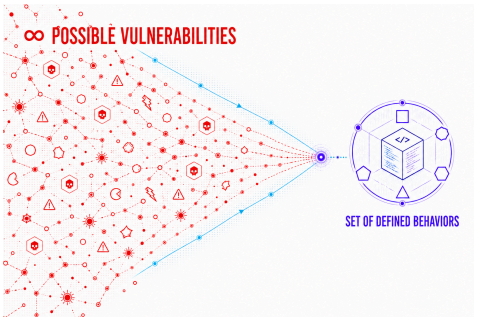
USING PROGRAM BEHAVIOR INTELLIGENCE™ TO PREVENT ZERO-DAY ATTACKS

Zero-day attacks and undisclosed supply chain vulnerabilities are the hardest risks a security team carries. Every control that has to know the threat first leaves them open.

praedictio.ai · September 2026

As attack surfaces expand and zero-day exploitation accelerates, vulnerability management and perimeter defense are no longer sufficient on their own. Protecting a modern application means controlling what it does while it runs, not only cataloging what might be wrong with it before it ships.

Runtime controls that work by recognizing vulnerabilities or known threat vectors inherit the same exposure: anything novel arrives unrecognized. **Program Behavior Intelligence™** (PBI™) inverts the question. Instead of responding to threats, it enforces expected program behavior, and in doing so materially reduces exposure to both known and unknown attacks, including zero-days.



Threats are unbounded. Behavior is not. The set of possible vulnerabilities has no limit. The set of things your application is supposed to do is finite, and you already know it.

> **Zero-day exposure is not a gap in the threat-vector model. It is a consequence of it.**

The structural limit of threat-vector security

Most controls work by identifying vulnerabilities or matching patterns associated with known threats. Machine learning systems are no exception: they remain anchored to the behavioral precursors of attacks someone has already seen. When an attack is novel, that logic has nothing to match it against.

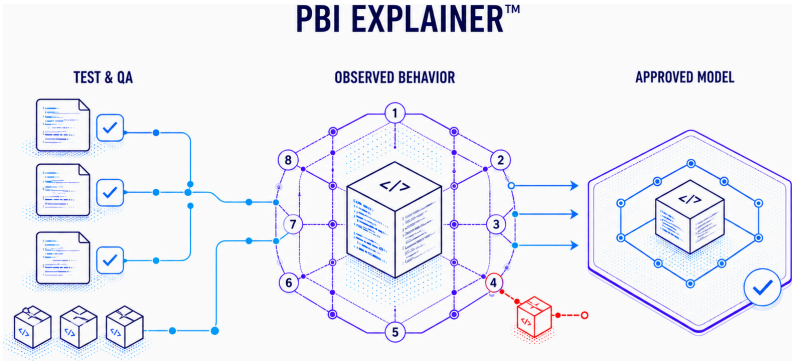
Behavior vectors, not threat vectors

PBI works from behavior vectors. It learns what an application does during testing and controlled use, and builds a deterministic model of acceptable execution. It comes in two parts. **PBI Explainer™** builds that model before release and shows you what it found, including behavior that is hidden. **PBI Enforcement™** applies the model in production, where only behavior inside it is permitted.

What Explainer does

PBI Explainer builds the model and explains it.

It fits into your existing testing and QA process and observes the application as your own tests drive it, recording every legitimate function call, dependency interaction and system response, in the context of the functions executed and the environment they run in. What it records becomes the model: an approved set of actions unique to your program and its implementation. No rules to write, no manual tuning, and the model can be reused across deployments. Where test coverage is thin, Explainer keeps learning during captive normal use, with the program running in production-like conditions but not exposed to attack.



What Explainer sees. Your tests drive the application, Explainer records what actually runs, and that observed behavior becomes the approved model.

It then explains what it found, showing how the application behaves rather than how it was intended to behave, in a behavioral map that supports supply chain assurance, improves test coverage and surfaces latent risk before release.

A conventional SBOM is a static inventory of declared components. Explainer produces a dynamic behavioral SBOM: not which libraries are present, but how components execute, interact and behave under real conditions. It surfaces dormant code paths, unused dependencies and runtime behavior nobody expected.

Explainer ships with ten **PBI Enforcement** licenses, so the model you build is one you can put straight into production.

What Enforcement does

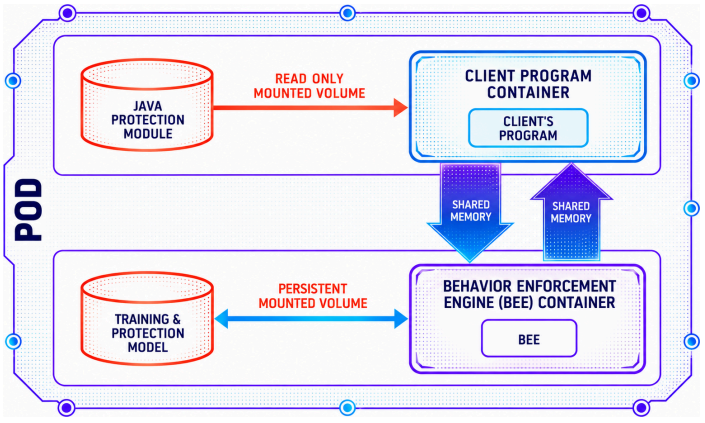
PBI Enforcement applies that model in production. Every action is checked against it before it happens. Expected behavior runs normally. Anything outside the model triggers a response you configure: silent logging, an alert, a process restart, a recovery routine, or a combination.

Enforcement is deterministic, so the outcome is consistent, auditable and repeatable. The model your teams reviewed before release is the model the application is held to in production.

Your software does exactly what it was designed to do, and nothing else.

Operation

PBI runs in user space, in shared memory alongside the program it protects, using carefully tuned runtime instrumentation. In many tests the overhead was indistinguishable from normal system noise. It varies by system and program, but widely accepted security tools impose far greater CPU load, and machine learning approaches trade heavy instrumentation against false positives.



How it deploys. The protection module and the enforcement engine run alongside your program in shared memory, with the model on a persistent volume.

PBI does not repair compromised programs or fix flawed code. It prevents unexpected behavior from executing. The root cause still needs investigation and patching, and PBI records a full trace of any new behavioral path for your teams.

Because it works at a low level, PBI needs no code inspection and very little configuration. Instant Implementation™ makes deployment fast. Training data

stays on your systems, no telemetry comes back to us, and nothing depends on external threat feeds.

What we have tested

We have demonstrated interception of exploits that caused real disruption: Log4j, Spring4Shell, SQL injection, serialization attacks, remote code execution, and Apache ActiveMQ CVE-2023-46604.

Each system was trained using captive normal use, then the exploit was attempted with protection enabled. PBI knew nothing of the exploit, only of known acceptable behavior. The behavior fell outside the model and was stopped.

We have filed patents across thirteen jurisdictions, including the US, the United Kingdom, the EU and Japan, and hold two issued US patents, 12,547,718 and 11,989,296.

Where it fits

PBI Enforcement supports applications written in Java, JavaScript and TypeScript (Node.js), and Python. Others, including .NET, are in development. Because the model learns from observed behavior, deployment depends on reasonable software hygiene: without adequate test coverage there is no reliable baseline, and teams that run fast and break things will find it harder to adopt.

How it compares

Advanced solutions built on machine learning already exist, Wiz and Palo Alto among them. They focus on threats and vulnerabilities, PBI focuses on program behavior. Machine learning can identify the patterns that precede a known threat vector, but it returns a probability, not an answer, and that coverage holds only while the threat is understood.

Program behavior is deterministic. You already know it, from your test cases and from normal use. PBI does not need to learn about new vulnerabilities, only whether what your program is about to do next is expected.

PBI is not a replacement for anything. It augments an existing code-to-cloud strategy. Given what a single zero-day can cost, Jaguar Land Rover being a recent example, the world still does not have adequate application security.

UNDERSTAND EXACTLY WHAT YOUR SOFTWARE DOES. ENFORCE THAT AT RUNTIME.

SEE IT ON YOUR SOFTWARE