Security is not a CVE list

Guide · Watchdog **Version 1.0 - 2026-08-03** Author: Canine Development

Who this is for: engineers, security leads and platform owners who already use dependency-vulnerability scanning and want to understand what a repository-level security assessment adds. This guide explains what Watchdog measures on the security side, how it operates alongside existing tools, where its scope ends, and how to use the result.

A CVE list tells you which dependencies map to known vulnerability advisories. That is useful, but narrow. It does not show whether secrets remain in git history, whether dependencies are deprecated or unmaintained, or whether committed infrastructure configuration applies basic hardening controls.

1. A CVE list is inventory, not posture

A dependency-vulnerability report reconstructs your dependency graph, checks it against advisory data and returns findings such as:

`CVE-2024-xxxx` · high severity · `some-lib` · upgrade to 3.4.1

That is an important inventory of known-vulnerable versions at a particular point in time. It does not, by itself, establish the broader security condition of the repository.

It will not tell you that an AWS key was committed to `appsettings.json`, removed in the next commit and left in repository history. It will not tell you that a third of your dependencies are two major versions behind or that one is unmaintained. It will not tell you that a Dockerfile runs as root, a Kubernetes workload has no egress restriction, or a release workflow contains no signing configuration.

None of those conditions is represented by a CVE. Each is relevant to the part of security posture that repository evidence can establish.

Watchdog reads the repository tree, working directory, git history, CI configuration and infrastructure manifests. It evaluates that material through the same scored and re-runnable model used by the rest of the Watchdog survey.

2. What Watchdog measures on the security side

Security & Compliance is one of ten lenses in the Watchdog model and one of five that are always enabled. Dimension codes are included below where the current published catalogue documents them, allowing readers

to inspect the relevant specification at cai.canine.dev/dimensions.

Secrets

Watchdog uses two dimensions because they answer different questions.

The native in-process scanner (`D13`) always runs. A clean result therefore means that the scanner ran and found no match, rather than that the repository was not examined.

The history scanner (`D28`) runs `gitleaks` across the full git history and the current working tree. The history pass detects credentials that were committed and later deleted but remain reachable in repository history. The working-tree pass detects plaintext secrets in tracked configuration. Findings are merged and de-duplicated by rule, file and line.

Neither result proves that no secret exists. No scanner can establish complete absence.

Dependencies and known vulnerabilities

Dependency hygiene (`D12`) covers outdated, vulnerable and deprecated packages. Each package is counted once at its most serious applicable condition, so a package that is both outdated and vulnerable is not penalised repeatedly for the same underlying dependency.

The scoring distinguishes between risk conditions. The existence of a newer version is informational, while vulnerable and deprecated packages carry substantially more weight.

Known-vulnerability analysis then follows the relevant ecosystem:

- .NET uses `dotnet list package --vulnerable` (`D30`).
- JavaScript and npm ecosystems use lockfile analysis across npm, yarn, pnpm and bun (`D33`).
- Other supported ecosystems use the OSV database (`D38`).

Watchdog therefore includes the conventional CVE view, but does not limit the assessment to it.

Code analysis

Static application security testing (`D29`) runs `semgrep` across the repository using pinned OWASP and security-audit rule packs.

The rule packs and their versions are fixed for the run. Under the same recorded inputs, the same commit is expected to produce the same findings.

Infrastructure and runtime hardening

Infrastructure-as-code and container analysis (`D31`) runs `trivy config` across Dockerfiles, Terraform, Kubernetes, Helm and CloudFormation. Misconfiguration severity is mapped into the Watchdog scoring model.

Watchdog also inspects committed runtime-hardening configuration that a general misconfiguration scan may not fully represent. This includes:

- Kubernetes `NetworkPolicy` configuration that restricts egress.
- Seccomp profiles.
- Mandatory-access-control configuration such as AppArmor or SELinux.

These checks establish the presence of committed configuration. They do not prove that the configuration is enforced in a deployed environment.

A repository without relevant infrastructure manifests is marked **not applicable**. It is not awarded a perfect result for having nothing available to assess.

Supply-chain integrity

Provenance and signing (`D36`) performs deterministic file inspection for four integrity signals:

- Generated build provenance.
- Artifact signing through cosign, sigstore or gitsign.
- A software bill of materials.
- Pinned build actions rather than floating references such as `@main` .

Governance evidence

Licence compliance (`D14`), vulnerability-disclosure policy (`D37`), and data-compliance and PII signals (`D32`) appear within the Security & Compliance and Readiness lenses according to the Watchdog model.

These are governance indicators rather than direct security controls. A licence conflict, for example, is not a vulnerability.

Language coverage

Many of these checks inspect files and configuration rather than application-language semantics, so they can apply regardless of the language used by the application.

For dimensions that do inspect application code, analysis depth varies. The current public language-support endpoint lists 13 languages: 12 with Deep analysis and JavaScript with Structural analysis. The current tiers and sign-off dates are published at watchdog.canine.dev/api/public/language-support.

3. It brings its own scanners

Watchdog invokes `semgrep` , `trivy` , `gitleaks` and OSV as part of its own analysis and normalises their output into the same scored model as the other dimensions.

These tools ship in the analyzer image. A customer does not need to install them for the survey to run.

Existing SAST, SCA, CI and linting tools remain separate. Watchdog runs alongside them and does not replace their role in development and release workflows.

Two boundaries are important.

Watchdog does not incorporate external scanner findings into its score

When a SARIF file from another tool is supplied, Watchdog uses finding counts for calibration comparison. The external findings are not stored as Watchdog findings and do not contribute to the index.

Turning an external scanner on or off therefore does not alter the Watchdog score.

A scanner that does not run is not treated as a pass

When a required tool cannot run, the relevant dimension is marked **not measured**, with the reason recorded. It is not silently converted into a passing result.

A survey result therefore contains an actual reading for each dimension that could be measured, while unavailable or inapplicable dimensions are identified explicitly.

4. Where the scope ends

Repository evidence, without deployment probing

Watchdog analyses code, configuration, history and manifests. It does not attack a running deployment, fuzz endpoints or probe an application from the internet.

Outside-in DAST and penetration testing are separate disciplines and should be used where those forms of assurance are required.

Repository evidence evaluated against external intelligence

A repository cannot state whether one of its dependencies has become vulnerable. Advisory databases and OSV provide that external intelligence.

This is why a vulnerability result can change even when the repository itself has not changed.

Committed configuration is not proven runtime enforcement

Watchdog can see that a pipeline is configured to generate provenance, pin actions or sign artifacts. It cannot establish from repository evidence alone that the deployment platform enforces those controls at runtime.

The result therefore records configuration evidence rather than claiming an operationally proven pass.

Evidence, not a compliance certificate

Watchdog produces evidence that can be reviewed by an auditor or a customer's security team. It does not make an organisation "SOC 2 ready" or "GDPR compliant".

No repository scanner can grant certification. The value lies in producing checkable evidence with recorded inputs and repeatable evaluation rules.

Reproducing a survey and verifying a delivery are different checks

Survey reproduction concerns whether Watchdog can evaluate the same commit under the same recorded rubric, analyzer image and run parameters and obtain the same score. Each run records the commit, timestamp, rubric version and analyzer image needed to identify those inputs.

Delivery verification concerns a result package shared with another party. The package is content-hashed and Ed25519-signed, allowing the recipient to verify its origin and confirm that its contents were not changed in transit.

The first procedure describes reproducibility within the Watchdog survey environment. The second is independently available to the recipient of a shared package. Both are documented at cai.canine.dev/verify.

5. The daily security-watch

Full surveys run on the cadence chosen by the team. Vulnerability information and repository contents can change between those surveys, so every Watchdog package includes a daily security-watch for tracked repositories. It is enabled by default.

Once per calendar day, at the repository's scheduled UTC slot, the watch re-runs the Security & Compliance lens and no other lens. It does not perform the architecture analysis, compiler loading or enrichment included in a full survey.

The daily security-watch is exempt from scan quota and is not billed.

Some security-adjacent dimensions are not included because they belong to the Readiness lens. Dependency hygiene (`D12`), native secret scanning (`D13`) and licence compliance (`D14`) therefore wait for the next full survey.

Secrets remain covered by the daily watch through the history-and-working-tree scan (`D28`).

The daily watch shortens two different detection windows:

- A newly published vulnerability can change the risk status of an existing dependency even when the repository has not changed.
- A secret committed between full surveys can be detected by the next scheduled daily run.

Watchdog does not monitor credentials leaked through channels outside the repository.

Results appear on the repository when the watch completes. Notifications are sent only when the team has configured a notification rule for the security-watch trigger.

6. How to use the result

1. **Open the Security & Compliance lens before relying on the headline score.** The aggregate index answers a broader question. The lens shows which security-related dimensions failed, were not measured or were not applicable, together with the recorded reason.

1. **Prioritise exposed credentials.** A history finding means the credential may remain in clones that contain that history. Rotate the credential first to limit further exposure. Then remove it from repository history and follow the organisation's incident-response procedure.

1. **Distinguish vulnerabilities from routine version lag.** Triage vulnerable and deprecated packages first. Merely outdated packages can usually be grouped into planned maintenance work unless other evidence raises their priority.

1. **Keep the daily security-watch enabled.** Add a notification rule when the team needs active delivery rather than relying on someone to check the repository result.

1. **Use the evidence when assurance is requested.** Recorded run parameters support reproduction within Watchdog. A signed delivery package allows an external recipient to verify who issued it and whether it was altered. Those are more useful assurance properties than an unexplained score alone.

Learn more and verify

- **Published dimensions and scoring.** The public catalogue documents the code, lens and evaluator for each published dimension: cai.canine.dev/dimensions. The scoring specification explains how lenses contribute to the aggregate index: cai.canine.dev/spec.
- **A complete security-lens example.** The public report gallery contains reports from real repositories, including their full Security & Compliance sections, without requiring an account.
- **Noise and accuracy.** Watchdog's design target is below 5% noise, on a deliberately broad definition: a finding counts as noise if it is a false positive, states opinion as fact, duplicates another finding, or is irrelevant to the shape of the code. That is a bar we hold ourselves to, not a measured result - and it is not uniform. Each language is calibrated separately, so some sit well below the bar today and others well above it; the ones added most recently have had the least of that work. The measurement protocol and judge prompt are public at github.com/CanineCC/watchdog-benchmark.

- **Industry context: secrets.** GitGuardian's *State of Secrets Sprawl 2026* reports 28.65 million new hardcoded secrets detected in public GitHub commits during 2025, a 34% year-over-year increase. It also reports that internal repositories were approximately six times more likely than public repositories to contain a hardcoded secret.
- **Industry context: open-source vulnerabilities.** Black Duck's *Open Source Security & Risk Analysis 2026* covers 947 commercial codebases across 17 industries. It reports that 87% contained at least one known open-source vulnerability, 78% contained a high-risk vulnerability and 44% contained a critical-risk issue. The mean number of vulnerabilities per codebase increased by 107% to 581.