WHY A NEW APPROACH

No single technique balances recall, precision & cost

Industrial-grade code security must satisfy recall, precision, performance, stability and cost at once. Every single-engine approach has a structural blind spot.

SYMBOLIC-ONLY

Legacy SAST

Stable and fast — good for full scans — but shallow on meaning.

- \+ Stable performance at large-scale full scans
- – High false-positive noise, costly manual review
- – Weak cross-function / cross-file / business-logic analysis

SEMANTIC-ONLY

LLM-only

Strong semantics and intent — but hard to scale affordably.

- \+ Understands context and business intent
- – Slow on large codebases, high token cost
- – Stability hinges on model / prompt / context quality

61%

say security testing noticeably slows development

Global State of DevSecOps 2024

82%

of orgs run 6–20 security testing tools in parallel

Global State of DevSecOps 2024

>90%

initial false-positive noise in research scenarios

OWASP Benchmark

The takeaway. Industrial code security can't rely on legacy SAST alone, nor on LLM-only — deterministic full-scan and deep semantic analysis must be fused.

THE CLEANCODE APPROACH

A dual-engine hybrid architecture

Understands code structure and security semantics — deep fusion, mutual reinforcement.

INPUT
AI-Generated Code
Coding Agent output

→

DETECT
Defect Detection
Dual-engine precision

→

TRIAGE
Smart Triage
AI semantic denoise

→

FIX
One-Click Fix
Auto-generated patch

→

OUTPUT
Secure Code
Baseline held

CleanCode Security Agent

THE PRODUCT

The code-security verification layer for the AI programming era. Once a Coding Agent writes code, CleanCode auto-runs the full **detect · triage · fix** loop — so teams keep AI's speed without dropping the security baseline.

SYMBOLIC ENGINE

Symbolic Engine

Deterministic full scan — walks execution paths to fence off potential defect zones.

`CFG / Call Graph / IR` `PTA / Alias / Dataflow` `Taint Analysis` `Symbolic Exec / Path Constraints`

SEMANTIC ENGINE

Semantic Engine

Deep semantic analysis of defect zones with a multi-agent "4D" triage and fix generation.

`Framework Behavior Model` `FP-TP Label Library` `Multi-Agent Triage` `Reachability Analysis`

HIGHLIGHTS & SPECIFICATIONS

Six capabilities, rebuilt for the AI era

Symbolic + Semantic
Dual-engine hybrid

Multi-Agent "4D" Triage

Verifiable conclusions — not just "is there a vuln?"

01
Verdict
Confirmed / false positive / needs verification.

02
Root Cause
Pinpoints the underlying cause and rule basis.

03
Fix Suggestion
Concrete, executable code-level remediation.

04
Trigger Condition
Reconstructs the trigger scenario and reachable path.

WHERE
Exact file, line number and trigger path.

WHY
Root cause, risk impact and rule basis.

HOW
Fix guidance, AI actions and a closed loop.

Complete Detection Coverage

Multi-language · multi-framework · multi-platform + key base libraries.

Vulnerability Classes

Basic security — input validation, web security, authn/authz

Data security — encryption, sensitive data, file operations

Code risk — API misuse, memory safety, concurrency

Java Frameworks

Web — Spring, Spring MVC

ORM / data access — JDBC, Hibernate, MyBatis, JPA

Middleware — Dubbo, Netty, Tomcat, Jetty

Compilers

Mainstream — Clang, GNU GCC / G++

Commercial — MSVC, Sun C++

Embedded — Green Hills, ARM C/C++

C / C++ Base Libraries

System — libc / glibc / musl / bionic

C++ standard — libstdc++, libc++

Key libs — OpenSSL, SQLite, Protobuf, libpng

01
More accurate
Hybrid symbolic + LLM breaks past both SAST and pure-LLM ceilings.

02
Fewer false alarms
AI semantics filter noise and cut alert fatigue sharply.

03
Faster fixes
Not just detection — auto-generated, executable patches.

04
Developer-friendly
Readable, fixable, precise — inside the IDE / Agent flow.

05
+60% fix efficiency
A full detect-to-fix loop compresses the remediation cycle.

06
Seamless integration
Native MCP interop with mainstream Coding Agents.

SEAMLESS INTEGRATION · MCP

Drops in as the code-security verification layer of any AI coding workflow.

Cursor | Kimi Code | CodeBuddy

SECTREND

Beyond security, more than open source.

1616 Tower A, Lvdihui Center
500 Yunjin Road, Shanghai China
+86 21-64181628 · info@sectrend.com.cn
www.sectrend.com.cn

Business

WeChat