A scanner gives you findings. A pentester gives you a plan.

websec-validator reads a repository the way a senior pentester reads it on day one, then hands your AI coding agent the plan: what the attack surface is, which findings are real, and exactly what to test. No model inside, no server, no running app.

THE PROBLEM

AI agents now write the code. Nobody briefs them on what to attack. Static scanners answer “which patterns match?” and bury the answer under volume. A pentest answers “what matters, and how would I prove it?” and happens once a year, if at all.

A REAL COUNT, FROM THIS PROJECT'S OWN CHANGELOG (0.17.0, BUG-311)

```
findings-ledger.json   total: 163            # what the scanners said
cluster_summary        distinct issues: 13   # what a reviewer actually has to fix
```

Same fixture, same run. The **163** is not wrong, and every one is kept for the machine consumers. But a human, or an agent, has to be told they are **13 problems**, where they live, and what would confirm each one. That handoff is the product.

FIG. 1 — THE SAME LEDGER, UNDER TWO DIFFERENT QUESTIONS. THE SECOND ONE IS WHAT A PENTEST REPORT CONTAINS.

THE METHOD

code in

Read the repository. Map every route, auth decision, tenant boundary, sink, secret and dependency. Run the installed static scanners, de-duplicate them, and rank what is left by whether it is reachable and exploited in the wild.

artifacts out

A fact file, a findings ledger, a SARIF report, a probe library tailored to what was found, and a briefing for the agent that will do the fixing. Then an exit code that says whether the picture is *complete*.

The rule that surprises people: there is no model in the tool. Same repository, same output, every run. The model sits on the *other* side of the briefing, with a human beside it.

A tool that never says “incomplete” is a tool you cannot trust when it says “clean.”

THE DESIGN CONSTRAINT EVERYTHING ELSE FOLLOWS FROM

EVERY AMBIGUITY ROUNDS TOWARD “INCOMPLETE”

Most security tools have two outputs: findings, or a green check. This one has a third, and it is what makes the other two believable.

```
WHAT THIS TOOL SAID ABOUT A DELIBERATELY VULNERABLE ELIXIR APP, UNTIL SEPTEMBER 2026

# raw SQL interpolation · System.cmd injection · a hardcoded credential · no auth plug
unsupported: []   gaps: []
REQUESTED CHECKS COMPLETED — 0 files read
```

A suffix list had been absorbed into another, leaving the “language I cannot read” branch reachable for one language. The fix does not make the tool read Elixir. It makes it **say** it did not: the banner leads with `NO ANALYZABLE SOURCE`, and `--require-analyzed` turns that into exit 3.

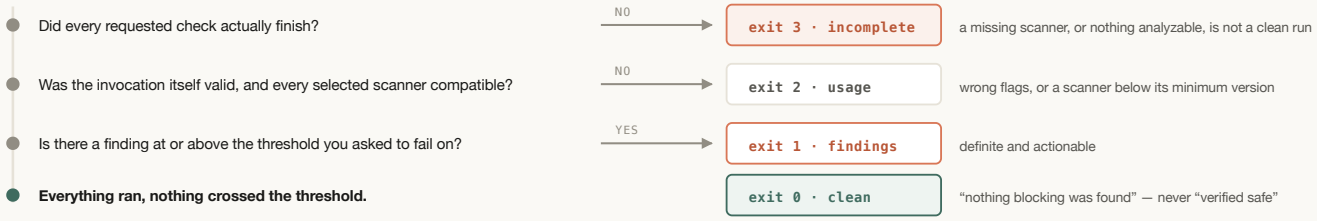

FIG. 2 — THE EXIT-CODE CONTRACT. INCOMPLETE OUTRANKS CLEAN, SO A CHECK THAT SILENTLY DID NOT RUN CANNOT PRODUCE A GREEN BUILD.

NINE THINGS THE TOOL WILL NOT DO, AND WHERE THAT IS ENFORCED

REFUSAL	WHY, AND WHERE
say “compliant”	The word appears in no output, and a test asserts that, so it cannot come back by accident. tests/test_attest.py
claim complete protection	`protection_complete` is always false: static analysis cannot see a running system. docs-canonical/SECURITY.md
fill an evidence gap	The audit projection has 12 control rows; 5 carry an empty evidence list, with the reason. attest.py
count “unknown” as “false”	The old rule scored every unmatched finding as a false positive. Relabelling against pinned source left 21 reviewed labels and **35 findings still unknown**, excluded from both sides. calibration.json
optimize its own score	Anything fitted from labels uses a strictly proper rule, Brier or log score, never accuracy or F1, which reward confident wrongness. It is printed, never optimized. calibration.SCORING_RULE
run a scanner unasked	TruffleHog verifies secrets against live services, so it runs only when named. scanners.py
read a cell too thin to trust	A per-class probability needs at least five samples. One class reaches that today; the rest fall back a tier, and the output names which. calibration.MIN_N
guess reachability	A vulnerable dependency is tagged `imported`, `no-import-found` or `n/a`. It never rounds `n/a` to safe. enrichment.py
claim a benchmark it has not run	The competitor comparison exists as a written protocol. Until it runs, no head-to-head claim is made. BENCHMARKS.md

THE ONE PLACE IT FAILS OPEN, ON PURPOSE

The in-editor hook that checks an agent's edit **exits 0 when it cannot run**, and says on stderr that the edit was not checked: a check that blocks every edit when broken gets uninstalled, and then there is no check at all. It refuses an environment-variable escape hatch, because the agent is what sets environment variables.

ONE REPOSITORY, START TO BRIEFING

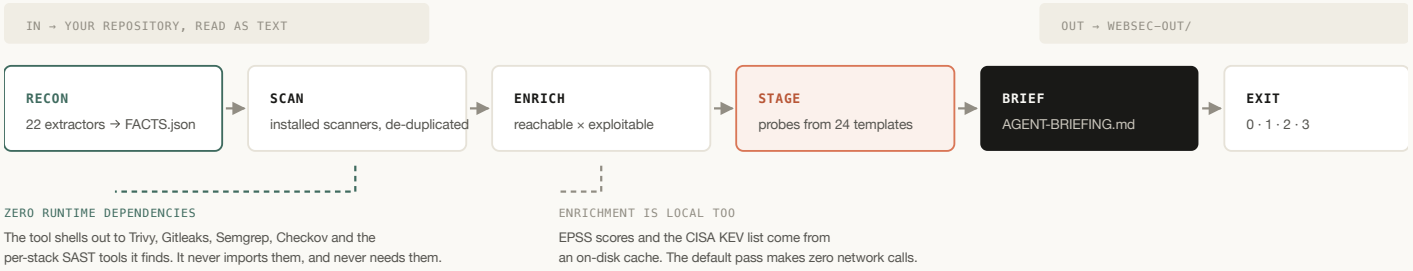

FIG. 3 — CODE IN, ARTIFACTS OUT. THE DASHED LINE IS THE WHOLE SAFETY MODEL: NOTHING ABOVE IT CAN TOUCH A RUNNING SYSTEM.

HOW A FINDING EARNS ITS SEVERITY

Severity is a rule, not a score. The same evidence yields the same label every run, and the rule is written in the methodology before it is written in the code.

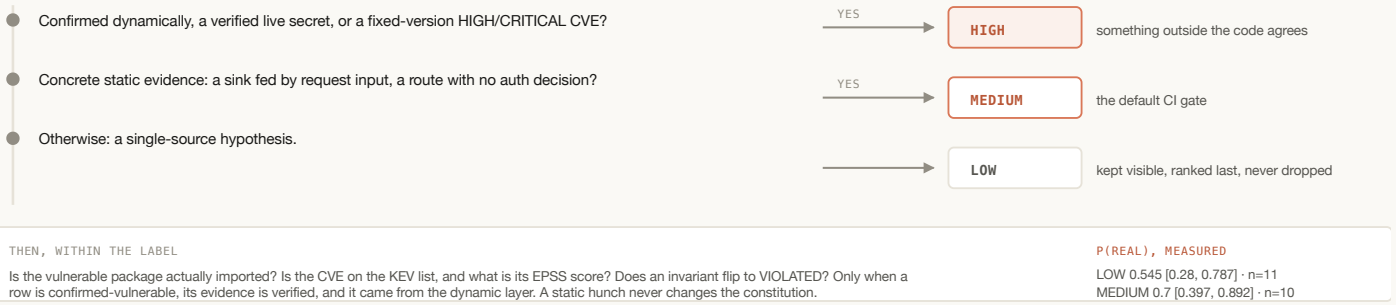

FIG. 4 — SEVERITY ORDER. A HIGH ALWAYS HAS SOMETHING OUTSIDE THE STATIC ANALYSIS AGREEING WITH IT.

WHAT LANDS IN WEBSEC-OUT/

ARTIFACT	WHAT IT IS
AGENT-BRIEFING.md	**The product.** Detected surface, the access-control map, targeting, findings, the method, and the staged probe list, written for the agent that will act on it.
FACTS.json	The full structured recon: every fact the 22 extractors produced, schema-versioned, with a coverage block that says what was skipped and why.
findings-ledger.json	The traceable ledger. Each finding carries its evidence chain, CWE / ASVS / OWASP-API citation, remediation, and a calibrated probability with its interval and sample size. `REPORT.md` is the same ledger for humans.
results.sarif	SARIF 2.1.0, always written. GitHub code scanning, GitLab, Azure DevOps and the VS Code SARIF viewer read it as-is.
probes/	Scripts staged from the 24 templates, pre-filled with the endpoints the recon found, for a human to review and run against a test instance.

TWENTY-TWO EXTRACTORS, SEVENTEEN SINK CLASSES, ONE FACT FILE

Each extractor answers one question a pentester asks on day one, and writes the answer into `FACTS.json` as data, never prose.

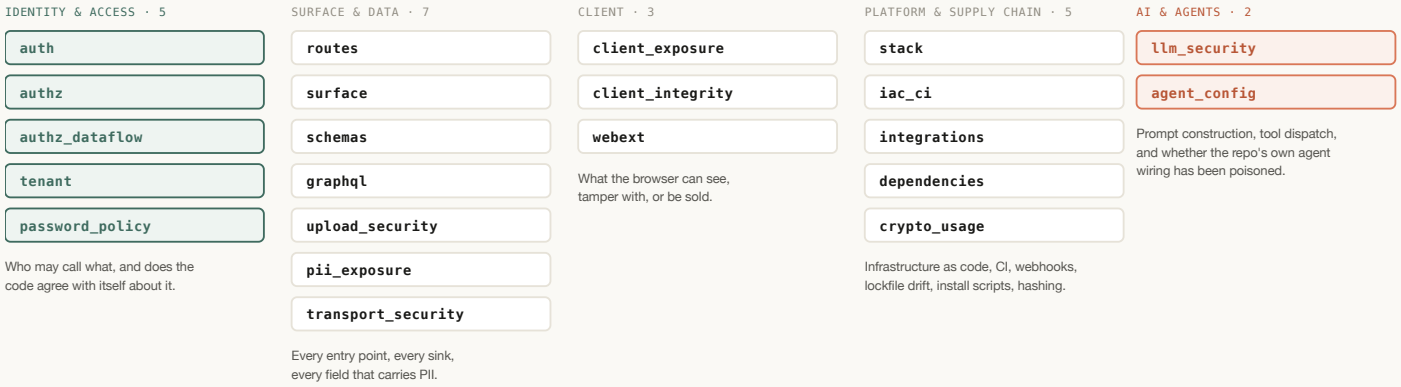

FIG. 5 — THE 22 EXTRACTORS. GROUPS ARE EDITORIAL; THE REGISTRY IS FLAT. EACH IS ISOLATED: ONE THAT CRASHES RECORDS AN ERROR IN COVERAGE AND THE RUN CONTINUES.

THE SEVENTEEN SINKS: REQUEST INPUT REACHING SOMETHING DANGEROUS, TRACED PER ROUTE

`ssrf` `command-injection` `sql-injection` `nosql-injection` `path-traversal` `ssti` `open-redirect` `insecure-deserialization` `xxe` `prototype-pollution` `mass-assignment` `redos` `eval-injection` `ssrf-outbound-http` `error-disclosure` `xss` `log-injection`

Why these twenty-two

Not designed at a whiteboard. Three rows exist because dogfooding on a large LLM-agent monorepo, verified by a 15-agent adversarial pass, exposed two classes the route-and-auth model could not see. Four more grew because a real authenticated pen test found its two Criticals and two Highs in files the recon never parsed: AWS-CDK auth config, AppSync GraphQL schema, VTL resolvers. The walker reads those now.

Mapped to what a reviewer will ask about

Every finding carries a CWE, an ASVS 4.0.3 reference and, where they apply, the OWASP API and OWASP LLM Top 10 entries. The methodology publishes the map as a 41-row table; the findings module cites 77 distinct CWEs. Two limits are stated up front: regex over CDK TypeScript is not an AST and can be evaded by aliasing, and man-in-the-browser can never be a confident static catch, so it ships as LOW.

THE SCANNERS IT SHELLS OUT TO, AND THE VERSION EACH ONE MUST BE

Presence on PATH is not compatibility, so each entry carries the lowest version whose CLI matches the invocation the tool builds, and `doctor` reports it.

SCANNER	CATEGORY · MINIMUM
Trivy	vulnerabilities, secrets, misconfig · 0.38
Gitleaks	secrets in history · 8.0; plus a working-tree pass
Semgrep/OpenGrep	SAST, with two shipped rules · 1.0
Checkov	infrastructure as code · 2.0
OSV-Scanner	dependency advisories · 2.0
Prowler	cloud posture · detect only

PER STACK	RUNS ONLY WHEN THE LANGUAGE IS PRESENT
Bandit	Python · 1.7
gosec	Go · 2.0
Brakeman	Ruby · 4.0
TruffleHog	live secret verification · opt-in only, never implicit

SIX PUBLIC REPOSITORIES, 116 LEADS, ZERO VERIFIED CRITICAL ISSUES

That is the headline, and it is the right one. A tool that reported a critical in every public project it touched would be a tool that mislabels. The review ran source-only against pinned revisions, executed nothing, sent nothing upstream, and wrote down what it could not conclude.

FIG. 6 — ONE DOT PER LEAD. COUNTS ARE DETECTOR OUTPUT AT A PINNED REVISION; THE REVIEW SAYS SO ON ITS FIRST LINE.

116	0	3 → 10
leads emitted across six repositories, none executed, nothing sent upstream	verified critical issues. Written as the headline, not buried in a footnote	false-positive classes adjudicated with source links, each turned into paired regression tests
DOCS/SECURITY-REVIEW/PUBLIC-REPOSITORY-REVIEW.MD	SAME DOCUMENT, FIRST PARAGRAPH	TESTS/TEST_PUBLIC_PRECISION.PY

What the false positives taught

- An archive error helper that formats ZIP messages was read as an **authentication decoder** because its strings said “unauthorized.”
- A local file stream piped to stdout was read as **HTTP file serving** because `createReadStream` alone counted as serving.
- A numeric GitHub expression in a workflow was read as **script injection**.

Each fix ships with its opposite: the test that makes the tool quieter is paired with one that proves the real case, the JWT decoded without verification, the stream piped to a response, still fires. The review also records its own gap: `@angular/core` was not recognized in the framework inventory.

The corpus, pinned

Three deliberately vulnerable apps at fixed commits, retrieved 2026-09-12: VAmPI `f16052d`, NodeGoat `c5cb68a`, DVGA `a961308`. The proof harness passes **10/10** surface checks with Noir present and **8/10** without it, with 7 truth labels left unknown. Both numbers are committed as JSON with the detector revision hash, and the sentences that quote them are bound to those JSON fields by DocGuard, so the prose cannot outrun the artifact.

In September 2026 those 56 corpus findings were relabelled one at a time against the pinned source, each with a note and the revision it was read at. That produced **21 reviewed labels** and a measured probability per severity, replacing a prior that had been guessed. The other **35 stay unknown** and are counted on neither side.

And one real pentest: an authenticated engagement whose two Criticals and two Highs sat in the managed-cloud boundary the recon did not yet parse. Those findings became extractor rows, and two of the retest lessons corrected the tool's own earlier output.

THE SUITE IS GUARDED THE WAY THE TOOL GUARDS YOU

Five checks gate the main branch. Two of them exist because of a specific, recorded failure.

TESTS · PY3.11 · PY3.12 · PY3.13

Install the wheel, byte-compile the package, smoke the CLI entry point, run the suite.

THREE REQUIRED CHECKS

HERMETICITY · HOSTILE GIT CONFIG

A global hooksPath, gpgsign, excludesFile and a renamed default branch. The suite must still pass, and sentinel hooks outside the temp repos must checksum clean.

SUITE-INTEGRITY

Derived test floor · 60 s ceiling · required-check names cross-checked against the jobs this run emits · unit tests for the bot that merges without a human.

WHY THE FLOOR HAS NO NUMBER IN IT

A stored MIN_TESTS was bumped 1241 → 1243 by two branches at once. Both merged cleanly, because the values matched, and the floor ended two below the real suite: two tests deletable unnoticed. So the baseline is now counted from the base commit at CI time, with the unittest loader, in a separate process per tree. A module that fails to import is a hard failure, not a count of one.

WHY SIXTY SECONDS

The suite runs in about ten. The known failure in this class was a 7.6× blowup that stayed green throughout. Sixty catches that shape and tolerates a slow runner.

FIG. 7 — THE GATE STACK. THE NAMES IN REQUIRED-CHECKS.JSON ARE THE SINGLE SOURCE OF TRUTH; A REQUIRED CHECK THAT NEVER RUNS WOULD BLOCK EVERY PR FOREVER.

1,549

tests across 80 files, stdlib only, no public network, at the time of writing

TESTS/ · COUNTED BY THE UNITTEST LOADER

0

runtime dependencies. The tool shells out to scanners and never imports them

PYPROJECT.TOML · DEPENDENCIES = []

5

required checks, cross-verified against the jobs the run actually emits

.GITHUB/REQUIRED-CHECKS.JSON

Docs are checked like code

DocGuard runs with 27 of its 29 validators on, pinned explicitly rather than inherited. Four canonical documents are review-gated. Five sentences in the validation record are bound to JSON pointers inside the committed proof artifacts, so a number in prose cannot differ from the number in the file. Every conscious deviation from those documents is logged in a drift file with its reason.

This brief is held to the same standard. Every count on these pages, from the 22 extractors to the 116 leads, is asserted against the source tree by `tests/test_explained_brief.py`. If the code changes and the page does not, the suite fails.

A release train that stops for a human

Every Monday a workflow proposes a version bump as a pull request and stops. A tag is cut only when a commit whose subject starts with `release:` lands on main, never from a version diff. Publishing to PyPI uses Trusted Publishing over OIDC, with no long-lived token, and refuses if the tag does not match the package version. The triage bot that labels and merges bot PRs never checks out the PR's code, and its classifier has its own unit tests in the required checks.

EVERY FIX CITES ITS BUG, AND EVERY QUIETING TEST HAS A LOUD TWIN

A field report from 2026-09-18 listed nine problems. They became **50 tests** in one file, and the fixes carry the bug number as a comment at the line that changed, in the CLI, the scanner adapters, the findings module, the coverage contract and the dynamic phase, so a reader can walk from the report to the code and back.

The test file's own docstring states the rule: every “this should be quieter” assertion is paired with a “this must stay loud” one. A precision fix that is not paired with the true-positive it must preserve is not accepted. That is the mechanical form of the no-regression bar this project holds itself to.

WHAT IS NOT NEW, AND WHO SOLVED WHICH PART

Static analysis, secret detection, dependency scanning and attack-surface mapping are all solved problems with excellent open-source tools. Anyone claiming otherwise is selling something. This tool **uses** them and adds the layer none of them ship: the plan.

The scanners

Semgrep, Trivy, Gitleaks, Checkov, OSV-Scanner, Prowler and the per-stack SAST tools: Bandit, gosec, Brakeman. Eleven registry entries, each with a minimum version that `doctor` checks, because presence on PATH is not compatibility. The tool shells out, parses, de-duplicates, and cites each finding back to the scanner that produced it.

Attack-surface detection

OWASP Noir finds routes in a codebase better than a fallback regex ever will. When it is installed the proof harness scores 10/10; when it is absent, 8/10. Both numbers are published, because a user without Noir deserves to know what they are missing.

Dynamic scanners

ZAP, Nuclei and sqlmap rediscover at runtime, expensively, much of what is visible in the source. So the briefing **aims** them: each static finding is mapped to the scanner signature it will produce, and the classes no scanner finds, BOLA, missing auth, mass assignment, row-level-security gaps, are listed as blind spots. A clean DAST run is not “safe,” and the briefing says why.

WHAT IS DIFFERENT: THE QUESTION BEING ASKED

A SCANNER	WEBSEC-VALIDATOR
Matches patterns, reports all of them	**Assembles a pentester's handoff**
`“Which lines look dangerous?”`	`“What would I test first, against which endpoint, and what would confirm it?”`
→ 163 findings, one severity each	**→ 13 issues, each with a location, a phase, an oracle, and a probe file**
A list. Every entry may be true. Nothing says which one a pentester would open first, what request would prove it, or which findings are the same bug seen from six files.	A briefing. Phase 1 safe recon, Phase 2 authorization with two identities, Phase 3 injection fired only at sink-backed endpoints and gated behind an explicit warning.

ILLUSTRATIVE BRIEFING ITEM — THE SHAPE, NOT A REAL FINDING

§5b · Phase 2 · authz · two identities

`GET /api/orgs/{id}/invoices ← route with no per-object ownership check (authz_dataflow)`

Oracle: authenticate as tenant B, request tenant A's id. A 200 with a body is BOLA. A 403 or an empty 200 is not.

PROBES/BOLA-CROSS-TENANT.SH · CONFIRM/DISCONFIRM IN ONE REQUEST · A SCANNER CANNOT SEE THIS CLASS

FIG. 8 — A FINDING KEEPS ITS NAME, ITS ENDPOINT AND ITS ORACLE. THAT IS THE DIFFERENCE BETWEEN A METRIC AND A PLAN.

WHAT IT DELIBERATELY IS NOT

- **Not a scanner.** It runs the ones you have installed and would be poorer without them.
- **Not a SaaS.** No server, no account, no telemetry, and zero runtime dependencies. It runs where the code is.
- **Not a DAST.** The shipped dynamic phase is read-only and GET-only by default; write probes need an explicit flag and a localhost target.
- **Not a compliance tool.** No verdict, no score, no badge, and no signed attestation, because a websec-signed attestation would only attest that websec ran.
- **Not an autonomous fixer.** It never edits your repository. The agent proposes fixes; a human approves them.

ONE ENGINE, SEVEN SURFACES

SURFACE	WHAT IT GIVES YOU
CLI	`websec run .` is the whole pipeline. `gate` is the fast scoped pass on the files you just changed; `attest` projects a run into an audit-evidence table; `feedback` records that a finding is wrong, metadata-only by default, because a hardcoded-secret finding's snippet *is* the secret.
SARIF 2.1.0	Written on every run and importable back in, so GitHub code scanning, IDEs and other tools read the same ledger.
GitHub Action	Composite action with a `fail-on` threshold, a baseline for legacy repositories, and SARIF upload. Every third-party action it calls is pinned by commit SHA. It installs the engine from its own checkout, never from the target repository.
Docker	Everything bundled on a slim Python 3.14 image, running as a non-root user, with Noir 1.0.0, Gitleaks 8.30.1 and Trivy 0.74.0 pinned by version.
MCP server	Four read-only tools, `websec_recon`, `websec_findings`, `websec_sarif` and `websec_briefing`, over stdio or loopback-only HTTP that requires a token. Non-loopback bindings are rejected before listening.
pre-commit	One hook id, defaulting to `--require-complete --fail-on high`, so a partial run cannot pass the commit.
Claude Code plugin	A skill that runs the security pass, and a `PostToolUse` hook that checks each edit as it lands, failing open and loudly.

SIXTEEN WEEKS, TWENTY-EIGHT RELEASES, ONE BREAKING CHANGE

First public line. PyPI via Trusted Publishing from day one.
0.3 walks AWS-CDK, AppSync SDL and VTL: the managed-cloud boundary a real pentest exposed.
0.7 adds the LLM / agent extractor. 0.10 adds SARIF output and the CI gate. 0.11 adds MCP over HTTP and the benchmark protocol.
0.12 fixes the hermeticity bug and adds branch protection. 0.14 ships schema 2.0. 0.16 adds the in-loop gate and attest. 0.17 splits the exit codes: the one breaking change.
0.18 relabels the corpus for a measured P(real), and makes a scan that read nothing report as NO ANALYZABLE SOURCE rather than as complete.

FIG. 9 — RELEASE CADENCE. TWO OF THE LAST THREE RELEASES CORRECTED WHAT THE TOOL SAYS ABOUT ITSELF RATHER THAN WHAT IT DETECTS.

Code in, briefing out. Everything it could not check, it says so.

PIP INSTALL WEBSEC-VALIDATOR · GITHUB.COM/RACCIOLY/WEBSEC-VALIDATOR · MIT

Every number on these pages carries a path in the repository and is asserted by the test suite against the source tree at the version in the masthead. The field review, the proof artifacts, the calibration table and the CI workflows are all committed and readable. If you find a claim here that the code does not back, that is a bug: please open an issue.