Continuously Verify: Agentic Fuzzing to Improve Your Organizational Security



DAFITC 2026

AGENDA

00. INTROS

01. STATE OF THE UNION

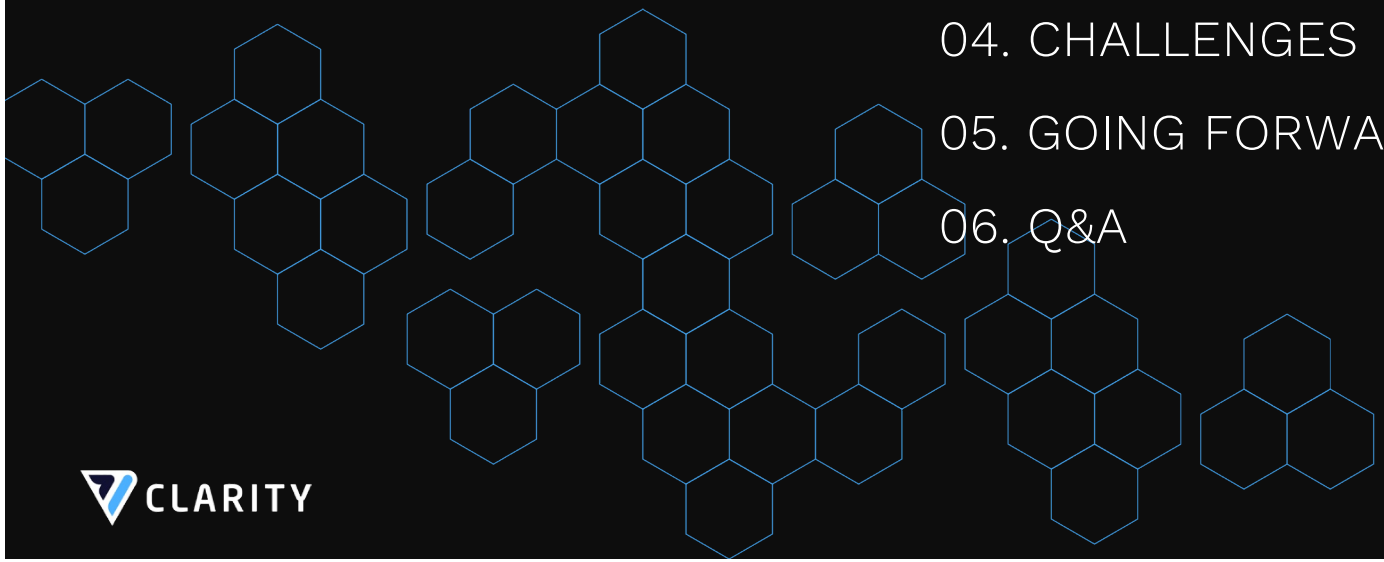
02. ENTER THE AGENTS

03. PIPELINE ARCHITECTURE

04. CHALLENGES

05. GOING FORWARD

06. Q&A



CLARITY

Introductions



Ricky Orndorff
Chief Mission Architect

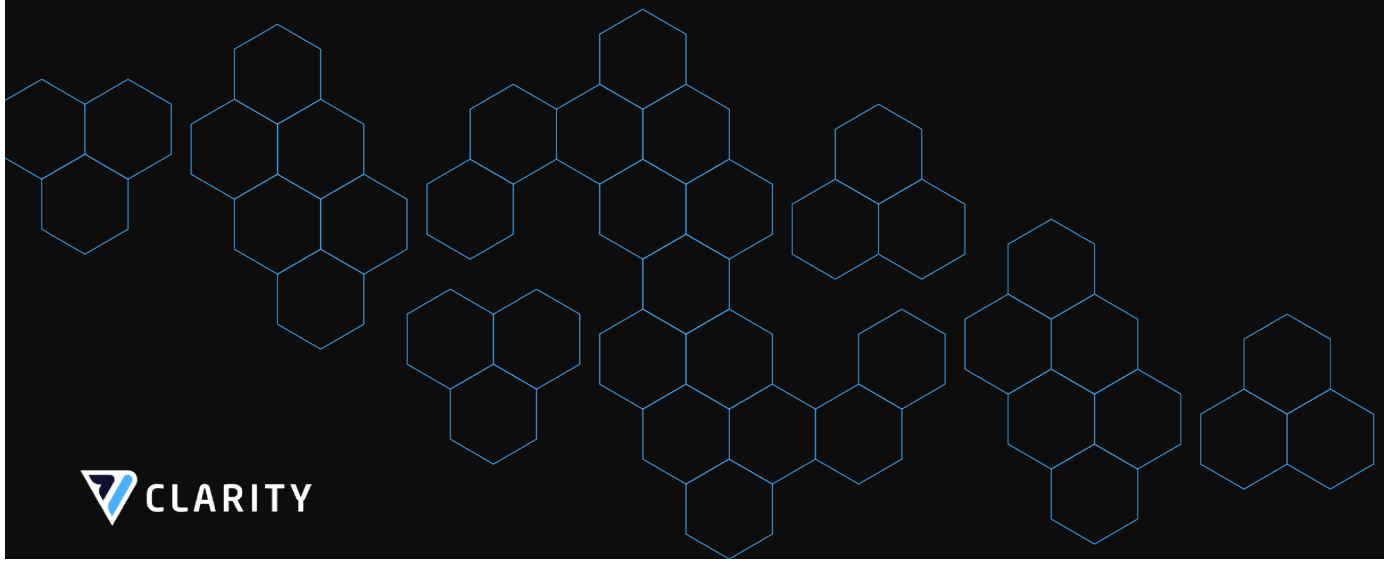
10+ Years of Mission Application Development Supporting CNO in IC & AF



Mike Lanciano
CTO, Enterprise Warfighting

10+ Years of Platform Delivery, Cyber Security, and Data Engineering in IC & DoW

01. STATE OF THE UNION



CLARITY

Where We Are Today

01

Post-Mythos World

We live in a complex, post-mythos environment where traditional security boundaries have completely dissolved.

02

Continuous Offense

The adversary now employs ongoing, autonomous offense systems that probe your enterprise around the clock.

03

System Exposure

Ordinary everyday business systems carry real operational risk and are just as vulnerable as vital C2 systems.

04

Ready to Evolve

We have the foundational platform, local models, and enclave infrastructure required to continuously adapt.

CLARITY

Compliance tells you what you documented.
Attackers exploit what you didn’t.

True security exposure lives in the inputs nobody planned for — the uncharted territory where leaks actually happen.

CLARITY

Three things that are not the same



01 **Compliance**

Confirms a control matches your documentation. A checkbox, assessed on a cycle.

02 **Pipeline tests**

Confirm precisely what you anticipated. Green means your assumptions held — not that none were missed.

03 **Real exposure**

Lives in the missed input, the wrong assumption, and the misconfiguration. This is where breaches actually start.

Checklists find the known. **Breaches come from the unknown.**

So how do you identify the unknown gaps?

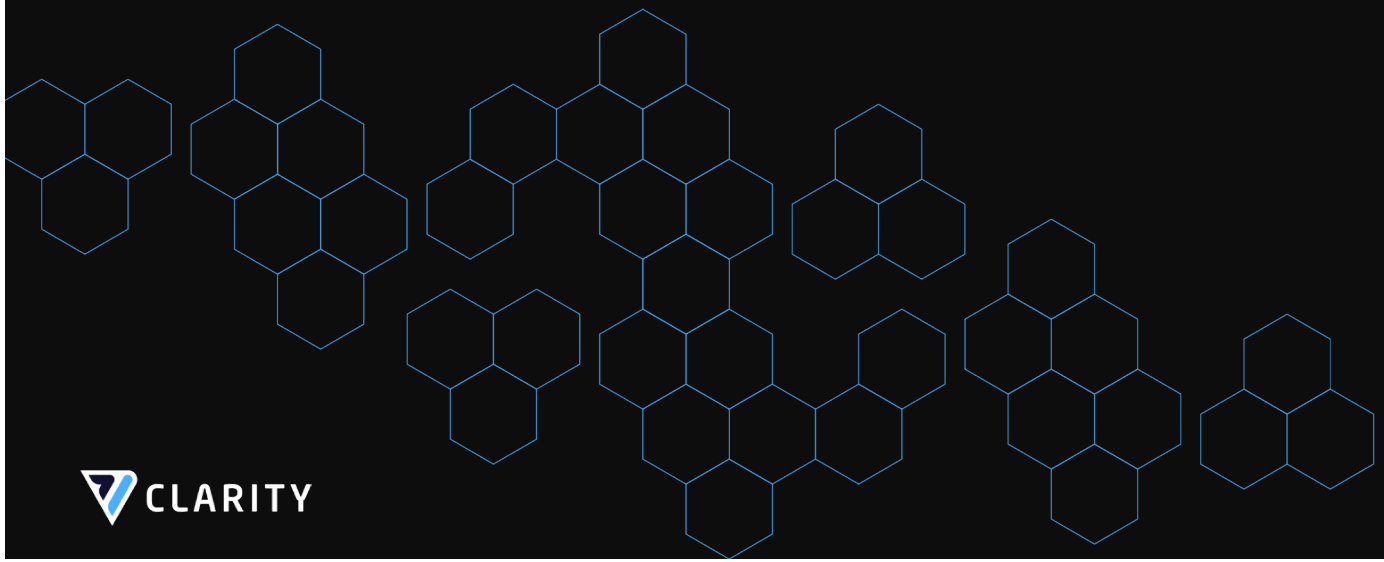
CLARITY

In Vulnerability Research, we use the concept of fuzzing.

What is fuzzing?

Generating mutated or grammar-based inputs to test software boundaries, relying on implicit failure detection (like system crashes) rather than hardcoded, explicit test outcomes.

02. ENTER THE AGENTS



CLARITY

So are we really fuzzing?

Yes, we are using fuzzing techniques where applicable and wrapping them in an autonomous AI agent. We are using the philosophy of fuzzing to represent a state of continuous evaluation, red-teaming, and VR to evaluate and test for security issues that **evolve or get missed**.

Traditional Fuzzing

A brute-force attempt focused purely on volume and aimed at detecting very specific, known types of vulnerabilities.

Agentic “Fuzzing”

Smarter, context-aware, and built on a deep understanding of code structure, logic, and software mechanics but still evaluates beyond the scope.

What does an agent add?

Turning brute force to guided reasoning.

01

Writes the harnesses

Generates fuzz targets and seed inputs for code nobody wrote tests for reaching paths manual harnesses miss.

02

Triages at scale

Clusters and minimizes a flood of crashes into distinct, real bugs — the human-hours sink of classic fuzzing.

03

Reasons about exploit

Agent utilizes tools, such as a debugger, to derive root cause and augment human-in-the-loop risk evaluation.

04

Drafts PoC

Produces proof-of-concept plus fix guidance for evaluation by the human implementation.

AI Widens The Aperture

Traditional Fuzzing

- Buffer Overflow / Out-of-bounds read-write
- Use-after-free
- Integer Overflow
- Memory Corruption
- Parser & Deserialization Crashes
- Protocol / State-machine Crashes

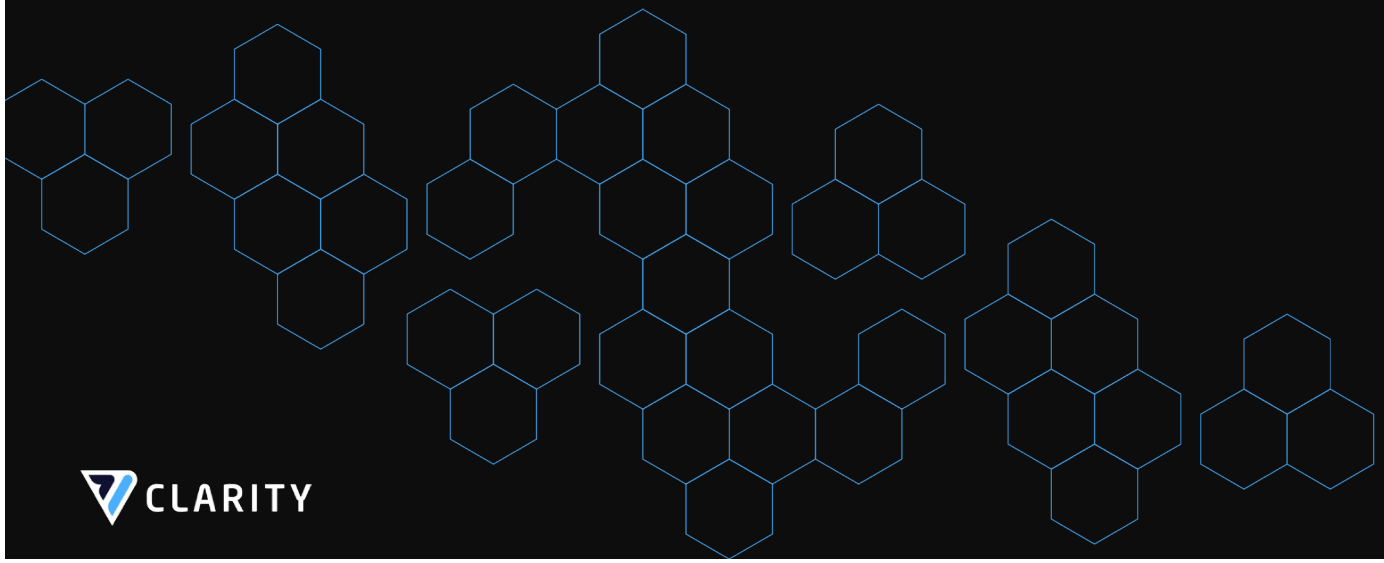
+

Continuous Agentic Evaluation

- Broken Access Control / IDOR
- Business-logic flaws
- Auth / Session Bypass
- SSRF
- Missing Authorization Checks
- Insecure Defaults
- Misconfigurations
- Drift

CLARITY

03. PIPELINE ARCHITECTURE



CLARITY

Agentic Fuzzing Pipeline

CLARITY

01 Ingest
Pull in everything you own and depend on:
- Code repos (git)
- Firmware images
- Dependency tree / SBOM
- IaC & live config

02 Harness Gen
Local LLM writes fuzz harnesses and seed inputs for code humans never targeted.
AI reaches code manual harnesses miss

03 Fuzzing Farm
Coverage-guided fuzzers (AFL++/libFuzzer) hammer inputs 24/7, steering to new paths.
Emits: crashes + coverage

04 Triage & Dedup
Thousands of raw crashes are bucketed and de-duplicated into distinct root-cause bugs.
12,704 raw → 61 unique

05 Reachability
Local LLM drops into GDB debugger to reason like a researcher: is this truly exploitable?
Separates real risk from noise

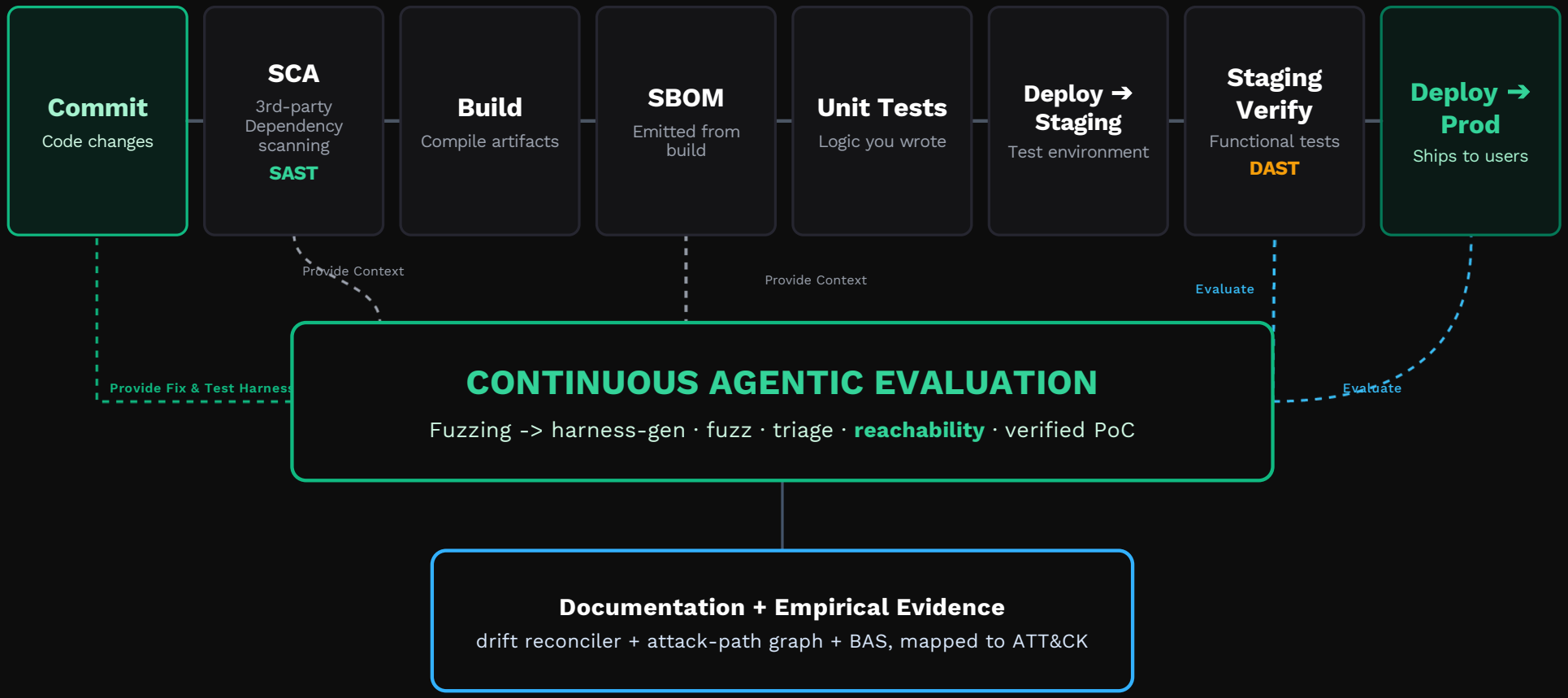
06 Auto-PoC
Drafts a working proof-of-concept and re-runs to prove the bug is real and reproducible.
✦ Human review gate

07 Remediate
Rank by exploitability, open tickets, and feed the blue/purple team & VR platform.
Fix what attackers can reach

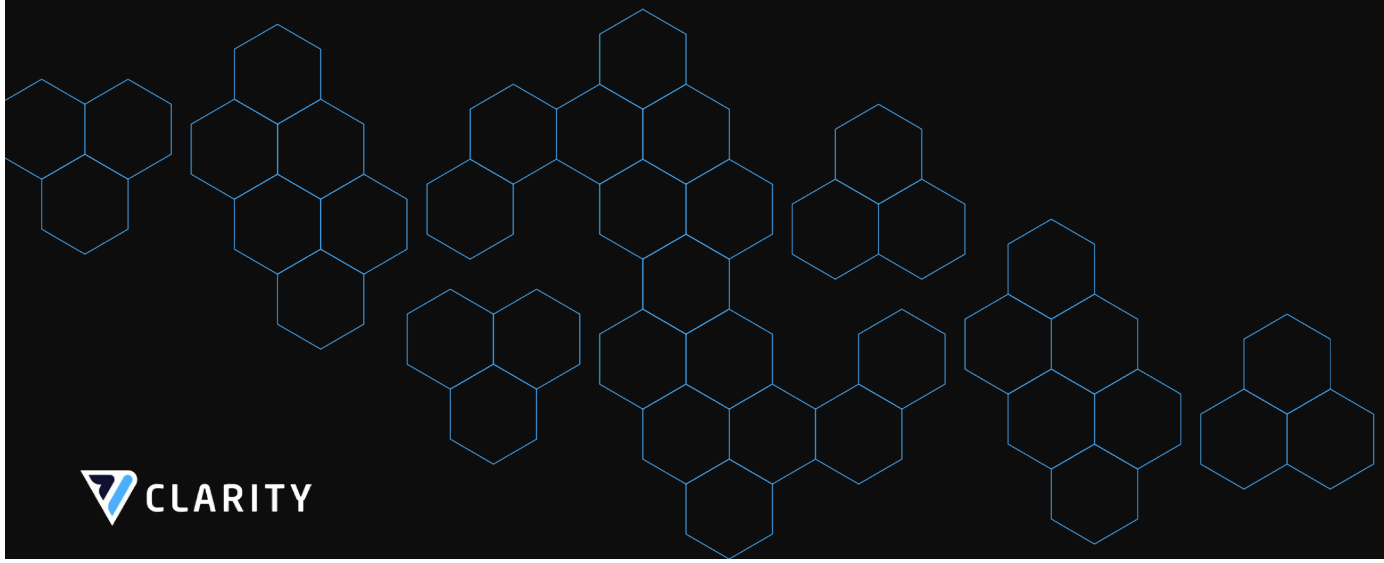
↻ Continuous Loop
New patches, new deps, and new coverage feed straight back into ingest & fuzzing engine.
Fully automated pipeline cycle

CI/CD INTEGRATION

CLARITY



04. CHALLENGES



CLARITY

The Agentic Challenges

CLARITY

01 / LIMITATIONS

No Easy Fixes

LLMs do not replace human vulnerability researchers. Standard open prompts lead to hallucinated exploits and failed dynamic library connections.

02 / THE PROCESS

Bounded Sandboxes

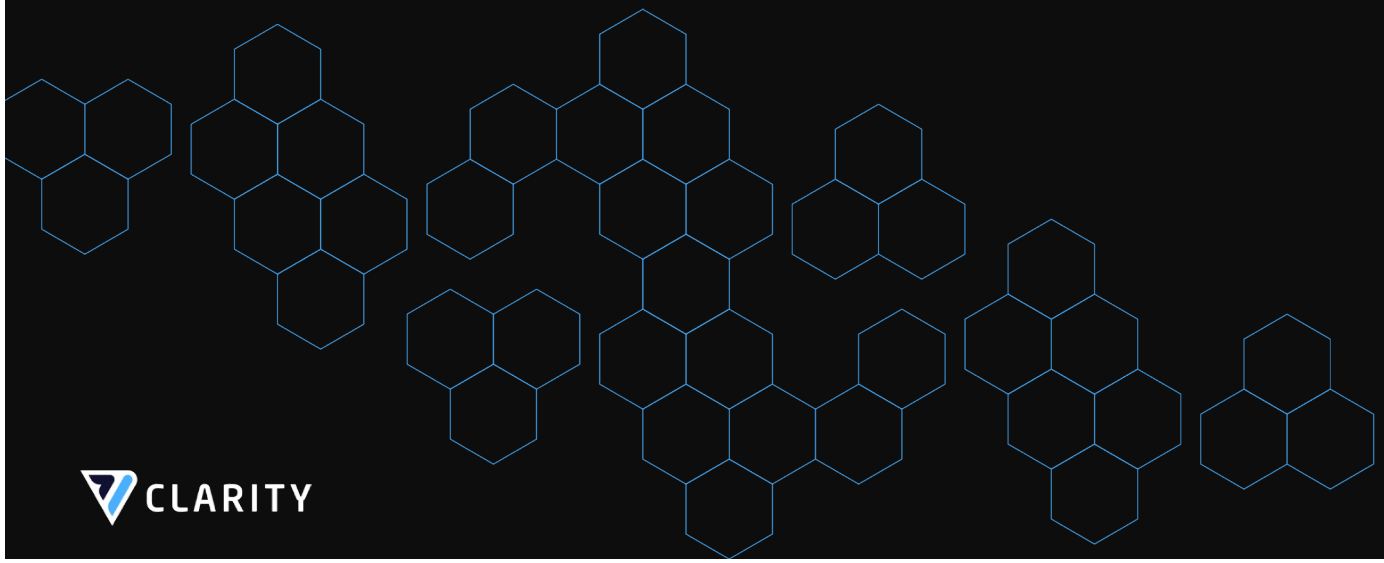
Effective agents operate in tool-augmented harness pipelines, strictly constrained by real compiler feedback and execution sandboxes.

03 / THE GOAL

Augmenting Talent

True vulnerability research expertise is scarce but critical. The objective is to augment existing engineers who possess overlapping security skills.

05. GOING FORWARD



CLARITY

Crawl, walk, run

bug_reportCRAWL

- Pick one crown-jewel app + its dependency tree
- Turn on coverage-guided fuzzing in CI (libFuzzer/AFL++ + sanitizers)
- Stand up SBOM + dependency monitoring
- Triage findings manually

directions_walkWALK

- Add a local LLM for harness-gen + crash triage
- Score reachability & exploitability
- Run a state-drift reconciler on one enclave
- Add a human review gate

directions_runRUN

- Continuous fuzzing farm across the portfolio
- Auto-PoC + attack-path graph
- Feed the purple team + VR platform
- Produce cATO monitoring evidence

CLARITY

Key Takeaways

Compliance says you're allowed to operate.

Continuous verification proves you can.

01 / PERSPECTIVE

Compliance vs. Offense

Checklists prove what you documented; agentic evaluation proves what you actually deployed.

02 / METHOD

Guided Reasoning

Traditional fuzzing brings the throughput; LLM agents bring the harness generation, triage, and contextual awareness.

03 / TALENT

Force Multiplier

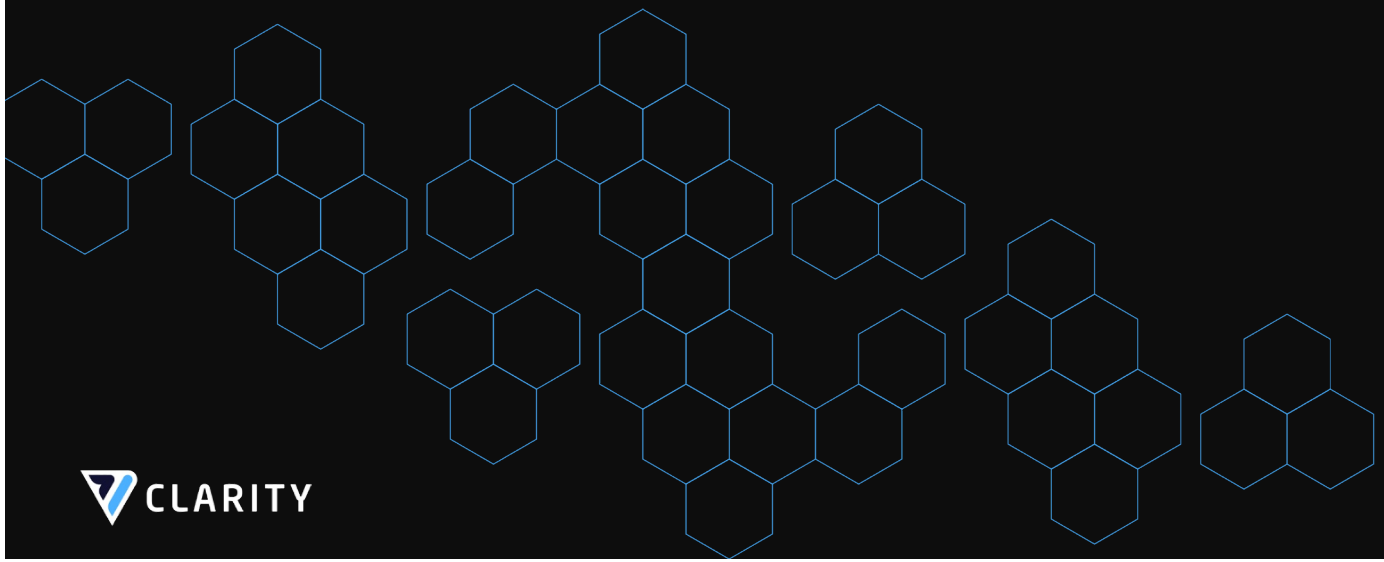
Use agents to automate the baseline so your engineers focus on high-impact mission risk instead of basic harnesses.

04 / PROOF

Empirical Verification

We continuously evaluate that we are doing what we say we are doing, finding new gaps faster and at greater scale.

06. Q&A



CLARITY