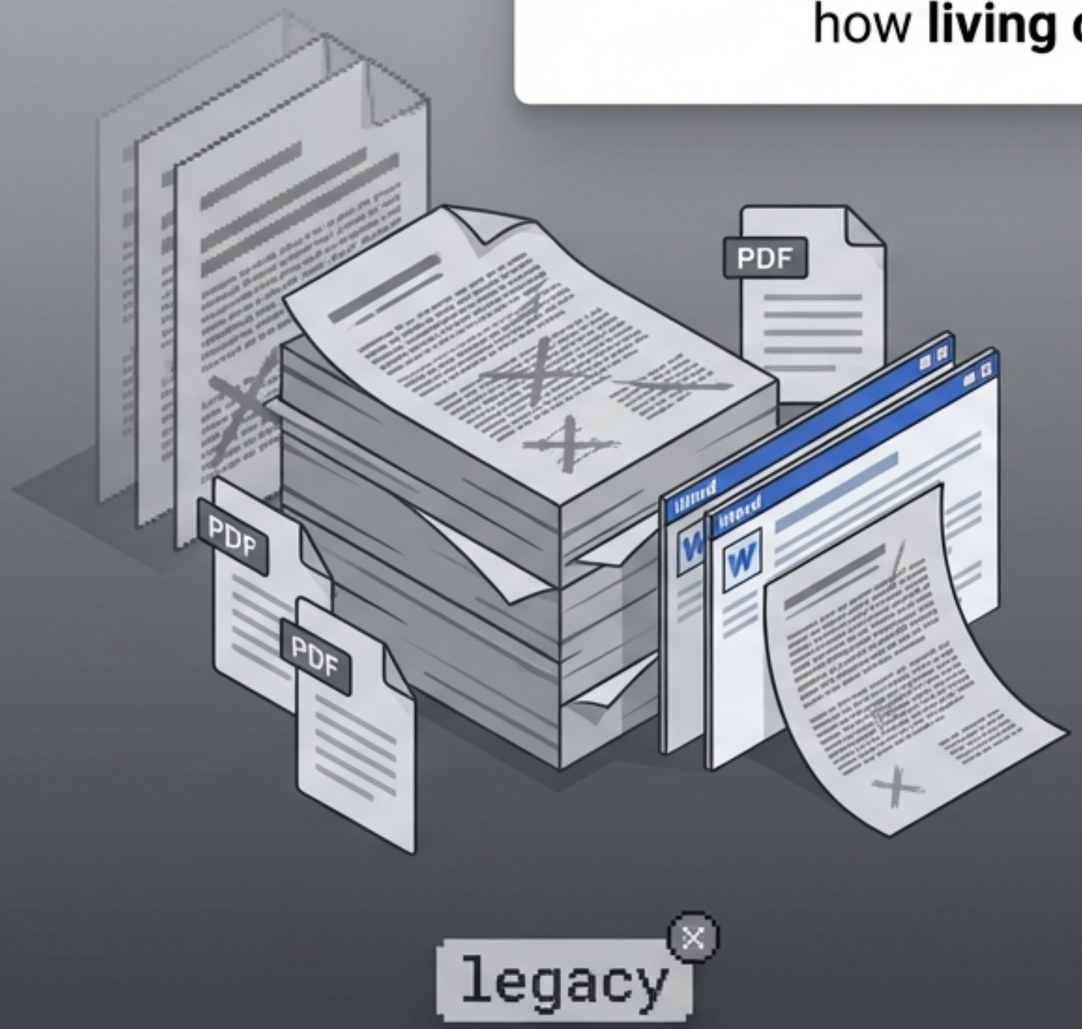
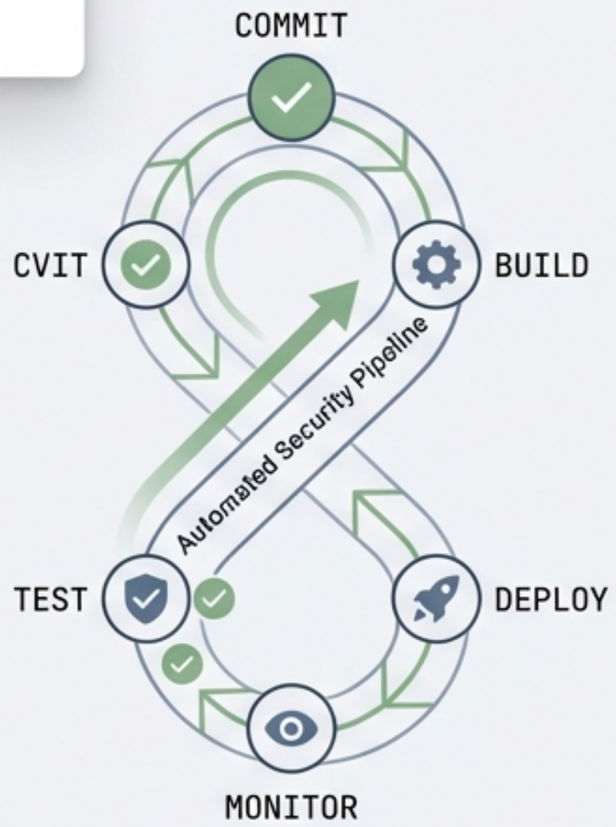
Security Testing as Code (STaC)

Why audit reports die upon publication—and how **living code** is replacing them

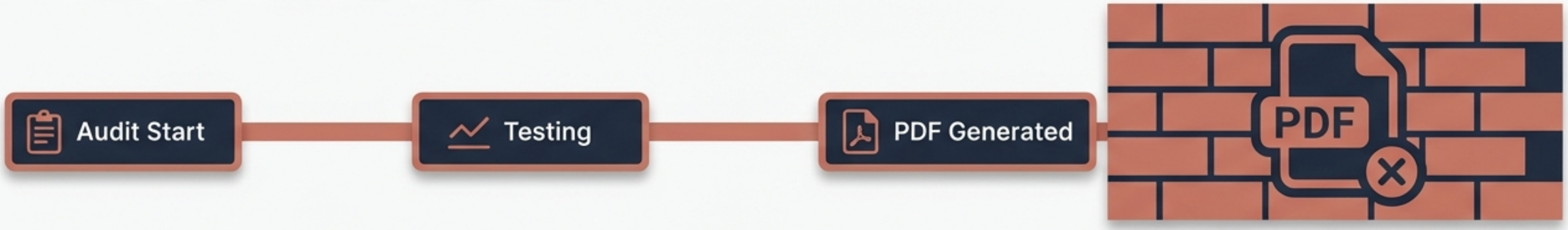




LIVING CODE

The Moment a Report is Published, It Dies

The Status Quo: The Medical Certificate



A snapshot in time. The knowledge evaporates, and the next auditor starts from scratch.

The STaC Method: The Continuous Medical Record

Architecturaal layer cake, medical record.
Living history. Context, test parameters, and executable proof transfer continuously to the next phase.

Key Takeaway: We are treating security assessments like static medical certificates when complex systems require continuous medical records.

Changing the Format Didn’t Change the Outcome

Word (2010s)

Focus: Volume > Substance

Flaw: High page counts filled with generic OWASP copy-paste. 'Critical 3, High 7' lacks contextual justification.

Excel (2012+)

Focus: Quantifiable Aggregation

Flaw: Fragmented over time. Custom tabs and lost files meant results survived, but context died.

Portals (2015+)

Focus: Centralized Databases

Flaw: Rigid schemas. Stores the WHAT, but has no space for the HOW or WHY.

The root cause remains: All previous iterations assumed security testing is a DOCUMENT. Documents cannot execute.

Security Testing is Not a Document. It's a Project.

LIVING CODE

The Three Core Axioms of STaC

1. Findings are Evidence, Not Claims

A vulnerability is a hypothesis until proven. Proof requires code.

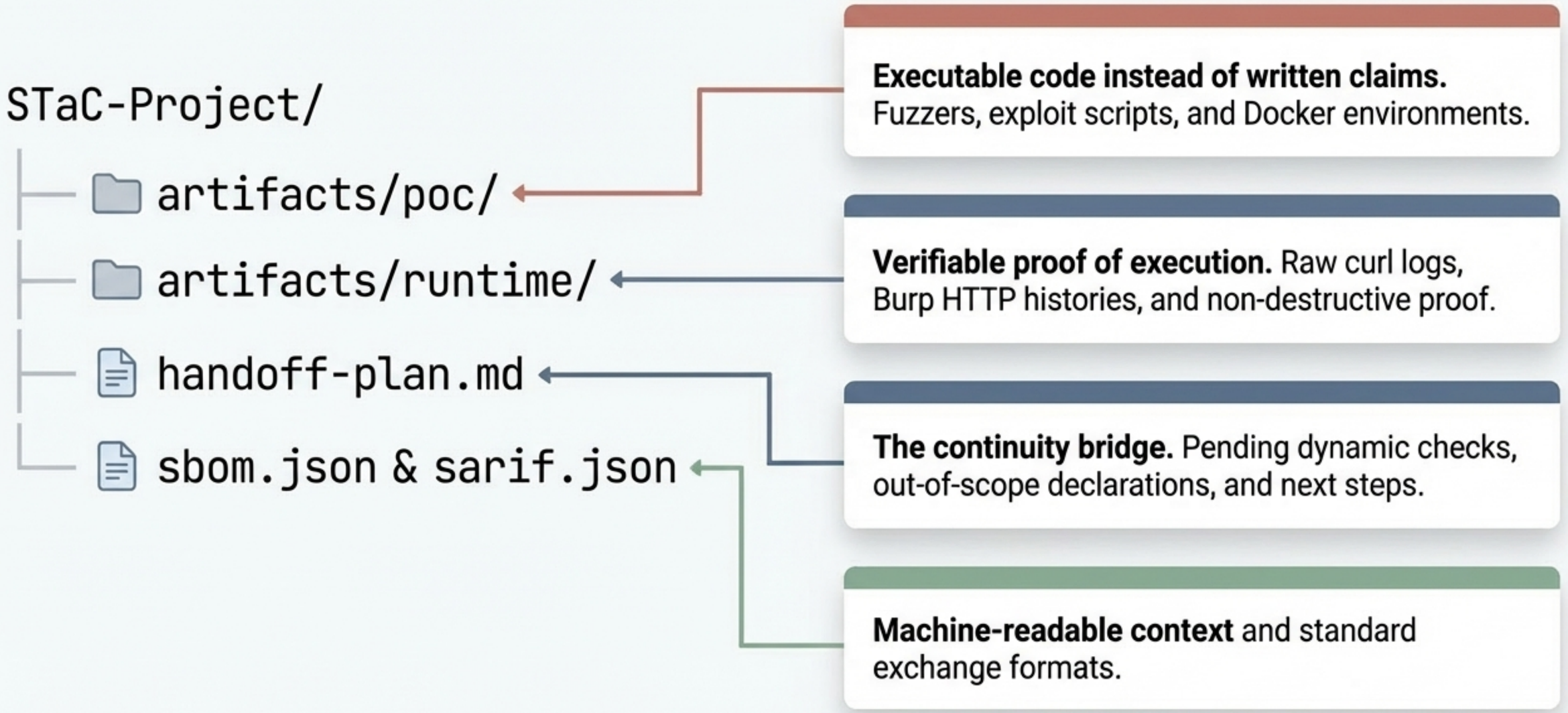
2. Evidence Must Be Reproducible

"I saw it" is insufficient. "Here is the command to see it yourself" is the standard.

3. Reproducibility Requires Version Control

Executable proof demands version-controlled files, tightly coupled to a specific commit hash.

Structuring the Truth: The STaC Repository Blueprint



Case Study 1: Stop Claiming, Start Executing

The XSSStringSerializer Vulnerability

The Static Report

The XSSStringSerializer may unescape HTML inputs, potentially nullifying XSS defenses.

AMBIGUOUS

The STaC Approach

artifacts/poc/fuzz-xss/

```
# Jazzer Java Fuzzer
$ jazzer --agent_path=./build/libs/jazzer.jar \
  --target_class=com.example.fuzzer.XSSSerializerTarget \
  --target_args="--output=./artifacts/poc/fuzz-xss/crashes/" \
  --corpus=./artifacts/poc/fuzz-xss/corpus/"
```

```
[Fuzzer Engine] 26 payloads executed. 7 crash files generated.
```

Anyone can clone this repository and **run one command to reproduce** the crash. **No guesswork required.**

Proving Logic Flaws by Capturing Context

Context

`@Acl Validation Bypass in Shopping Tab Batch Controller`

With HttpServletRequest

`HTTP 500 / ACL_DENIED: allowedIP=..., realIP=192.168.2.1`

✓ **Result:** IP Verification executed and logged.

Missing HttpServletRequest

`HTTP 500 / UndeclaredThrowableException: unknown`

✗ **Result:** IP Verification bypassed due to unhandled exception.

Instead of writing "design mismatch," STaC delivers a Spring Boot Docker container proving the exact condition where the IP check fails. The proof survives refactoring.

'Verified' Requires Proof, Not Just Trust

Verifying a Redis open port risk without performing destructive authentication attempts.

artifacts/runtime/

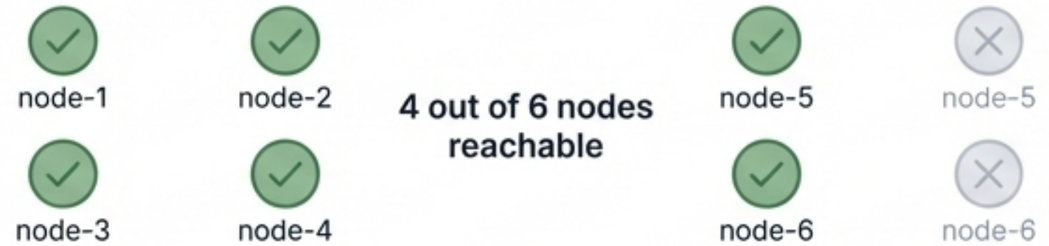
The Action

```
curl -X GET http://internal-redis.svc.cluster.
 local:6379 -v > /dev/null 2>&1
```

The Proof

```
HTTP/1.1 200 OK\r\n
Date: Mon, 27 Oct 2023 10:00:00 GMT\r\n
Server: Redis\r\n
...spring.redis.password is unset...\r\n
Content-Length: ...
```

The Environment State



‘I verified it’ is an opinion. A saved HTTP response header tied to a specific commit hash is undeniable evidence.

Breaking the Amnesia Cycle with Structured Handoffs

`Commit: a1b2c3d — Ensures 100% historical accuracy even 6 months later.`

Verified Code (Phase A)

Explicitly links to executed Jazzer fuzzers and curl logs.

Pending Dynamic Checks (Phase B)

Outlines specific prerequisites and expected outputs for the next auditor.

Out of Scope (Phase C)

Explicitly lists skipped endpoints with architectural justification.

The next tester doesn’t start from scratch; they resume execution from the exact point the previous tester paused.

AI Enables the Projectification of Security

Input: Human

Unstructured Cognitive Analysis (mental models, quick notes).

Processing: LLM Engine

Backed by TestPilot research on LLM test generation.

Executable Test Scenarios

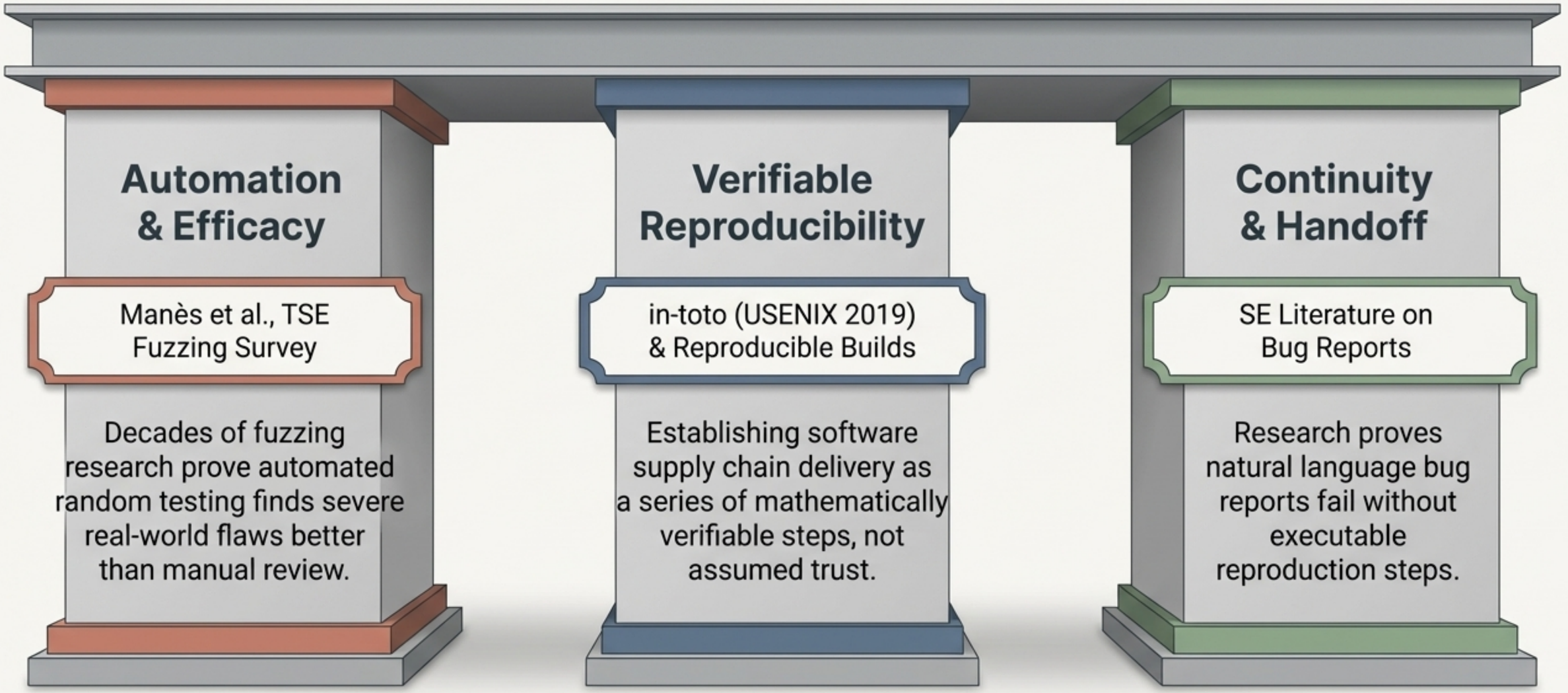
Jazzer/Docker configs

Directory Structures

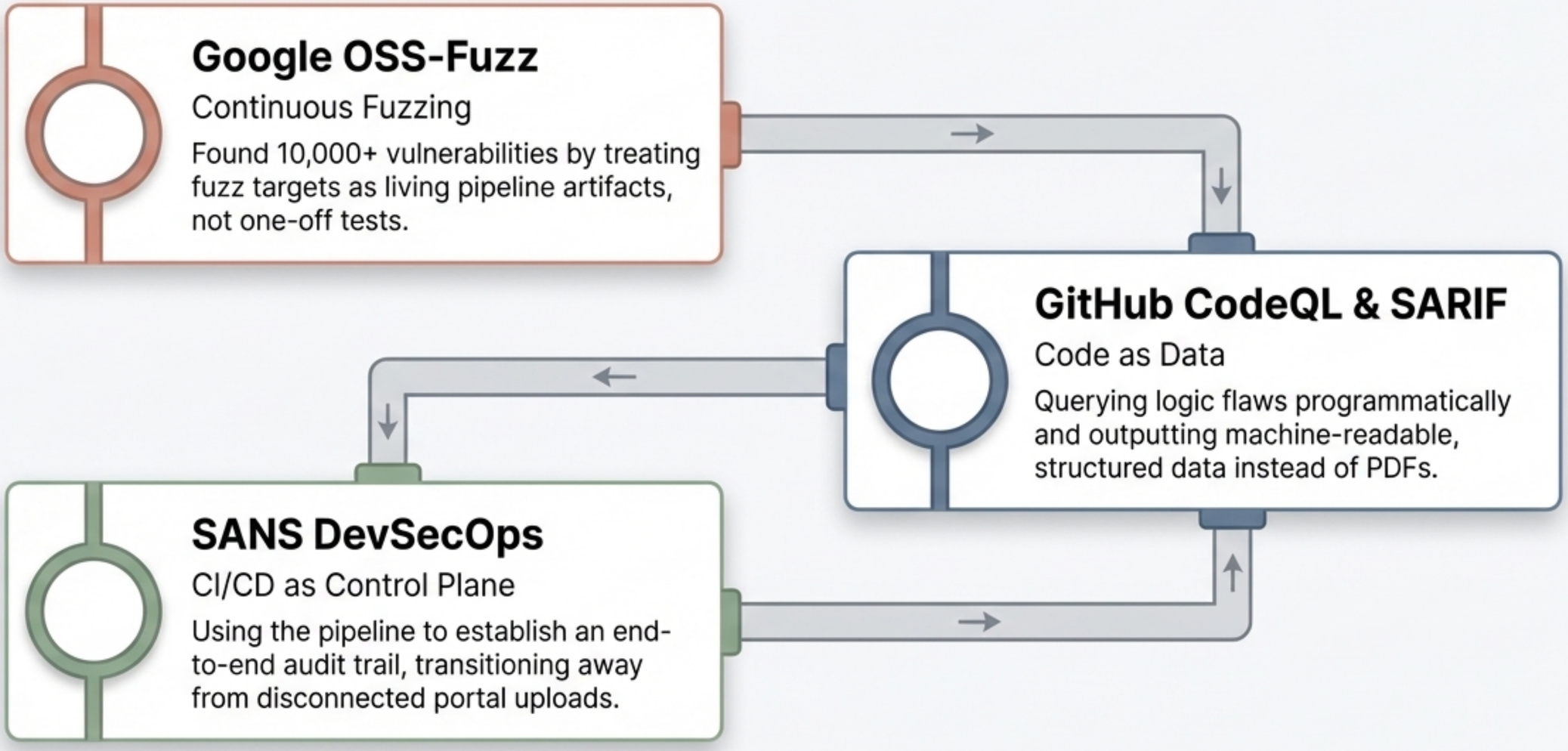
Markdown handoffs

Verification Callout:
Crucial Guardrail: AI hallucination is mitigated because the output must compile and execute. Code execution provides the ultimate ground truth, unlike static AI-generated prose.

Rooted in Decades of Software Engineering Science



Validated by Ecosystem Leaders



Constructing Your STaC Evidence Bundle

Continuity: Handoff Plan & Scope (Markdown/Git)

Integrity: Provenance (SLSA attestations, SBOMs via SPDX/CycloneDX)

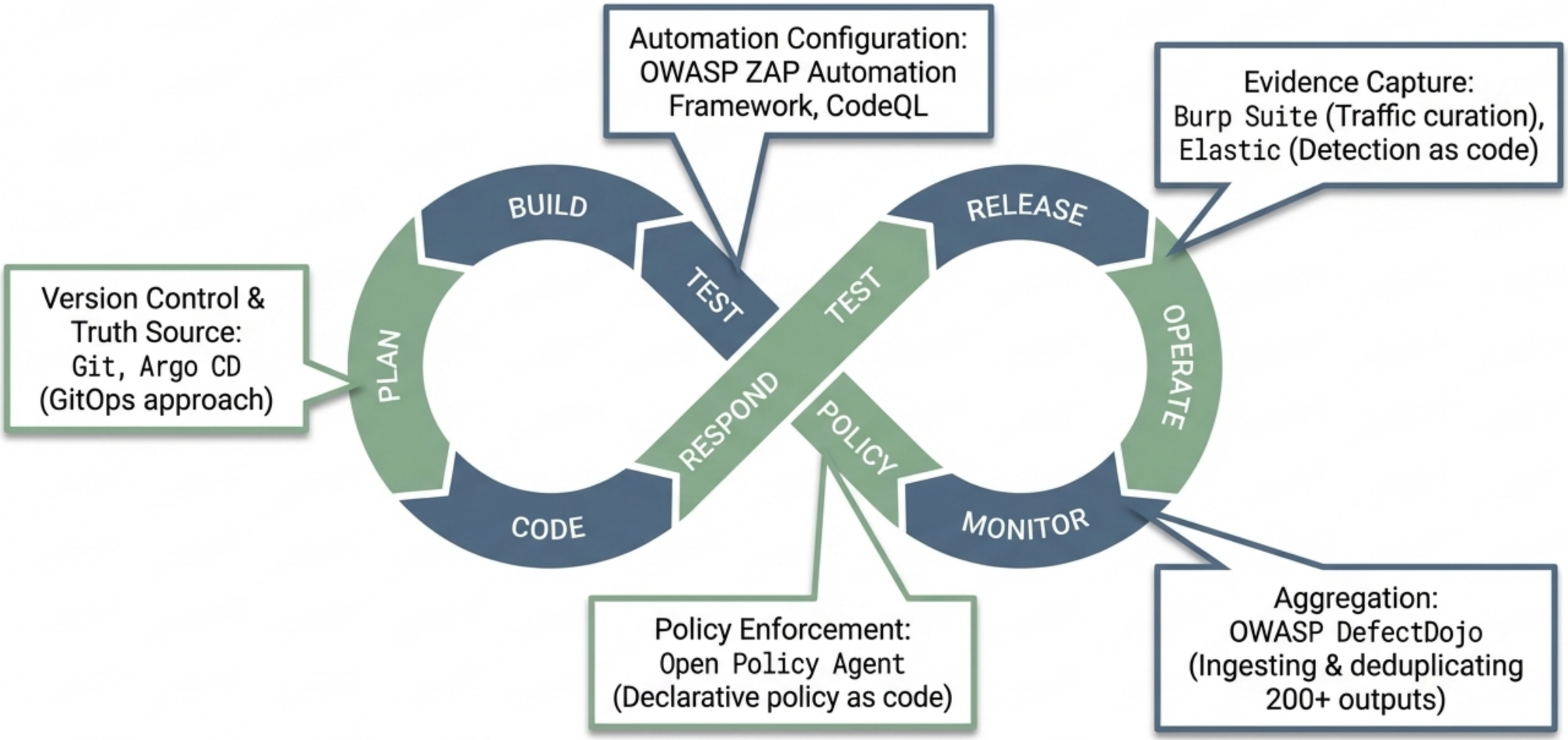
Aggregation: Machine-Readable Results (SARIF, JUnit XML, JSON)

Execution: Automated Checks (SAST, DAST, Jazzer Fuzzing)

Evidence: Runtime Captures (Burp Suite HTTP History, Raw curl logs)

Foundation: Reproducibility & State (Docker, IaC, Fixed Git Commits)

The Modern STaC Toolchain



Documents are Read. Code Executes.

Security Testing as Code is not a buzzword. It is the application of proven engineering principles—version control, reproducibility, and automation—to security.