

We find more real vulnerabilities than the leading static scanners

Head-to-head on an independent, labelled security benchmark — static detection, plus a live layer that *proves* each finding.

On a benchmark of **7 deliberately-vulnerable applications across 6 languages**, Verdict Runtime detected **78 of 152** statically-findable flaws — ahead of **CodeQL (66)** and **Snyk Code (65)** — and led in **5 of the 6 languages**. On top of that, our live exploit layer confirmed **9 additional vulnerabilities** that no static scanner can find at all.

78_{/152}

Flaws found —
Verdict Runtime

66

CodeQL · +12 for us

65

Snyk Code · +13 for
us

5_{/6}

Languages we lead

■ Detection recall by language

A flaw counts as found only when the correct vulnerability class is reported within ±3 lines of the labelled location, scored against each app's own answer key. Static analysis only, for a fair comparison against two static tools.

BENCHMARK APP (LANGUAGE)	FINDABLE	VERDICT RUNTIME	SNYK CODE	CODEQL	LEADER
Go web app	4	4	2	2	Us
Ruby on Rails app	17	10	7	8	Us
Python (Django) app	29	24	16	16	Us
JavaScript app A	17	10	8	6	Us
JavaScript app B	9	6	3	6	Tie
C# / .NET app	36	5	4	2	Us
Java app	40	19	25	26	CodeQL
All seven	152	78	65	66	Us

■ Where we pull ahead

- **Python:** 24 vs 16 — our deepest lead, +8 over both competitors.
- **Go & .NET:** top of the field where the others miss the most.
- **JavaScript:** ahead on client-side and injection flaws the others skip.
- Multi-engine design catches flaws any single engine misses.

■ Honest about the one loss

- **Java:** CodeQL and Snyk Code lead (26 / 25 vs our 19) on a large Java benchmark — a known strength area we're closing.
- We show it because the rest of the numbers are only as trustworthy as the one we'd rather not.

■ The layer the static scanners don't have

LIVE PROOF

9 additional vulnerabilities confirmed by actually exploiting the running apps

Verdict Runtime doesn't just read code — its agents send real, scope-authorized requests and **prove** each finding against the live application. Across three running benchmark apps they confirmed, with a reproducing request each:

- **Broken object authorization (BOLA/IDOR)** — one account read another account's private data
- **Authentication-token forgery** — an unsigned (`alg:none`) token accepted as valid
- **Exposed admin / debug surfaces** — internal config and server logs served to an unauthenticated request
- **Missing login rate-limiting & weak session cookies** across all three apps

Not one of these classes is detectable by Snyk Code or CodeQL — they are logic and runtime flaws that only surface when you exercise the live application. We don't report risks; we deliver verdicts, with proof.

Method & scope. Benchmark of 7 deliberately-vulnerable training/benchmark applications (6 languages), each scored against its own published answer key; a detection counts only when the correct vulnerability class is reported within ± 3 lines. Each competing scanner was run at its documented default / security-extended settings. The static-recall comparison is static-analysis-only for fairness; the live-proof results are from Verdict Runtime's agent layer against disposable, explicitly-authorized local instances. These are labelled benchmark targets, so results demonstrate measured advantage on this suite and are not a claim of universal superiority. Open-source dependency findings were comparable to the competitor's open-source product and are not scored here. Figures verified against run artifacts, 2026-10-18.