Secure Velocity:
What SAST Changes in Open-Source Software

Why This Matters Now

Static Application Security Testing (SAST) analyzes source code before it runs to catch flaws like injection, hardcoded secrets and insecure configurations. It matters more than ever today because, with the rise of agentic engineering and the acceleration of software development, code is being produced at unprecedented speed and volume - often by AI agents operating with significant autonomy. SAST provides a critical automated safety net, ensuring that security keeps pace with delivery by surfacing risks at the point of creation rather than after deployment, when remediation is far more costly.

Understanding SAST's Role, and its Limits

SAST is a first-line, preventative control; it complements (not replaces) other security tests.

What SAST Does

SAST analyzes source code, bytecode or binaries without executing them, using techniques like pattern matching, data-flow/taint tracking and control-flow reasoning to flag *risky* constructs (e.g., injection, unsafe deserialization, path traversal). It integrates directly into developer workflows and CI/CD so issues surface during review, before deployment.

What SAST Doesn't Do

It doesn't "prove" the absence of vulnerabilities, and it can't see environment-specific behavior. It should sit alongside dynamic application security testing (DAST), fuzzing, software composition analysis (SCA) and human code review. Framed correctly, SAST reduces load on those later-stage controls by eliminating obvious issues early.

What Changes When You Enable SAST

SAST changes workflow more than velocity, if introduced incrementally and tuned for signal.

Pipelines and Gating
Analysis add continuous implementation time; most organizations start in "warn" mode for new findings and switch to "block" after tuning rules and developer trust improves. Clear policies (what fails a build vs. what creates a ticket) keep momentum.

Developer Experience
Signal quality is everything. Begin with high-confidence, high-severity rules to build trust; expand coverage once teams see value. Findings should be specific, actionable and linked to code locations in developer tooling.

Ownership and Routing
Decide whether triage sits with module owners or a central application security team. Route findings automatically to code owners to avoid bottlenecks and reduce context switching.

Policy and Auditability
Suppression should require justification, preventing re-triage loops and creating an auditable trail. Over time, rule tuning and baselining stabilize workload.

Evidence From the Field: What the Data Shows

Scope

The top 300 projects, drawn from npm, PyPI and Maven, were selected on GitHub stars as a proxy for popularity. All but six were hosted in GitHub, enabling 294 to be analyzed by a purpose-build harness to examine pipeline configurations.

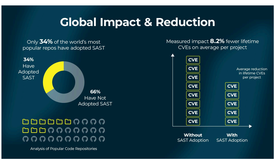
Adoption Snapshot

- No SAST in CI/CD: 65.6%
- CodeQL present: 30.3%
- other SAST tools: 4.1%

Outcome

Across the cohort's 630 total CVEs observed over project lifetimes:

- No SAST: 2.21 CVEs per project (avg.)
- SAST enabled: 2.04 CVEs per project (avg.)
- 8.2% relative reduction where SAST is present



Interpretation

This is correlation, not proof of causation; but it aligns with SAST's preventative purpose. Also, because CVE systems catalog public disclosures, not every latent defect, the true risk reduction from SAST is likely larger than what CVE counts alone can show.

Adopting SAST Without Slowing Teams

A small set of well-chosen rules, applied to new code, delivers outsized early gains.

Start Simple, Aim for Trust
- New code first: Turn on SAST for all new pull requests; create a baseline for existing debt to avoid noisy backlogs.
- High-confidence rules: Begin with injection, unsafe deserialization and path traversal; widen coverage after the signal proves useful.
- Warn to block: Gate merges only after false-positive rates are under control and developers see value.

Keep The Workload Under Control
- Ownership routing: Auto-assign to code owners; avoid central queues becoming the bottleneck.
- Suppression with rationale: Require a short justification to reduce the impact of resurfaced poor findings.
- Auto-fix where safe: Use ecosystem-supported refactors to reduce toil on repetitive patterns.

Pair With Adjacent Controls
SAST covers first-party code. Pair it with software composition analysis for third-party dependencies and dynamic application security testing for runtime behavior to address the full attack surface. This layered approach reflects modern secure development guidance.

Leadership Metrics That Matter

Measure behavior change, not just scanner throughput.

- Fix rate for new issues: Are high-severity findings resolved within service level agreements?
- False-positive rate: Are developers' trust and compliance improving?
- Time-to-remediate (TTR): From PR to merge for security fixes.
- Hotspot trends: Which modules repeatedly introduce risk and need enablement?
- Adoption coverage: % of repos (and critical services) with SAST enabled and enforcing.

These metrics describe **operational assurance**, not just tool activity. They also create a clear conversation with auditors and customers about preventative controls embedded in delivery.

Recommendations

Takeaway for leaders: *Make it default, simple and let teams keep moving.*

1. Enable SAST by default for new repositories; baseline mature codebases and burn down over time.
2. Choose one primary tool aligned to your ecosystem, add complementary rules or tools later and avoid a noisy tool pile-up.
3. Define a triage SLA and merge policy focused on high-severity, high-confidence findings; keep suppressions auditable.
4. Treat dependencies separately; pair SAST with SCA to widen the risks caught.

Conclusion

The competitive edge doesn't only come from writing code fastest; it comes from shipping safely at speed. In popular open-source projects, SAST presence correlates with fewer disclosed vulnerabilities and most projects still haven't enabled it, leaving clear headroom for improvement. For enterprises, the message is simple: enforce SAST for new code, measure the signals and keep moving. You'll reduce risk without reducing velocity.

FAQ

1. Will SAST slow our developers down?

Not if adoption is staged. Start with warn-only on new code and a small ruleset; move to blocking once false positives are under control and developers trust the findings. CI time increases slightly, but merge friction remains low when policies and ownership are clear.

2. Is the 8.2% CVE reduction causal proof?

It's an observed association across 294 projects. It's consistent with SAST's preventative design, and because CVE lists track public disclosures, not every latent flaw, the true reduction in exploitable defects is plausibly larger than CVE deltas show.

3. Which tool should we start with?

Pick one that fits your languages and workflow and integrates with your CI/CD. This report does not recommend or quantify the difference in any specific tool.

4. We already use SCA and DAST, do we still need SAST?

Yes. SCA manages third-party dependency risk; DAST tests running apps. SAST finds issues before they run. These approaches are complementary, not mutually exclusive.

5. How do we handle false positives and alert fatigue?

Baseline existing findings start with high-confidence rules, require justified suppressions and route ownership to the right module maintainers. Trust improves as signal quality improves.

6. What should we show customers or auditors?

Demonstrate adoption coverage (repos with SAST enabled), policy settings (what blocks vs. warns), triage SLAs and metrics (fix-rate, TTR, false-positive rate). Code-scanning documentation helps evidence how scanning is embedded in delivery.

Sources:

Newton, B. (2026). Analysis of SAST adoption and CVE incidence across 294 popular GitHub repositories, spanning npm, PyPI, and Maven ecosystems. Original analysis conducted Q1 2026 using publicly available repository and pipeline data.

GitHub. What is Static Application Security Testing (SAST)?
https://github.com/resources/articles/what-is-sast

Palo Alto Networks. What Is Static Application Security Testing (SAST)?
https://www.paloaltonetworks.com/cyberpedia/what-is-sast-static-application-security-testing

OWASP Foundation. Top 10 Risks for Open-source Software.
https://owasp.org/www-project-open-source-software-top-10/

Black Duck. 2025 Open-source Security and Risk Analysis Report.
https://www.blackduck.com/resources/report/2025-open-source-security-risk-analysis

MITRE Corporation. Common Vulnerabilities and Exposures (CVE) Program.
https://www.cve.org/

National Institute of Standards and Technology (NIST). National Vulnerability Database (NVD).
https://nvd.nist.gov/

IBM. What is CVE (Common Vulnerabilities and Exposures)?
https://www.ibm.com/think/topics/cve