known-Good Enforcement Works

The mechanism, for the architect who will test the claim.

Establishing the baseline

During onboarding, Mimic profiles each protected system in its approved state and records what it captures as the enforcement policy: files and directories, processes and services, registry keys and settings. The kernel is the right position to capture this, because at Ring 0 the attribution is unambiguous: the change, the process behind it, and the object it acts on are all visible at the point of execution, before anything above the kernel can obscure them. As authorized change occurs, the baseline is maintained so that legitimate operations, including approved updates, continue without friction.

Evaluating change in real time

Every attempted change to a protected component is intercepted at the kernel, evaluated against the baseline, and either allowed to proceed or blocked and logged before it executes. This is not a detection-and-response cycle. There is no interval between the change attempt and the enforcement decision in which the change has already happened and the system waits for an alert. The decision is the same kernel call that would have carried out the change. Inside the baseline, the change runs. Outside it, the change does not.

Access-control rules for targeted exposure

For specific vulnerability scenarios, Mimic authors targeted access-control rules that encode the exact behavior an exploit requires and block that pathway at the kernel. The Mimic filter driver intercepts operations, file access, registry modification, process execution, and enforces the rule before the operation reaches the operating system. This holds a compromised process to its known-good scope: lateral movement and privilege escalation outside that scope are not available to the attacker, whether or not the underlying vulnerability has been patched.

Change Intelligence

Individual-action blocking is not enough against an adaptive agent, which simply varies its next action when one is blocked. Change Intelligence is Mimic's continuous, kernel-level awareness of what is changing, in what sequence, at what rate, and against what baseline. It reads the change itself, which is the one signal that does not degrade as an agent adapts. That awareness enables pattern-level enforcement: disrupting the trajectory of an attack before it completes, not just blocking the steps one at a time after they succeed.

Forensic logging

Every blocked action is recorded in real time as it happens: the process, the file write, the registry change, the network call. Mimic's forensic record, referred to internally as DashCam, produces a complete, sequenced, tamper-evident audit trail at the kernel, capturing activity that user-space logging tools would miss because it exists below them. The record is a property of where enforcement runs, not a setting configured after the fact. It continues through maintenance windows, when most logging is relaxed and most unauthorized change hides.

AT A GLANCE

Key Metrics

60sec	Zero	Infinite
Time for modern ransomware to encrypt a full enterprise server. A rogue agent moves just as fast.	Attackers required for an AI agent to delete a production database or take down a code repository.	Ways to misbehave that detection has to anticipate. Mimic enforces one finite known-good model instead.

These three numbers describe the same architectural problem from three directions. The speed of the attack removes the human from the loop. The absence of an attacker removes the threat pattern. The infinity of bad behavior removes the possibility of cataloguing it. A model built on recognizing the attack has no answer to any of the three. A model built on recognizing authorized change answers all three with the same test.

Section 02

Architecture in Practice

THE ARGUMENT

Why Enforcement Addresses What Detection Cannot

The case for known-good enforcement is not that it is a better detector. It is that it asks a different question, and that question keeps having an answer when the detection question runs out of one. Four points make the difference concrete.

Against novel threats

Known-good enforcement does not require knowing the threat in advance. Every successful exploit still ends in the same place: an unauthorized change to a protected component. Enforcement evaluates the change, not the threat category. A zero-day with no CVE, an AI agent acting outside its sanctioned behavior, a ransomware variant no scanner recognizes, all require an unauthorized change, and all are caught by the same enforcement. Novelty is a problem for a model that has to recognize the attack. It is not a problem for a model that only has to recognize that a change is out of scope.

Against AI-speed attacks

Detection responds after observation. Enforcement acts at the moment of change. When frontier AI compresses the time from vulnerability disclosure to working exploit from weeks to hours, the detect-analyze-write-deploy cycle cannot close in time. Enforcement does not run on a cycle. It evaluates at the kernel before the change executes, so the speed of discovery on the attacker's side does not open a window on the defender's side.

Against the privileged-account blind spot

EDR, SIEM, allowlisting, and AI-driven detection all treat activity from privileged accounts using approved tools as legitimate by default. An AI agent with valid credentials making an unauthorized change through an approved tool looks legitimate at every layer above the kernel, because at those layers it is legitimate: the identity is valid, the tool is sanctioned. Known-good enforcement does not evaluate on account privilege or tool approval. It evaluates whether the change was authorized against the baseline. A privileged account making an unauthorized change is still making an unauthorized change.

Against exposure that will never be patched

When a vulnerability lives in code that will not be patched, end-of-life systems, unsupported third-party dependencies, protocol code no vendor maintains, a detection-and-patch model has no closing move. The fix is not coming. Known-good enforcement closes the exposure regardless, because exploitation still requires an unauthorized change, and the change is still evaluated. The vulnerability can remain. Its exploitability does not.

> **The known-good question does not get harder as attacker capability increases, because it never depended on knowing the attack.**
>
> THE ARCHITECTURAL CONSEQUENCE

By this point the conclusion is not a claim to accept. It is where the architecture leads. If every attack that matters ends in an unauthorized change, then evaluating the change, at the layer where it executes, at the moment it executes, is the response that does not degrade as the threats do.

THE PLATFORM

One Architecture, Across the Platform

Mimic's platform is not four products that happen to share a vendor. It is one enforcement architecture applied to four problems. In each, the mechanism is the same: establish the known-good baseline, evaluate change against it at the kernel, allow what is authorized and block what is not. What differs is the class of change each capability governs.

Ransomware Defense

Ransomware is a change problem. Encryption requires modifying files. Persistence requires modifying registry keys and services. Lateral movement requires altering configurations and credentials. Known-good enforcement blocks these changes at the kernel, before the payload can act, without needing to recognize the variant. When containment triggers a recovery event, an event-driven snapshot captures the last good state of protected critical applications, so recovery starts from a baseline that was never compromised rather than a time-based snapshot that may already be corrupted.

AI Guardrails

AI agents operate at machine speed, with valid credentials, inside production systems. Their individual actions look legitimate above the kernel because they are legitimate there. Across the change lifecycle, Mimic binds an agent's declared intent to its approved scope before any change executes, evaluates each action against that scope and the baseline during execution, and assesses system state against intent afterward. Change Intelligence identifies when the sequence, rate, or nature of change diverges from approved scope, and disrupts the trajectory before it completes, rather than blocking one action after another while the agent routes around each block.

Virtual Patching

When AI-assisted discovery surfaces a vulnerability before a vendor patch exists, an exposure window opens that enterprise change management cannot close in time. Mimic authors targeted rules against the specific behavior the exploit requires and cuts off the pathway at the kernel. Protection is immediate on rule deployment and does not depend on the patch arriving.

Four properties make this operationally usable rather than a research capability. Mimic maintains a curated catalog of vetted mitigations published ahead of the vendor patch cycle, so the rule exists before the

enterprise needs it. Each candidate rule is backtested before commit, replayed against a host's own recorded behavior so the blast radius is known per host rather than estimated. Rules roll back in seconds if a host behaves unexpectedly. Audit evidence is generated automatically, per host, per CVE, per timestamp, so the compliance record exists without a separate documentation exercise.

The vulnerability remains until it is patched. Its exploitability is removed in the meantime, including for systems that will never receive a fix.

Change Control

Every change to the production environment is evaluated against the known-good baseline in real time: authorized changes proceed, unauthorized changes are blocked and logged. Integration with existing ITSM platforms such as ServiceNow routes unauthorized-change events into the workflows teams already run. Maintenance Mode relaxes enforcement for authorized administrative work, and Audit Mode re-learns the environment before full enforcement resumes. The forensic record continues unbroken through both, which matters because the maintenance window is the period when most change-control frameworks relax and unauthorized change is easiest to hide.

> **One control plane. One audit trail. One verdict on every change: was this authorized?**
>
> THE PLATFORM STANDARD

The same enforcement decision, expressed as configuration. One observer, one baseline, one scope, evaluated at Ring 0.

```
// Mimic Networks  ·  Ring-Zero Change Intelligence
// kernel.watch.js | v2.4.1

import { KernelObserver } from '@mimic/core';
import { ChangeVector, DriftBaseline } from '@mimic/analysis';
import { EnforcementGuard } from '@mimic/enforce';

const observer = new KernelObserver({
  target: 'production',
  scope: ['files', 'registry', 'processes'],
  depth: 'ring-zero',
  decision: 'pre-execution',
  evaluation: 'inline',
});
```

THE OUTCOME

The Enterprise Value of Time and Confidence

Architecture is only interesting to a CISO if it maps to the outcomes they are accountable for. This one maps to two: the time a security program needs, and the confidence that the posture holds as the landscape moves.

Time

Enforcement buys the enterprise time. Time for patch cycles to complete without leaving systems exposed while they run. Time for AI governance frameworks to mature at the pace of deployment instead of lagging it. Time for the security team to investigate an incident whose blast radius is already contained. The enforcement holds while the team responds, which is the opposite of a model where the team races the attacker to the same finish line.

Confidence

The posture does not weaken as the threat landscape accelerates. A vulnerability discovered today is covered the same way as one no one has found yet. A new ransomware variant is blocked by the same enforcement that blocks every other unauthorized change. The known-good model does not degrade as attacker capability increases, because the question it asks does not depend on knowing the attack. That is a different kind of confidence than a detection program can offer, where every new technique is a gap until a rule closes it.

Operational resilience

The math of a breach is a function of containment speed. A breach contained at the application level in real time is an isolated incident the business absorbs without operational impact. The same breach, allowed to

propagate while detection and network controls catch up, becomes downtime, regulatory exposure, and recovery cost. Enforcement changes which of those two outcomes is the default.

Reference

Questions a Skeptical CISO Would Ask

Direct answers. No deflection.

01 What is known-good enforcement?

A security model that establishes a verified baseline of a system's authorized state, every approved file, process, service, and configuration, then evaluates each attempted change against that baseline at the kernel level before it executes. Authorized changes proceed. Unauthorized changes do not, regardless of the threat behind them.

02 How is this different from application allowlisting?

Allowlisting decides which applications are permitted to run, and it operates in user space. Known-good enforcement operates at the kernel and governs change, not just launch: file writes, registry edits, service and configuration changes, evaluated against the baseline at the moment of execution. An allowed application making an unauthorized change is still stopped, and the enforcement layer sits below the tools an attacker would target to disable it.

03 Does known-good enforcement replace EDR?

No. It is additive. EDR, XDR, and SIEM stay in place and keep doing what they do well against commodity threats with signatures. Enforcement runs beneath them and makes the whole stack more durable by ensuring those tools cannot be disabled, their configurations altered, or their data tampered with while an attack is in progress.

04 What happens when an authorized application is updated?

Approved change is part of the baseline's lifecycle. Legitimate updates proceed without interruption, and the baseline is maintained as the environment evolves through authorized change. Enforcement is not a freeze on the system. It is a boundary around unauthorized change specifically.

05 How is the baseline established, and who manages it?

The baseline is captured by profiling each system in its approved operational state during onboarding, files and directories, processes and services, registry keys and settings, and it becomes the enforcement policy. It is maintained as authorized change occurs. It is a measured record of how the system actually runs, not a policy an administrator has to author by hand from assumptions.

06 Can this stop a zero-day before a patch exists?

Yes, within the terms of what enforcement does. It does not prevent the vulnerability from existing or the exploit from running. It prevents the unauthorized change the exploit needs to achieve impact. The CVE can have no number and no published proof of concept. The enforcement question is unchanged: was this change authorized?

07 What protects the enforcement layer itself?

Two things. Its position, at the kernel, below where user-space techniques can reach it, and the execution substrate. Mimic runs its security logic inside a WebAssembly sandbox, so a module that misbehaves stays trapped inside its own boundary rather than destabilizing the system. That delivers kernel-level reach without the risk profile of a traditional kernel extension, and without requiring restarts. Honest note: how a vendor secures its own agent and update pipeline is a fair and important question, and it deserves a direct, documented answer from Mimic beyond architecture. That material is worth requesting as part of any evaluation.

08 How does this work during maintenance windows?

Maintenance Mode relaxes enforcement for authorized administrative work, and Audit Mode re-learns the environment before full enforcement resumes. The forensic record continues unbroken through both. That matters, because the maintenance window is exactly where most change-control frameworks relax and most unauthorized change hides, and the record is what makes the window auditable after the fact. Planned, authorized maintenance is handled as authorized change against the baseline.

09 What is the operational overhead?

The model is designed to reduce the analyst burden rather than add to it, because unauthorized change is blocked at execution rather than surfaced as an alert to triage. There is no queue of maybe-malicious events waiting on human judgment for the blocked class of change. Baseline maintenance tracks authorized change as the environment evolves.

10 How does it handle AI agents that are supposed to make changes?

It holds them to a declared scope. An agent’s authorized behavior is part of the baseline, and Change Intelligence evaluates the sequence, rate, and nature of its changes against that scope. In-scope change runs normally, with no need to pre-catalogue every legitimate action. Out-of-scope change is stopped, regardless of whether the credential and the tool were valid.

11 Does this work for legacy and end-of-life systems?

Yes, and this is one of its strongest cases. Exploitation of an unpatched legacy system still requires an unauthorized change, so enforcement closes the exposure even though a vendor fix will never ship. For systems where patching is not an option, enforcement is the option.

12 How does known-good enforcement relate to zero trust?

It is the enforcement layer zero trust implies but rarely specifies at the level of system change. Zero trust says never trust, always verify. Identity and access controls verify who can act. Known-good enforcement verifies what can change, at the kernel, independent of whether the actor’s identity and tools are valid. It is the control that answers the change question after identity has answered the actor question.

MIMIC

THE STANDARD

Thank You

If it is not authorized, it does not happen.

SALES@MIMIC.COM

Mimic Networks, Inc. · mimic.com