CONTRAST SECURITY

SOLUTION BRIEF

Agent performance

Security from the inside out — at production scale

Security tools that operate outside the application runtime are blind to the attacks that matter most. They watch from the perimeter while exploits traverse application logic undetected. Contrast closes that gap — operating within runtime to detect and block real attacks as they happen and to surface vulnerabilities in the context of actual production behavior.

This is runtime security done right: continuous, contextual and built to run where risk is real. The natural question is what it costs to get there.

Real-world performance

This is what customers running Contrast in production actually see:

Real-world customer telemetry: ADR, across production environments

5%
Response time overhead for 70% of production requests

< 1 ms
Response time overhead for ~90% of requests

Based on 4.4 billion requests across production customer environments

How we benchmark

We also run controlled benchmarks against a realistic enterprise application to understand performance under demanding, consistent conditions. The figures below reflect a Jira 9.13 environment designed to surface worst-case behavior — a conservative anchor for evaluating overhead.

ADR + security observability: Internal Jira benchmark

~12–15%
Response time overhead

~10–12%
CPU overhead

~650–750 MB
Additional memory

About this benchmark

Measured against Jira 9.13 running on 8 vCPUs and 12 GB memory, under 20 concurrent users over 30 minutes on AWS EC2. High statistical confidence across 18 independent test runs. Figures reflect ADR and Security Observability combined — the additional load from security observability reflects the continuous architectural mapping it performs. Results reflect the Contrast Java agent; performance characteristics are expected to be directionally consistent across supported languages. The Jira test averages 46 KB input payloads — significantly larger than the sub-100-byte requests that make up most real production traffic.

What affects these numbers

The figures above reflect a real but specific environment. A number of factors influence actual overhead in your deployment:

- **Payload size:** Over 90% of production requests carry less than 100 bytes of input, at which point overhead is at its lowest. Applications that send larger payloads will see a proportionally higher impact.
- **Traffic volume:** The benchmark ran a consistent set of users; significantly higher concurrency may amplify CPU and memory impact.
- **Application behavior:** Applications handling security-relevant traffic, such as database queries, authentication or outbound API calls, will see the agent perform more analysis per request, pushing overhead higher.
- **Container sizing:** Applications running lean on memory may need to provision additional headroom to accommodate agent overhead.

IAST in production

Contrast also brings IAST to production. Finding vulnerabilities as real users exercise the application under real conditions. Most teams deploy IAST on a subset of instances per service — enough to exercise all code paths under real traffic — without touching the rest of your fleet. In those instances, Contrast's pacing algorithm handles the rest.

1	**Active analysis**	As real traffic hits the application, Contrast analyzes initial requests to each route — discovering vulnerabilities as they execute in live production conditions.
2	**Route verified**	Once a route is exercised and confirmed, it is marked complete. Full analysis stops for that route until the next code change.
3	**Near-zero overhead**	Analysis deactivates until the next deployment. When new code ships, only affected routes are re-examined — as real traffic exercises them, not all at once.

Overhead is momentary and concentrated on new routes after new deployments. Once a route is verified, Contrast steps back. When the instances are not running, IAST does not observe this impact.

6800 Koll Center Parkway
Ste 235
Pleasanton, CA 94566
Phone: 888.371.1333