CODEHUNTER®

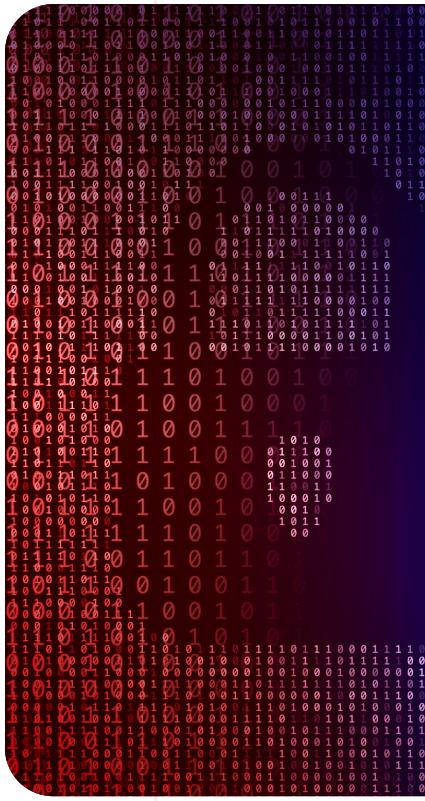
Zero Trust for Code

A New Control Layer for AI-Driven Malware Defense

Zero Trust for Code

A New Control Layer for AI-Driven Malware Defense

Executive Summary



Enterprise malware defense was built to identify threats through signatures, reputation, known indicators, suspicious patterns, or observable runtime behavior.

But AI is changing the rules of the game, enabling attackers to generate, mutate, package, and deliver code at machine speed. Unique payloads no longer match known signatures, files are easily altered without changing their functional purpose, and malicious code exploits trusted software channels before reputation systems, sandboxing tools, endpoint controls, or human review can catch up.

Malware is no longer just a detection or a post-compromise response problem.

Every binary, script, package, container, installer, automation artifact, document macro, browser-delivered payload, and AI-generated code asset represents a trust decision: should this code be allowed to execute inside the enterprise?

Most security stacks cannot answer that question with enough confidence before execution. SBOMs identify what is inside an artifact. Signing and provenance validate where it came from and how it was built. SAST, DAST, sandboxing, EDR, container scanning, and workload identity all solve important pieces of the software risk problem. But they do not fully answer the question that determines the outcome:

WHAT IS THIS CODE CAPABLE OF DOING IF IT RUNS?

CodeHunter is defining a new cybersecurity category called Zero Trust for Code, which extends Zero Trust principles to software execution by treating every artifact as untrusted until its behavioral capability is understood, evaluated against policy, and authorized before it runs.

This new control layer does not simply alert on suspicious code after execution begins, but helps prevent unauthorized code from executing in the first place.

CODEHUNTER
ZERO TRUST FOR CODE

Malware Defense is an Execution Trust Problem

AI-assisted malware breaks three assumptions behind traditional controls.

First, malware is no longer stable. Attackers can use AI to generate or alter payloads in real time. Each instance of code can be unique while still pursuing the same objective: credential theft, persistence, privilege escalation, lateral movement, data destruction, or exfiltration.

Second, malware delivery is no longer limited to conventional channels. Email, collaboration tools, APIs, cloud workloads, package repositories, CI/CD systems, browsers, endpoint installers, and agentic development tools can all become paths for executable code.

Third, execution can happen before humans understand what was introduced. Agentic coding tools and automation systems can install dependencies, modify configurations, fetch scripts, and execute tasks in seconds. Attackers no longer need to deceive a human. They only need to manipulate the environment where autonomous tools operate.

This shifts risk from detection to execution. If malicious code executes before the organization understands its behavior, the enterprise has already accepted the risk. At that point, the security program shifts into response mode: contain the endpoint, investigate logs, scope the blast radius, notify stakeholders, and determine regulatory impact.

That is too late. Enterprises need a control point upstream from execution, not after compromise signals appear.

The Missing Pillar in Software Trust

Software supply chain security has matured around three primary questions.

EXISTING PILLAR	WHAT IT ANSWERS	WHAT IT DOES NOT ANSWER
What's inside?	SBOMs and software composition analysis identify components, dependencies, and known vulnerabilities.	Whether a signed or trusted artifact has been manipulated prior to being signed.
Can we trust its origin?	Signing, certificates, reputation, and publisher verification help validate authenticity.	Whether a signed or trusted artifact behaves safely.
How was it built?	Provenance and build integrity controls help establish whether the build followed expected rules.	Whether the build process resulted in an artifact that behaves in the way that was defined in its original specification.

The Missing Pillar in Software Trust, cont.

These controls remain important, but they do not answer the question that increasingly determines security outcomes: **What can this code do?**

That is the missing fourth pillar.

A signed artifact can still introduce harmful behavior. A trusted vendor update can be compromised. A package with a clean dependency tree can still execute malicious logic. A build can follow an approved process and still produce code that violates enterprise policy. This is where existing controls stop short.

The gap in the enterprise security stack is not visibility — it's pre-execution control.

CODEHUNTER IS NOT:

- SAST, DAST, or IAST
- An SBOM or software composition analysis tool
- A sandbox-only malware detonation product
- A replacement for EDR, SIEM, SOAR, vulnerability management, or application security testing
  It is a pre-execution behavioral analysis and enforcement platform that determines what an artifact is capable of doing, evaluates that behavior against enterprise policy, and enforces a decision before the code is allowed to run.

Zero Trust for Code

The core idea behind Zero Trust, 'never trust, always verify', has changed how enterprises think about identity, devices, networks, and access.

CodeHunter applies that principle to software artifacts.

No code should be trusted based on origin, signature, reputation, build process, or prior observation alone. Trust must be earned through behavioral verification before execution.

In this model, every artifact is untrusted by default. Behavioral capability is analyzed before execution, policy determines whether execution is authorized, and decisions are enforced deterministically.

This is not a replacement for Zero Trust identity. It is the missing execution layer.

EXISTING CONTROL PLANE	CORE QUESTION
Identity	Who is requesting access?
Device security	Is the device trustworthy?
Network controls	What traffic is permitted?
Data security	What information can be accessed?
Zero Trust for Code	**What software is allowed to execute, and under what conditions?**

Zero Trust for Code, cont.

That question matters because software exercises privilege. Code can access files, spawn processes, open network connections, modify configurations, invoke APIs, persist across reboots, load other code, harvest credentials, or move laterally. Once it runs, it acts with the permissions available to the user, process, workload, service account, or automation context that invoked it.

With AI-generated and AI-modified code making origin-based trust unreliable, behavior becomes the decisive control point.

ZERO TRUST FOR CODE PRINCIPLES

Zero Trust for Code is built on seven practical principles:

- Never Trust Code by Origin
- Verify Before Execution
- Catalog Impactful Behaviors
- Constrain Capabilities to Declared Behaviors
- Continuously Evaluate Behaviors
- Automate Decisions
- Assume Compromise

How CodeHunter Works

CodeHunter operationalizes Zero Trust for Code through four core capabilities: behavioral intent analysis, behavior-scoped policy, deterministic verdicts, and local enforcement.

1 BEHAVIORAL INTENT ANALYSIS

Traditional malware detection often asks whether a file resembles something known to be malicious. CodeHunter asks a more durable question: What is this artifact capable of doing?

The platform analyzes inbound, internal, and third-party executable artifacts. And uses static control-flow and data-flow analysis with parallel dynamic observation to build a behavioral profile.

That profile models execution paths and detects system interactions, privilege use, persistence mechanisms, and other behavioral indicators that reveal purpose rather than appearance.

A file can be obfuscated, modified, repackaged, or newly generated. But if it attempts to establish persistence, escalate privileges, manipulate credentials, alter security settings, call out to unexpected infrastructure, delete data, or move laterally, those capabilities still matter.

CodeHunter evaluates the artifact's functional behavior, not just its surface characteristics.

2 BEHAVIOR-SCOPED POLICY

CodeHunter's behavioral analysis is not limited to labeling something "good" or "bad." It evaluates behavior in context.

Behavior lists can be mapped to recognized behavioral frameworks such as MITRE ATT&CK and MITRE Malware Behavior Catalog. This gives security teams a common language for understanding what the artifact can do and why it matters.

The platform can distinguish between behaviors that are clearly malicious and behaviors that are dangerous depending on context. Credential access may be expected in a password manager but unacceptable in a package dependency. Data deletion may be ordinary in a disk management utility but catastrophic in an office macro or AI-generated helper script.

This is where policy becomes critical. The question is not simply, "Is this behavior known to be malicious?" The question is, "Is this behavior authorized for this artifact, in this environment, under this policy?"

3 DETERMINISTIC VERDICTS

Many security tools generate scores, alerts, confidence levels, or advisory findings. That leaves analysts to decide whether an artifact is dangerous enough to block. In high-volume environments, this produces alert fatigue and inconsistent enforcement.

CodeHunter is designed to produce clear policy-driven decisions.

VERDICT	MEANING
Allow	The artifact's behavior is consistent with policy and approved for execution.
Block	The artifact violates policy and is prevented from running.
Restrict	The artifact may run only under defined constraints.
Quarantine	The artifact is isolated pending further analysis.
Require Review	The artifact is escalated for human decisioning when risk or business context requires it.

This moves the organization from probabilistic detection toward repeatable enforcement. Every decision is backed by behavioral evidence, which improves governance, auditability, and security operations consistency.

4 LOCAL ENFORCEMENT

CodeHunter translates behavioral evidence into enforceable execution policy. If an artifact violates policy, the enforcement plane can prevent execution rather than merely forwarding an alert to the SOC.

It separates analysis from enforcement. Heavy analysis can occur in the CodeHunter cloud, while decisions are enforced locally inside the customer environment. This supports lower latency, customer control, and resilience. It also avoids making execution dependent on a real-time cloud round trip for every decision.

The CodeHunter Control Plane

CodeHunter is designed to fit into the workflows enterprises already use to distribute software, deploy workloads, and manage security operations. It acts as an enforcement layer without requiring rip-and-replace changes to the existing stack.

At a high level, CodeHunter turns software execution into a deterministic decision flow.

STEP	WHAT HAPPENS	OUTCOME
1 **SUBMIT**	Artifact is submitted via endpoint, CI/CD, registry, or API	Entry point established
2 **ANALYZE**	Relevant behaviors are detected	Capability identified
3 **PROFILE**	Behavioral profile is generated	Behavior cataloged
4 **DECIDE**	Policy evaluates behavior against context and risk	Verdict assigned
5 **ENFORCE**	Decision is enforced at the control point	Execution controlled
6 **RECORD**	Evidence and verdict are logged	Audit and response enabled

This layered architecture separates analysis from enforcement. With the analysis layer producing behavioral intelligence, the policy layer interpreting that evidence against enterprise rules, and the enforcement layer applying the decision locally.

Once an artifact has been analyzed, CodeHunter can associate the verdict and behavior profile with the artifact hash. This allows repeated instances of the same artifact to be evaluated quickly against current policy. If the artifact changes, if its dependencies change, if the behavior taxonomy changes, or if runtime drift is detected, the artifact can be re-profiled.

CodeHunter can also feed structured behavioral evidence, verdicts, and enforcement outcomes into SIEM, SOAR, ticketing, case management, and compliance workflows. Through APIs, it can integrate with enterprise systems that need to submit artifacts, retrieve verdicts, enforce policy, or consume behavioral evidence.

FROM DETECTION TO EXECUTION CONTROL

Zero Trust for Code is a practical control model for deciding what software should be allowed to execute across endpoints, cloud workloads, software distribution systems, and development pipelines.

To see how Zero Trust for Code works in practice, applied to live artifacts and production workflows, contact CodeHunter to schedule a demo at codehunter.com.

Key Use Cases

Zero Trust for Code can be applied broadly, but four use cases provide a practical starting point.

USE CASE	WHERE CODEHUNTER FITS	BUSINESS VALUE
Software distribution	Before installers, scripts, packages, updates, and internal tools are deployed through endpoint, mobile, server, or software depot workflows.	Reduces risk from trojanized software, compromised update channels, malicious installers, and dangerous third-party tools without disrupting existing deployment systems.
AI-generated and agent-authored code	Before AI-produced artifacts, scripts, packages, or dependencies move downstream or execute under developer or automation privileges.	Adds behavioral verification where human review, provenance, and reputation are weakest.
Software factory and deployment pipelines	Before containers, services, scripts, infrastructure-as-code modules, and deployment artifacts are promoted or launched.	Enforces policy without slowing delivery when behavior is unchanged and policy-compliant.
Endpoint production decision support	When endpoint protection software produces suspicious or inconclusive events a deeper analysis is required to understand software behaviors.	Automatically provide enhanced analysis of alerts when they trigger, allowing for quicker triage and resolution.

Business and Governance Impact

Regulators, auditors, boards, and customers increasingly expect organizations to demonstrate control over software supply chain risk, not just attest that software came from a trusted source. CodeHunter provides a repeatable record of which artifact was evaluated, what behaviors were identified, which policy was applied, what decision was made, whether execution was allowed, and whether runtime behavior later diverged from the approved profile.

EXECUTIVE OUTCOME	WHY IT MATTERS
Reduced risk from unknown and AI-generated code	CodeHunter evaluates behavior before execution, even when the artifact is new, modified, obfuscated, or previously unseen.
Lower alert and triage burden	Deterministic verdicts reduce dependence on analyst interpretation and lower the volume of “maybe malicious” findings.
Faster software approval	Known behavioral profiles and policy-based decisions can accelerate approval for software that stays within expected capability boundaries.
Stronger supply chain control	CodeHunter adds the missing behavioral pillar to SBOM, signing, and provenance programs by answering what the artifact can do.
Improved audit and breach-scoping evidence	Every decision can be backed by behavioral evidence, policy context, verdict history, and enforcement outcomes.
Defense-in-depth without rip and replace	CodeHunter complements endpoint security, application security testing, SIEM, SOAR, CI/CD tooling, and cloud security platforms by adding a pre-execution enforcement layer.

CODEHUNTER®

ZERO TRUST FOR CODE

About CodeHunter

CodeHunter enables organizations to enforce Zero Trust for Software Execution. By analyzing the behavioral intent of software artifacts before they run, CodeHunter provides deterministic execution decisions that prevent unknown, AI-mutated, or internally generated code from executing without verification.

Powered by automated reverse engineering and behavioral analysis, CodeHunter acts as the execution trust layer across endpoint environments and software delivery pipelines. Security teams gain the ability to enforce Allow, Block, Contain, or Escalate decisions before software executes.

About CodeHunter Labs

CodeHunter Labs is the research and development division of CodeHunter, focused on the frontier of binary deconstruction and automated reverse engineering. Our team of security researchers and engineers is dedicated to eliminating software uncertainty by uncovering the "functional DNA" of code before it executes.

Through advanced symbolic execution and deterministic analysis, the Labs provide the technical foundation for the Zero Trust for Code framework: transforming complex forensic data into actionable, pre-execution enforcement.

Learn more at codehunter.com