Never Clean Without Evidence: Design and Evaluation of the openness Credentialed Vulnerability Scanner

The openness Team
S4E.io (Security For Everyone OÜ)
Contact: Onur Aktaş (onur@s4e.io), Hacı Özdemir (haci@s4e.io), Eren Akdağ (eren@s4e.io)

Abstract

Credentialed host scanners decide whether a server is vulnerable by comparing installed package versions with vendor advisories. Their most damaging failure is not a false positive but a false all-clear: a report with no findings for a host that was never actually evaluated, because the package database was unreadable, the distribution was not recognised, the inventory was truncated, or no vendor rules were loaded. We describe openness, a credentialed, open source, agentless scanner for Linux and Windows (for now) servers that is built around one invariant: a host is never reported clean without evidence that it was evaluated. openness separates observation from judgement. A short-lived collector, uploaded over SSH or WinRM and removed afterwards, writes a self-describing evidence file whose every module records its status, truncation and errors. A deterministic resolver matches that evidence against vendor data (Ubuntu OVAL, the Debian Security Tracker, Alpine secdb, the OSV exports of Red Hat, Rocky Linux, AlmaLinux and openSUSE, Amazon Linux updateinfo, Microsoft CVRF, NVD and OSV PyPI) using ports of the native version comparators, and emits a typed coverage report alongside its findings. Vulnerability data reaches the desktop through a mirror whose manifests are signed with Ed25519 under a domain-separation prefix and protected against rollback by a monotonic sequence number. We describe each data source and each comparator in enough detail to be reimplemented, define the finding states and the coverage gap taxonomy, and report what we have measured: 5,478 version-comparison pairs labelled by native `dpkg` and `apk` with zero disagreements, 91 rpm test vectors, the unit and end-to-end test suites, and end-to-end runs in eight distribution container images, where the packages openness reports as having a pending security fix agree with the distributions' own package managers on 47 of 49 packages, with every disagreement explained. We are explicit about what we have not measured, notably detection accuracy against a ground truth at scale, and list it as future work.

1 Introduction

Vulnerability management on servers rests on a deceptively simple question: *is the software on this host affected by a known vulnerability?* For software installed from a Linux distribution the answer cannot be read from upstream version numbers. Distributions backport security fixes into old upstream versions, so `openssl 3.0.2-0ubuntu1.15` on Ubuntu 22.04 may be fixed for a CVE that upstream fixed only in 3.0.7. The authoritative answer is the distribution's own advisory data, compared with the installed version using the distribution's own version ordering. Network scanners, which infer versions from banners, cannot see backports and cannot see software that does not listen on a port. Credentialed scanners log in, read the package database and compare it with vendor data; this is the approach of local checks in Nessus [65], the Notus scanner in Greenbone OpenVAS [30], Vuls [26], and, for container images, Trivy [8] and Grype [5].

A credentialed scanner can fail in two directions. A false positive costs an engineer time. A *false all-clear*, a report with no findings for a host that was not actually evaluated, costs much more: it is indistinguishable from a well-patched host, it ends the investigation, and it is often produced silently. In our experience building openness, the causes are mundane: a package database locked by a concurrent `apt` run, an unrecognised derivative distribution, a container image without `/etc/os-release`, a release for which no advisory feed was downloaded, a module that timed out, an inventory cut at a size limit, or an unprivileged login that could read nothing. Each of these yields *zero findings* unless the scanner models the absence of evidence explicitly.

This paper describes openness, a credentialed, agentless vulnerability scanner for Linux and Windows servers (on Windows, the operating system build and catalogued third-party applications), whose design is organised around the following invariant:

> *A host is never reported as clean unless there is positive evidence that its inventory was read and*

that vendor rules for its platform were loaded and evaluated.

The invariant concerns the *evaluation*, not the completeness of the vendor data: a host whose packages were all evaluated against current rules is covered even if some packages have no advisories at all. Section 9 states precisely what a covered result does and does not mean. We make the following contributions.

- **An evidence model that separates observation from judgement** (§4). A collector that makes no vulnerability decisions writes a typed evidence file in which every module reports a status, every limit that was hit is recorded as truncation, and unknown values are `null` rather than defaults.
- **A deterministic resolver with explicit states** (§5 to §7). We document how each vendor feed is parsed and normalised into binary-specific rules, how each native version comparator is ported and validated, and how findings are assigned one of seven states, including `unknown` and `conflicting_evidence`.
- **A coverage-gap taxonomy** (§9) that turns the common ways an evaluation can be incomplete into typed, machine-readable gaps, and the rule that finding counts are never an all-clear on their own.
- **A signed vulnerability-data supply chain** (§11) with domain-separated Ed25519 signatures, monotonic sequence numbers against rollback, per-file hashes, version pinning and verifiable offline bundles.
- **An honest evaluation** (§13) that reports only measured numbers and states which properties remain unmeasured.

The collector and the resolver are open source under the Apache License 2.0; the desktop application and the backend services are proprietary. Section 17 lists what is available.

2 Background and Threat Model

2.1 Network versus credentialed scanning

A *network* (unauthenticated) scanner probes open ports and infers software and version from protocol behaviour and banners. It needs no credentials and sees the host as an attacker on the network would, but it has three structural blind spots for patch assessment: it cannot see software that does not listen, banners report upstream versions that do not reflect distribution backports, and many services hide their version.

A *credentialed* (authenticated) scanner logs in, enumerates installed packages and applications, and compares them with vendor advisories. It sees every installed package and the exact distribution version string, at the cost of handling credentials and executing code on the target. Credentialed scanning can be *agent-based* (a resident service, as in Qualys Cloud Agent [52] or Wazuh [70]) or *agentless* (the scanner logs in for each scan). openness is agentless: it uploads a collector for the duration of a scan and removes it afterwards (§4.1). openness also performs credential-free network checks, but this paper focuses on the credentialed path.

2.2 Why vendor data, not upstream versions

The NVD [41] describes affected software with CPE names and upstream version ranges [16, 49]. For distribution packages this is the wrong frame of reference: Debian, Ubuntu, Red Hat, SUSE and Alpine backport fixes and publish their own fixed versions per release, in their own version syntax, sometimes per binary package. A scanner that matches distribution packages against NVD ranges produces both false positives (fixed backports) and false negatives (vendor-specific packaging). openness therefore uses distribution data as the sole authority for distribution packages, and uses NVD only for software that no distribution describes: third-party Windows applications and network devices (§8). The resolver encodes this as a source precedence in which distribution vendors outrank OSV, which outranks NVD-derived data.

2.3 Threat model and goals

Assets. The scanning workstation holds credentials (SSH passwords and keys, sudo passwords, WinRM and SNMPv3 secrets) for many servers, and the scan results describe their weaknesses.

Adversaries. We consider: (A1) a *compromised scanned host* that controls everything it returns, including the evidence file, and may try to inject markup, formula or shell payloads into the reports and remediation commands, or exhaust resources. Such a host also receives what the scan presents to it: the SSH password or key authentication, the sudo password on standard input (in `su` mode, the root password), and an NTLM exchange on WinRM, which a malicious host can relay or attack offline (a lab-only setting even uses Basic authentication over plain HTTP); and it can report false package versions, which no credentialed scanner can detect; (A2) a *network attacker* between the desktop and its vendor services or between the backend and the upstream publishers, who may tamper with or replay vulnerability data; (A3) a *malfunctioning or truncated upstream*, for example a publisher serving a partial file. We do not defend against a compromised scanning workstation, a malicious vendor or a malicious upstream publisher that signs or serves wrong advisories; the scanner trusts the publishers' content. Credentials should therefore be dedicated scan accounts with the least privilege that still allows a complete inventory.

Goals. **G1, correctness by construction**: decisions follow vendor data and native version order, deterministically, with a recorded reason. **G2, honesty**: absence of evidence is reported as such, never as a clean result. **G3, small footprint**: the target is read, not changed, and nothing of the scanner remains

on it. **G4, data integrity**: vulnerability data on the desktop is exactly what the backend published, and comes from a manifest no older (by sequence number) than the last one the client accepted. **G5, confidentiality**: credentials, evidence and findings stay on the scanning workstation unless the user opts into a feature that sends a documented, redacted subset.

3 Architecture

Figure 1 shows the four components.

Desktop daemon and UI. `opennessd` is a single Go service that runs on the analyst's workstation (Windows, macOS or Linux), installed as a Windows service, a launchd agent or a systemd unit. It serves an embedded React user interface over HTTPS on `127.0.0.1:8835` by default, with a self-signed ECDSA P-256 certificate created at first run; a non-loopback listen address is refused unless explicitly enabled. Local users authenticate with bcrypt-hashed passwords; sessions use `HttpOnly`, `Secure` (except in an explicit development mode over plain HTTP), `SameSite=Strict` cookies and every state-changing request must carry a custom header as a cross-site request forgery guard. State is kept in a pure-Go SQLite database with WAL journaling. Stored secrets are sealed with AES-256-GCM under a per-installation random 32-byte key in a file of mode `0600`; the data directory is forced to mode `0700`.

The scan engine runs each target as a job of named steps: `network` (credential-free port and service checks), `connect`, `host_key`, `upload`, `verify_collector`, `privilege`, `collect`, `fetch_evidence`, `cleanup`, `resolve_wait`, `resolve` and `store`. Steps and log lines stream to the UI over Server-Sent Events. By default three servers are scanned in parallel and two resolver runs may execute concurrently; a job waiting for a resolver slot releases its server slot.

Collector. A static Go binary (Linux amd64/arm64, Windows amd64) that reads the machine it runs on and writes one evidence file (§4). It contains no SSH or WinRM client, no vulnerability logic and no network code apart from an optional upload and token-check client (HTTPS by default) that openness does not use.

Resolver. A Go program that reads an evidence file and a local rule store and writes `findings.json` (§5 to §9). It never touches the network while resolving; feeds are fetched by a separate `feed sync` command or imported from the mirror. The desktop runs it as a subprocess with a per-job deadline. The resolver contains no machine-learning component; every decision is produced by rules, version arithmetic and explicit logic.

Flow	Content	Default
License	key or activation secret, e-mail (Free sign-up), hashed fingerprint, nonce, product, version	on use
Data mirror	signed license document; downloads only	check 6 h
AI	redacted structured payload	off
Usage statistics	per job: type, hour, duration, result, error code; license document	on
Unknown apps	per app: name, publisher, version, uninstall key, product code, install folder (user name masked), machine count; license document	off
App updates	GET of a signed release manifest (public code-hosting site)	daily
Publisher fallback	direct feed downloads from publishers	off

Table 1: Outbound flows from the desktop. Calls to vendor services carry the signed license document. All flows are disabled by offline mode.

Backend. Vendor services for licensing (Ed25519-signed license documents), the vulnerability-data mirror (§11), the application catalog build (§8), the opt-in AI proxy (§12), and minimal telemetry. Public and administrative APIs are served on separate ports so that a proxy misconfiguration cannot expose the administrative interface.

3.1 Data flows and trust boundaries

Table 1 lists every flow that leaves the workstation. Scan targets' addresses and host names, credentials, evidence files and findings are never sent to the vendor; the single exception is the redacted, structured AI payload, which is sent only when the user enables AI (§12). Usage statistics are on by default and contain, per finished scan job, only the event type, the time rounded to the hour, the duration, the result and an error code, wrapped in an envelope with a random installation identifier and the application version. Like every call to the vendor services, the request is authenticated with the signed license document, so the vendor can attribute it to a license (customer, e-mail address, license identifier and hashed machine fingerprint).

Offline mode. A single setting blocks every outbound vendor call: update checks, mirror synchronisation, license calls, AI, unknown-application reports and usage statistics. User-initiated calls return an explicit `offline_mode` error rather than timing out. For air-gapped installations a license is obtained through a request code that encodes the machine fingerprint, a nonce and a CRC-32, carried to a connected machine and answered with a license file valid for the whole term; vulnerability data arrives as a signed offline bundle (§11).

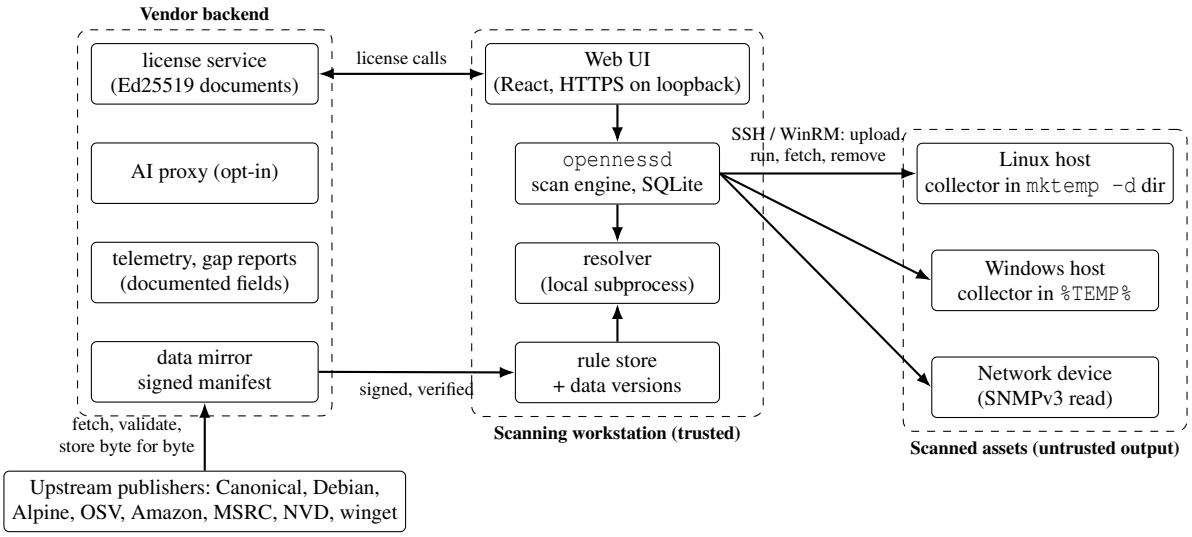


Figure 1: openness architecture. Arrows show the main data flows; all calls to vendor services are made by `opennessd`. Scanned assets are treated as untrusted; vendor services are contacted only for licensing, signed vulnerability data and opt-in features, and not at all in offline mode.

4 The Collector and the Evidence Model

4.1 Delivery, execution and removal

SSH. The desktop authenticates with a password, a private key (optionally with passphrase) or a private key with an OpenSSH user certificate. Host keys are pinned on first contact (trust on first use); a changed key fails the job with `host_key_mismatch`, and once a key is pinned the negotiated host-key algorithms are restricted to its type so that a server cannot downgrade to a different key. The remote system must report `Linux` from `uname -sm`; the architecture selects the collector build. The collector is uploaded with SFTP into a fresh `mktemp -d` directory under `/tmp`, `/var/tmp`, `/dev/shm` or a private directory in the user's home, trying the next location when a file system is mounted `noexec` or read-only.

Privilege is established in order: user id 0, then passwordless `sudo -n`, then `sudo -S` with the password supplied on standard input (never in argv or logs); an optional `su` mode is also supported. Under sudo the collector is copied into a root-owned `0700` directory and its SHA-256 is compared with the local copy before it runs. It is started in its own process group (`setsid`, when the target has it) with a collector timeout (default 900 s) plus 60 s grace. Without elevation the collector still runs, and the result carries a "partial inventory" notice.

The evidence file is fetched over SFTP, or with `sudo cat` when it is root-owned, and refused above 64 MiB. Cleanup always runs on a fresh 60 s context (45 s on WinRM), reconnecting if needed. If collection did not complete, the process group is sent `TERM`, then `KILL`, but only after checking that `/proc/<pid>/cmdline` still names the uploaded collector, so a recycled PID is never signalled. Both work directories are then removed. A later scan also sweeps directories left by an interrupted earlier scan, with an age margin, owner check and a cap of 20 per scan.

WinRM. The desktop uses NTLM with message encryption over HTTP (port 5985) by default, or NTLM over TLS (5986). On HTTPS the server certificate is pinned on first use rather than validated against a CA, because WinRM listeners commonly use self-signed certificates. The collector is gzip-compressed (an 8 MB binary becomes about 3.3 MB) and sent as base64 fragments by four parallel uploaders into `%TEMP%\s4e-openness-<guid>`, then reassembled in PowerShell. The `verify_collector` step runs the binary's `version` command. WinRM sessions do not elevate; a token outside the Administrators group yields a partial-inventory notice. Cleanup stops the process and removes the directory.

4.2 Modules

The collector runs a fixed set of modules per operating system, 15 on Linux and 17 on Windows (Table 2). Each module runs in its own goroutine under a per-module deadline (default 120 s) inside a global deadline (default 900 s). Its design rules, stated in its design documentation, are: observe and do not judge (the collector never says "vulnerable"); unknown is `null`, not `false`; a missing hop in a chain is an explicit gap; permission failures are explicit, never empty success; and caps are honest (hitting a limit marks the module partial).

The collector reads package databases directly where possible. On Linux it streams `dpkg-query` output and falls back to parsing `/var/lib/dpkg/status`; it parses Alpine's `/lib/apk/db/installed` without running `apk`; and it runs `rpm`

Module	What is read (Linux / Windows)
os_inventory	os-release, machine id, kernel release, reboot-required marker / CIM and registry build, UBR, edition, reboot-pending flags
packages	Linux: dpkg status (incl. distroless status.d), rpm database, apk installed; APT lists for highest available version and origin
applications	Windows: Uninstall keys (64/32-bit machine keys and the running user's key), MSI, AppX, browser and runtime adapters
cbs_packages, patches	Windows: servicing packages, hotfixes
missing_updates	Windows Update Agent search, WSUS policy, supersedence
kernel_inventory	Linux: running kernel, loaded modules
containers	Linux: docker, podman, nerdctl (up to 40 per runtime): images, digests, os-release, package databases; containerd: identifiers only
language_deps	Python, Node, Java, Go artifacts reached from running processes
evidence_chain	socket, PID, executable, service, package, source package, repository
others	hardware, services, processes, listening ports, loaded libraries, users, network, security configuration, software inventory

Table 2: Principal collector modules.

-qa only when a system rpm database exists, because rpm would otherwise create a database in the user's home, a write the collector must not cause. Containers are inspected without being started or entered: package databases and os-release are read from the host-visible root file system, or copied out with docker cp; os-release is read only if it is a regular file of at most 64 KiB, following at most one symbolic link that stays inside the container root. On Windows, data comes from CIM, the registry and PowerShell (version 3 or later). Command lines are redacted of password- and token-like arguments and credential URLs before they are stored.

Side effects. Apart from the evidence file and short-lived temporary files, the collector does not write to the target. Two Windows modules deserve mention: missing_updates asks the Windows Update Agent for applicable updates without forcing offline mode, so the agent may contact the host's configured update service; and applications runs dotnet -list-runtimes when dotnet is present.

4.3 The evidence file

The evidence file (extension .openness, schema 1.6.0) is a JSON document, optionally gzip-compressed. Its envelope holds schema_version, collector (name, version, build id), collection (identifier, start and end time, duration, execution_mode, privilege_level, status, partial), asset (stable asset id, host name, machine id, platform, architecture), modules, aggregated errors and integrity. Inventories merged from modules (operating_system, packages, applications, kernel, containers, language_deps, evidence_chains and others) appear at the top level for convenience. The asset_id is the first 8 bytes of SHA-256 over a fixed prefix, the platform and the lower-cased machine id, so it is stable across runs; it is random when no machine id is available.

Every module result carries module_id, module_version, status (success, partial, error, unavailable, permission_denied), timing, record count, evidence records and errors. Every evidence record carries a collection_status (observed, permission_denied, unavailable, error, skipped, partial). A collection starts as complete; any module that does not end in success turns it partial and copies its errors to the top level. Relationships between entities (for example a binary package and its source package) carry a confidence label, direct, strong or inferred. The validator requires the endpoints of standalone relationship records (used for the Python inventory) to resolve to entities in the file; relationships embedded in package and process records are not cross-checked.

Partial and truncation semantics. Limits are part of the schema. A module that hits a limit does not end in success and records one of the following error codes; the record and size caps also set truncated: true: package_limit_truncated (20,000 packages per host or container, 100,000 across containers), output_size_truncated (the document would exceed 256 MiB, so the module with the most records is halved repeatedly until it fits), timeout (module or global deadline; the evidence collected so far is still written), no_packages (no package was read; the module ends unavailable, never success with an empty list), and container_packages_unread (a container has a package database that could not be read, with a per-container reason such as rpm_timeout). A module abandoned at its deadline ends error without evidence. The resolver maps no_packages, package_limit_truncated, container_packages_unread and a truncated or timed-out packages module to coverage gaps (§9).

Integrity hash. The collector computes SHA-256 over a canonical re-encoding of the document with the integrity field cleared, and stores the hex digest and algorithm name. The canonical form is Go's encoding/json output after a round-trip through a generic map (sorted keys, compact), not RFC 8785 [60]. The resolver recomputes the hash and refuses a mismatching file; a file without a hash is processed with a note. The hash is unkeyed: it detects corruption and accidental edits but not a deliberate forgery, since anyone who can edit the file can recompute it. Authenticity of evidence therefore rests on the channel (an authenticated SSH or WinRM session whose output the desktop fetches directly), and the content remains untrusted under adversary A1.

5 Vulnerability Data Sources

The resolver normalises every source into one rule type, `PackageVulnerabilityRule`, whose identity is *(release, source package, binary package, CVE or advisory)* and which carries a normalised vendor status (`released`, `needed`, `deferred`, `not_affected`, `does_not_exist`, `ignored`, `unknown`), the raw vendor status text, the fixed version, an optional introduced version, a support tier, the architecture, the vendor's own urgency and a source priority. Parsers count every malformed or unusable record with a reason; out-of-scope entries (other ecosystems, debug and source packages, non-fix remediations) are filtered by documented rules. Table 3 summarises the sources.

5.1 OVAL

The language. The Open Vulnerability and Assessment Language (OVAL), created at MITRE, stewarded by the Center for Internet Security from 2015 and now sponsored by the Naval Information Warfare Center (NIWC) Atlantic, is an XML language for describing machine states. Version 5.11.2 was the last release under MITRE and CIS; the community has since published versions 5.12 and 6.0 [9, 47, 48, 66]. An `oval_definitions` document contains five kinds of element. A *definition* has a class (`vulnerability`, `patch`, `inventory`, `compliance` or `miscellaneous`), metadata (title, references such as CVE or advisory identifiers, description) and a *criteria* tree. A criteria node has an `operator` (`AND`, `OR`, `ONE`, `XOR`), an optional `negate` flag, and children that are nested criteria, *criterion* references to tests, or `extend_definition` references to other definitions. A *test* (for example `dpkginfo_test`, `rpminfo_test`, `uname_test`) binds an *object*, which says what to collect (a package named `libssl3`), to zero or more *states*, which say what to compare (an `evr` that is `less than 0:3.0.2-0ubuntu1.15`), together with `check` and `check_existence` attributes that quantify over the collected items. *Variables* (`constant_variable`, `local_variable`, `external_variable`) let an object or state refer to a list of values. An OVAL interpreter evaluates the tree to one of six results: `true`, `false`, `unknown`, `error`, `not evaluated` and `not applicable`.

How vendors publish it. Canonical publishes per-release OVAL documents in two flavours: the USN flavour (`com.ubuntu.<codename>.usn.oval.xml.bz2`), with one `patch` definition per Ubuntu Security Notice, and the CVE flavour, with one `vulnerability` definition per CVE including those not yet fixed; a third, `pkg` flavour and `oci` variants for Ubuntu cloud images used with an image manifest also exist [11]. Red Hat published OVAL (version 2) for RHEL for many years; it is now in maintenance mode and Red Hat recommends its CSAF advisories and VEX documents [55–57]; SUSE publishes OVAL per product and version [63].

How openness parses it. The resolver consumes only Canonical's USN OVAL, the source Canonical recommends for patch assessment. It streams the XML and consumes `patch` definitions, `dpkginfo_test`, `dpkginfo_object`, `dpkginfo_state` and `constant_variable` elements. It does *not* implement general OVAL evaluation. Instead it relies on the regular structure of Ubuntu's documents. Each relevant criterion carries a comment of the form *"openssl package in jammy was vulnerable but has been fixed (note: ...)"*, naming the source package; the parser reads from it the source package, the release with its optional tier prefix (`esm-apps/jammy`), and a status phrase that it normalises ("has been fixed" becomes `released`, "needs fixing" becomes `needed`, and so on, with unknown phrasing preserved and flagged). The binary package names come from the test's object, through its `var_ref` to a `constant_variable`, and the fixed version from the state's `evr` with operation `less than`; any other operation is rejected. One rule is emitted per (CVE, binary package). Criteria are flattened, so `operator` and `negate` are not evaluated; tests other than `dpkginfo` (for example `uname_test`, `textfilecontent54_test`) are counted as unsupported and their criteria rejected; and kernel criteria, whose comments name a kernel flavour, become kernel rules keyed by kernel source family (§7.4). This is a deliberate trade-off: it is simpler and auditable, it preserves the vendor's exact wording in every rule, and its rejection counters make any change in Canonical's document shape visible. Its cost is that it is not a general OVAL interpreter and would not handle documents that encode logic in operators.

We checked that flattening preserves the meaning of the documents it is used on. In the jammy and noble USN OVAL files downloaded on 7 November 2026 (2,638 and 1,587 `patch` definitions), no element is negated. Every definition is an `AND` of the release check, which the resolver applies through the file's release, and either one package criterion or an `OR` of package criteria, which per-criterion evaluation reproduces exactly. The only other `AND` groups pair a kernel criterion with its "Is kernel *X* running?" check (1,778 and 788 groups), which the running-kernel evaluation of §7.4 reproduces, or concern Livepatch (36 and 19 groups), which openness does not evaluate. A future change of shape would show up in the parser's rejection counters, not as silently different results.

5.2 Ubuntu Security Notices and Ubuntu Pro

A USN is Canonical's advisory for a set of package updates in one or more Ubuntu releases [14]. A USN typically fixes several CVEs; a CVE may be fixed by several USNs over time (for example a fix and a later regression fix); and some USNs reference no CVE. In the USN OVAL each definition corresponds to a USN; the resolver takes the USN identifiers from `<reference source="USN">` and the CVEs, with Canonical's per-CVE priority, from the advisory metadata. When no `<cve>` element is present, CVEs are taken from CVE references and

Platform	Source	Format	Rule level	Notes
Ubuntu 20.04, 22.04, 24.04	Canonical USN OVAL [11]	OVAL 5 XML (bzip2)	binary	ESM tiers from pocket; kernel by source family
Debian 12, 13	Debian Security Tracker [22]	JSON	source	compared with source version
Alpine 3.14 to 3.22	Alpine secdb [1]	JSON	origin	applied to all subpackages
RHEL 8, 9	Red Hat OSV export [46]	OSV JSON (zip)	binary	EUS/AUS/E4S/TUS streams skipped
Rocky 8, 9; AlmaLinux 8, 9	OSV exports [46]	OSV JSON (zip)	source / binary	introduced/fixed intervals
openSUSE Leap 15.5, 15.6, 16.0	OSV export [46]	OSV JSON (zip)	source	binaries from `ecosystem_specific`
Amazon Linux 2, 2023	ALAS updateinfo [4]	`updateinfo.xml`	binary	security updates naming a CVE
Windows 10, 11, Server	MSRC CVRF [38]	CVRF JSON	OS build	fixed build and UBR per product
Windows applications, devices	NVD [42] via app-mapper	NVD JSON 2.0, CPE	product	§8
Python (PyPI)	OSV PyPI export [46]	OSV JSON (zip)	package	PEP 440 comparison

Table 3: Vulnerability data sources and the level at which rules are expressed.

the description (except for regression notices); a USN that still has no CVE becomes a rule without a CVE, reported as an `advisory_only` finding with the USN in `advisory_ids`; USN identifiers are never placed in the CVE field.

Ubuntu Pro extends security maintenance through Expanded Security Maintenance (ESM) pockets: `esm-infra` for the main archive beyond standard support and `esm-apps` for the universe archive [10, 12]. Fixes in these pockets appear in the same OVAL documents with a tier prefix, which the resolver stores as the rule's support tier (§7.5).

5.3 Debian Security Tracker

The Debian Security Tracker publishes one JSON document mapping each source package to CVEs and, per release, a status (`resolved`, `open` or `undetermined`), a fixed version, an urgency and an optional `nodsa` reason [22]. The resolver streams it one source package at a time, drops `TEMP-` identifiers, and maps: `resolved` with fixed version `0` to `not_affected`; `resolved` to `released`; `open` with urgency `unimportant` or `end-of-life` to `ignored`; `open` with `nodsa` reason `postponed` to `deferred`; other `open` to `needed`; `undetermined` to `unknown`. Debian rules are source-level, and the engine compares them with the *source* version recorded by dpkg (`Source: name (version)`) when present, so that binNMU or epoch-bearing binary versions are not misread as fixed. Debian 11 is reported as not supported because the tracker no longer publishes it.

5.4 Alpine secdb

Alpine publishes, per branch, `main.json` and `community.json`, in which each origin package has a `secfixes` map from fixed version to CVE identifiers [1]; the version `0` denotes not affected. Rules are expressed on origin (source) packages and applied to every installed subpackage of the same origin. The branch is derived from `VERSION_ID` (3.20.10 maps to `v3.20`).

5.5 RPM distributions and Amazon Linux

For RHEL, Rocky Linux, AlmaLinux and openSUSE the resolver uses the vendors' OSV exports [45, 46], downloaded as `all.zip` per ecosystem, rather than OVAL. OSV records list affected packages with `ECOSYSTEM` ranges of `introduced` and `fixed` events. The parser pairs events into intervals and stores the introduced version, which the engine enforces (an installed version older than `introduced` is not applicable); `last_affected` ranges are rejected as unsupported, and a range without a `fixed` event becomes `needed`. Ecosystem strings select the release (`Rocky Linux:9`, `AlmaLinux:8`, `Red Hat:enterprise_linux:9::<repo>`, `openSUSE:Leap 15.6`). Rocky and openSUSE name source packages, AlmaLinux and Red Hat binary packages; Red Hat extended-support streams (EUS, AUS, E4S, TUS) are skipped. For RHEL 8 module streams, a fix whose release carries a `.module+` tag applies only to modular builds of the same stream.

For Amazon Linux 2 and 2023 the resolver follows the release's mirror list to the core repository's `repomd.xml` and its `updateinfo.xml.gz`, every hop over HTTPS on the same CDN host, and keeps `type="security"` updates of the release's own collection that name a CVE [4]. Rules are binary-level, since Amazon publishes no source package names; debuginfo and source packages are skipped.

5.6 Microsoft MSRC

Microsoft publishes its Security Update Guide data through an API that serves one document per month in the ICASI Common Vulnerability Reporting Framework (CVRF) 1.1 format, in JSON or XML, under the API path `cvrf/v3.0` [32, 38, 39]; Microsoft also publishes CSAF [37, 44]. A CVRF document contains a product tree, vulnerabilities with CVE identifiers, threats (including a severity per product), and remediations. A *Vendor Fix* remediation names a knowledge base (KB) article, the affected product identifiers, whether a restart is required, the KB articles it supersedes (a field Microsoft spells `Supercedence`), and, for Windows, a `FixedBuild` such as `10.0.20348.2849`.

The resolver keeps only Vendor Fix remediations with a `FixedBuild` on products whose CPE is `cpe:2.3:o:microsoft:windows*`; Office, .NET, Edge and other products are counted as skipped. It models the operating system as one synthetic package `windows` with version `<build>.<ubr>`, where the Update Build Revision (UBR) is read from the registry by the collector. Because Windows is serviced cumulatively, comparing UBRs within one servicing branch is sufficient; rules whose fixed build is on another branch are rejected rather than compared. The collector-reported build selects the MSRC product (build 20348 is Windows Server 2022; build 26100 is Windows Server 2025 on a server SKU and Windows 11 24H2 on a client SKU: the two share a build number and receive the same cumulative updates, but are separate MSRC products with their own CVE applicability). The remediation card names the KB with the highest fixed UBR among vulnerable findings, because that cumulative update closes all of them. Supersedence chains are parsed and stored but are not needed for this decision and are not used. MSRC data is retained for 48 months in the mirror.

5.7 NVD, CVE JSON 2.0 and CPE

The National Vulnerability Database publishes CVE records enriched with CVSS scores and applicability statements in the NVD JSON 2.0 format, through the CVE API 2.0 and yearly data feeds [41, 42]. Applicability is expressed as configurations of `cpeMatch` entries, each a CPE 2.3 name with optional bounds `versionStartIncluding`, `versionStartExcluding`, `versionEndIncluding` and `versionEndExcluding`. A CPE 2.3 formatted string `cpe:2.3:part:vendor:product:version:update:edition:language:sw_edition:target_sw:target_hw:other` names a product with `part` `a` (application), `o` (operating system) or `h` (hardware) [16]; name matching is specified in [49], and the CPE Dictionary lists official names with human-readable titles [17]. openness uses NVD only to build the application catalog (§8): the backend downloads the yearly NVD JSON 2.0 feeds from 2002 to the current year, verifying each against its `.meta` file, together with the CPE dictionary. CVSS scores come from this data: application and device findings are rated by it, and distribution findings show it next to the vendor's own rating (§7.3).

5.8 OSV for PyPI, and winget

The OSV schema [45] describes vulnerabilities in open-source ecosystems with an identifier, aliases (for example CVE and GHSA identifiers) and affected package ranges. The resolver downloads the PyPI `all.zip` export, keeps records with a CVE identifier or alias, records the OSV identifier as an advisory, and uses as fixed version the highest `fixed` event across all ranges, compared with the PEP 440 comparator (§6); `introduced` and `last_affected` are ignored, and a record without a `fixed` event becomes `needed`. Python distributions owned by a dpkg package are instead evaluated as that package against Ubuntu data and merged onto the host finding, so the same vulnerability is not reported twice.

The winget package index [36] is an input to the catalog build only: winget package identifiers name catalog entries, and winget packages whose name and publisher match exactly one NVD product are added as unverified entries.

Source	Stated terms
NVD	U.S. government data; use permitted with the NVD non-endorsement notice [43]
CVE records	CVE Terms of Use; copies must reproduce MITRE's notice [19]
Ubuntu OVAL	CC BY-SA 4.0 [13]
Alpine secdb	CC BY-SA 4.0 [2]
PyPA advisory database (PyPI)	CC BY 4.0 [51]
Red Hat security data	CC BY 4.0 [57]
AlmaLinux OSV data	MIT, as listed by OSV.dev [29]
Rocky Linux OSV data	BSD, as listed by OSV.dev [29]
openSUSE OSV export	no data licence found
Debian Security Tracker	no data licence found
Amazon ALAS	no data-specific licence found; general AWS site terms
MSRC CVRF API	no API-specific terms found; general Microsoft terms
winget-pkgs	MIT [36]

Table 4: Terms stated by data publishers (accessed November 2026).

5.9 Licensing of data sources

openness redistributes vulnerability data: the mirror serves upstream files byte for byte, the backend derives processed rule archives, and the catalog is published as a public release. Table 4 lists the terms that the publishers state. Where a publisher states no licence for the data we say so rather than infer one; the absence of a stated licence does not imply permission to redistribute, and general site terms of the publisher may apply. Three obligations follow from the stated terms: attribution for the CC BY and CC BY-SA sources, the NVD notice and the CVE Terms of Use notice; ShareAlike for material derived from the Ubuntu and Alpine data, which covers the processed rule archives; and, for sources without stated terms, a decision based on the publisher's general terms.

6 Version Comparison Semantics

Every decision reduces to *installed* $<$ *fixed* under the right ordering, and each ecosystem defines its own. The resolver registers one comparator per ecosystem and never shells out to native tools at runtime; an ecosystem without a registered comparator produces `unknown`, never a guess. Generic semantic versioning [50] is deliberately not offered for operating-

system packages. Table 5 gives characteristic edge cases.

dpkg. A Debian version is `[epoch:]upstream[-revision]` [21]. Epochs (missing means 0) are compared numerically (the port accepts epochs of any length, whereas dpkg itself rejects epochs above `INT_MAX`); then upstream versions; then revisions, after the last hyphen (missing means `0`, so `1.0` equals `1.0-0`). Within upstream and revision, the algorithm alternates between non-digit and digit runs. Non-digit runs are compared character by character with a modified order in which `~` sorts before everything, including the end of the string, and letters sort before non-letters; digit runs are compared numerically. Thus `1.0~rc1` $<$ `1.0` and `2.30~esm1` $<$ `2.30`. The resolver's implementation is a direct port of `verrevcmp` in dpkg's `lib/dpkg/version.c`.

rpm. An RPM version is `[epoch:]version-release`, compared epoch first, then version, then release, each with `rpmvercmp` [58, 59]. `rpmvercmp` splits strings into maximal alphabetic or numeric segments, ignoring separators; a numeric segment is newer than an alphabetic one; numeric segments compare numerically after leading zeros are stripped; and the longer remaining string wins. Since rpm 4.10, `~` sorts before everything, including the end, and since rpm 4.15 `^` sorts after the end of the string but before any further segment. The port follows `rpmvercmp.c` with one deliberate deviation: if either side lacks a release, releases are not compared. This is rpm's dependency-matching rule, not its version ordering (native rpm orders `1.0` below `1.0-1`); the resolver uses it because some vendor fixed versions carry no release. The deviation does not change a decision for installed packages: the rpm database always records a release, and when only the fixed version lacks one, native ordering (`1.0-1` $>$ `1.0`) and the deviation (equal) both yield `fixed`. It matters only if an installed version without a release were ever compared.

apk. Alpine versions are token sequences: digits, letters, suffixes and an `-r` revision [3]. Pre-release suffixes `_alpha`, `_beta`, `_pre`, `_rc` sort below no suffix; post-release suffixes `_cvs`, `_svn`, `_git`, `_hg`, `_p` sort above. A component with a leading zero is compared as a fraction, so `1.05` $<$ `1.5`. The port follows apk-tools' `version.c`.

PEP 440. Python versions have an epoch (`N!`), a release, and optional pre-release, post-release, development and local segments with a defined total order [18]. The resolver's comparator is a simplified implementation of this order that has not yet been validated against a labelled corpus (§14).

Windows builds. Versions are `<build>.<ubr>`; a `10.0.` prefix is tolerated. Two versions on different servicing branches are not comparable and yield an explicit error, which rejects the rule. Builds that share a servicing stream through enablement packages are grouped (19041 to 19045; 22621 and 22631; 26100 and 26200).

Eco.	a	b	order
dpkg	`1:1.0`	`2.0`	$a > b$ (epoch)
dpkg	`1.0~rc1`	`1.0`	$a < b$
dpkg	`1.0`	`1.0-0`	$a = b$
dpkg	`1.0a`	`1.0+`	$a < b$ (letters first)
rpm	`1.0^git1`	`1.0`	$a > b$
rpm	`1.0^git1`	`1.0.1`	$a < b$
rpm	`1.0`	`1.0-1`	$a = b$ (deviation*)
apk	`1.0_rc1`	`1.0`	$a < b$
apk	`1.05`	`1.5`	$a < b$
apk	`1.0_p1`	`1.0`	$a > b$
win	`20348.2849`	`22621.1`	incomparable

Table 5: Ordering edge cases handled by the comparators. *Native rpm orders $a < b$; see text.

Applications. Version strings of Windows applications are compared part by part: numeric parts numerically, alphabetic parts case-insensitively, missing parts as zero (`24.08` equals `24.08.00.0`), and a number above letters (`1.0` $>$ `1.0beta`).

7 Matching

7.1 Binary and source packages

Distributions build several binary packages from one source package, and advisories are expressed at different levels: Ubuntu USN OVAL and Amazon per binary, Debian and Alpine per source (origin), the OSV exports per binary or source depending on the vendor. The collector records the source package and, for dpkg, the source version, with a confidence label (`direct` when the database names a source, `inferred` when the binary name is used as a fallback). The resolver indexes rules by binary and source name and selects candidates from both indices, subject to one invariant: *a rule that names a binary package applies only to that binary*. Sibling binaries of one source never share a fixed version. This matters in practice: in Ubuntu 22.04, `fonts-opensymbol` is built from the LibreOffice source with version `2:102.12+LibO7.3.6-...`, while `libreoffice-base-core` has version `1:7.3.6-...`; applying one's fixed version to the other would produce nonsense under dpkg ordering. Source-only rules apply by source; Alpine origin rules apply to every subpackage of the origin.

7.2 States and the decision table

Each (package, vulnerability) pair receives one of seven states: `vulnerable`, `fixed`, `not_applicable`, `potentially_vulnerable`, `unknown`, `conflicting_evidence` and `vulnerable_runtime`. A rule first passes applicability

gates (binary, distribution, release, architecture, RPM module stream, OSV introduced version); failing a gate rejects the rule, which is counted in `rules_rejected`, except an architecture mismatch, which is reported as `not_applicable`. Vendor statuses then map as follows: `not_affected` and `does_not_exist` give `not_applicable`; `released` with a fixed version gives `vulnerable` if *installed* $<$ *fixed* and `fixed` otherwise; `released` *without* a fixed version gives `unknown` (the resolver refuses to claim a fix without a fixed version); a comparator error gives `unknown` with the error recorded; `needed` and `deferred` give `vulnerable` with fix availability `unavailable` or `deferred`; `ignored` gives `potentially_vulnerable`; anything else gives `unknown`.

When several rules address the same binary and vulnerability (for example a USN and a later regression USN), the released rule with the highest fixed version wins and the others are recorded as superseded; supersedence never crosses binaries. Without a released rule the most severe outcome wins. If a `not_affected` rule disagrees with a `vulnerable` outcome from a rule of the same source priority, the result is `conflicting_evidence`. Rules are sorted by identifier before tie-breaking, so the same input always produces the same findings; only timestamps and durations differ between runs.

Each finding records the matched rule (identifier, source URL, snapshot, comparator, raw vendor status), the comparison expression and result, the evidence identifiers it rests on, a confidence level (lowered from `high` to `medium` when the source mapping was inferred or the package came from a third-party repository), and human-readable reasons. A finding identifier is a truncated SHA-256 over asset, scope, vulnerability, binary package and installed version, which makes it stable across runs.

Runtime enrichment. If a package is `fixed` on disk but a running process still maps a deleted executable from before the update, the finding becomes `vulnerable_runtime` with the process identifiers and, when known, the systemd unit to restart.

7.3 Severity: vendor rating and CVSS

A CVE has two severities that often disagree. The CVSS base score published by NVD [25] rates the flaw in general, for every product and configuration at once. The rating of the operating-system vendor (Ubuntu priority, Debian urgency, Red Hat and SUSE severity, Microsoft severity) rates the same flaw *as it is built and shipped in that release*: a hardening flag, a disabled feature or a component that is not reachable in the default installation lowers it, and a default exposure raises it. Most scanners count by CVSS. openness counts by the vendor's rating by default, for the reason it uses vendor data everywhere else (§2): the vendor knows its own build.

The severity of a finding is decided in a fixed order:

1. the severity the vendor's rule gives, else the vendor's urgency;
2. if the vendor has not rated the CVE (for example Debian "not yet assigned"), the band of its NVD CVSS score, with a reason that says so;
3. otherwise `unknown`. A finding is never given a severity that no source states.

Findings from the application catalog and from network devices have no distribution vendor, so they carry the NVD severity and score directly.

The CVSS score is shown on every finding whichever source decides the severity. openness ships a table with the NVD score of every CVE; for each CVE it keeps the score of the newest CVSS version NVD has published for it (4.0, then 3.1, 3.0 and 2.0) and records that version next to the score, so a 2.0 score is not read as a 3.1 score. Bands follow the CVSS qualitative scale: 9.0 and above critical, 7.0 high, 4.0 medium, above zero low.

The source is a setting, `vendor` or `cvss`, and can be changed per scan after the scan has finished: the stored resolver output is scored again, no host is contacted. Under `cvss` a finding takes the band of its score, and its reasons keep what the vendor said ("the vendor rates it medium"), so neither answer is lost. A finding without a score (NVD has not scored the CVE, or the finding has no CVE) keeps the vendor's rating, and the scan reports how many such findings there are. A regression fix, which is not a vulnerability, is never rescored. Switching the source changes severities and counts only; states, fixed versions and remediation are the same under both.

7.4 Kernels and reboots

A host may have several kernel images installed but runs exactly one. The resolver evaluates kernel advisories only against the running kernel. On Ubuntu it parses the running release (`5.15.0-91-generic`) into ABI and flavour, finds the installed image or modules package for that ABI and flavour, and maps it to a kernel source family (for example `linux-hwe-6.8`) through an explicit table rather than string-prefix heuristics; meta-packages are excluded. Kernel rules are selected by exact source family and upstream track. If a newer fixed kernel is installed but not booted, the finding stays `vulnerable`, is marked `fixed_image_installed`, and the remediation is a reboot. The `reboot_required` marker from `/var/run/reboot-required` produces a separate informational card. On Ubuntu, a running kernel that is not an Ubuntu kernel (a container's view of the host kernel, WSL, a custom build) yields no kernel findings and an explicit note. On Debian and RPM hosts, installed kernel packages that do not match the running kernel are filtered out; when no installed package matches the running kernel at all, nothing is filtered and the installed kernel packages are evaluated.

7.5 Ubuntu Pro entitlement

A fix whose rule is in an `esm-infra` or `esm-apps` tier is available only to hosts entitled to Ubuntu Pro. The resolver considers a host entitled only if at least one installed package actually came from the ESM archive (by origin); a configured but unused repository does not count. For an unentitled host the finding stays `vulnerable` with fix availability `requires_esm`, and the remediation card is `requires_entitlement: ubuntu_pro` instead of an upgrade command that would silently do nothing.

7.6 Containers

Container findings never merge with host findings. Containers are grouped by image digest, so containers of one image share findings, and the image's `os-release` (from the collector) is mapped through the same distribution and release logic and support matrix as a host. Debian, Ubuntu and RPM-family images are evaluated with the host evaluator of that distribution; kernel rules are skipped. Alpine images take their branch from anchor packages (`alpine-baselayout`, `apk-tools`, `musl`, `busybox`), with confidence lowered to `medium`. A non-Alpine image whose packages cannot be evaluated (no `os-release`, unsupported release, no rules, or an unreadable package database) is a `container_not_covered` gap. For Alpine images, an unknown branch yields an `unknown` finding and a branch without rules a `missing_rules` gap. A container without a `sha256:` image digest, and an image with no package database at all, such as a `scratch` image, are only noted.

7.7 Network devices over SNMPv3

For routers, firewalls and switches the desktop reads identity with SNMPv3 (user-based security model only; authentication SHA-2 family, SHA-1 or MD5; privacy AES-128/192/256 or DES). It reads the MIB-II system group (`sysDescr`, `sysObjectID` and others), the ENTITY-MIB physical table for the chassis model and software, firmware and hardware revisions, and vendor version objects for MikroTik, Fortinet and Palo Alto when ENTITY-MIB is empty. Weak configurations are themselves findings (`authNoPriv`, MD5, DES). The desktop writes a small device file without host name, address or serial number and runs the resolver on it; device entries of the application catalog (Cisco IOS XE, IOS, NX-OS, ASA, FortiOS, Junos, RouterOS, PAN-OS, AOS-CX, EdgeOS) are matched against it, and NVD ranges decide. This path is not yet active end to end: all device entries are unreviewed, so matching requires the user to enable unverified matches, and the catalog bundle built by the backend currently excludes device entries by default until the resolver shipped with the desktop reads them. A device without matches is reported as not evaluated rather than clean.

8 The Application Catalog

Third-party Windows software has no distribution advisory feed; NVD is the only broad source. The difficulty is naming: Windows reports a program through its Add/Remove Programs (ARP) entry, for example display name "Mozilla Firefox (x64 en-US)", publisher "Mozilla" and display version "128.3.1", while NVD names it `cpe:2.3:a:mozilla:firefox`. Automated CPE assignment is a known hard problem [69]. openness solves it with a curated catalog, the *app-mapper*.

Entries. A catalog entry has an identifier (the winget identifier when one exists, for example `Mozilla.Firefox`), a kind (`application` or `device`), match rules, a version-extraction rule, one or more NVD `vendor:product` pairs, an update hint, and an optional verification rule. Match rules are case-insensitive RE2 regular expressions on the ARP display name, optionally the publisher, and an exclusion pattern; the first matching entry wins. Version extraction reads the display version or the product name, optionally through a regular expression or a template (for example "Java 8 Update 401" becomes `1.8.0.401`).

Verified and unverified entries. Entries are *curated* (reviewed by a person, `verified: true`) or *mapped* (proposed by a language model or derived automatically from winget, `verified: false`). Unverified matches are reported only when the user enables them. An entry may also carry a `verify` rule: the ARP entry counts only if its uninstall key or MSI ProductCode is in a list, or a named file exists with a valid Authenticode signature from a given signer, which the Windows collector checks through file probes. Findings record the match strength (`signature`, `product_code` or `name`).

Matching. For a matched application, the version ranges of every `cpeMatch` on the entry's products decide. A range without any bound ("every version") is not reported, because it usually reflects incomplete NVD data rather than a statement about the installed version. A `same_branch` option keeps only ranges on the installed release line, which is essential for products with parallel branches such as Firefox ESR. An optional fallback compares names with NVD product titles under strict word rules; it yields only `potentially_vulnerable`, never counts as a finding and never produces an update card.

Generation. The backend rebuilds the mapper bundle daily and after every catalog publication. Inputs are the latest published catalog, the NVD NVD JSON 2.0 yearly feeds (each verified against its `.meta` digest; any missing or mismatching year fails the build), the CPE dictionary (for titles and the fallback), and the winget index. The output (`openness-app-mapper/1`) contains the catalog, the CVE ranges for its products, and the fallback product list.

Safety checks. A new bundle is *held*, the previous one stays published and an alert is raised, if NVD files are missing or unverified, if the number of CVE records or catalog entries falls by more than 2% compared with the published bundle, if more than 10% of catalog CPEs have no CVE in NVD (which indicates a feed problem or naming drift), or if the bundle exceeds 64 MB. An identical build is recorded as unchanged. Versions are dates (`2026.11.05`, then `2026.11.05.2`), and the resolver refuses a bundle older than the one it has unless explicitly overridden.

Publication. The bundle is published as a source of the signed data mirror (§11), and a copy is released publicly as `app-mapper.json.gz` with a `SHA256SUMS` file, which the open-source resolver's `feed sync` downloads. The release notes state that the NVD does not endorse the product. The catalog bundled with resolver release data (version 2026.11.03) has 732 entries, of which 47 are verified applications, 675 unverified applications and 10 unverified device entries; only 6.5% of the application entries are verified, and unverified matches are reported only when the user enables them. The bundle holds 40,479 product-CVE records for 766 products and a fallback list of 25,947 NVD products.

9 Coverage and Honesty

The resolver writes, next to the findings, a `rule_coverage` object whose `state` is `covered`, `partial`, `missing` or `not_applicable` (device runs use `not_supported` and `unknown` instead of failing), a list of typed `gaps`, the provenance of every rule directory used (publisher URL, SHA-256 of the file the rules were built from, download time, parser), and an `unknowns` list of everything seen but not judged. Its output contract states it directly: an empty findings list with a non-empty unknowns list is not a clean host. Table 6 lists the gap kinds.

These gaps are designed to close the false all-clear paths identified in the introduction. A locked or unreadable package database yields `no_packages` rather than zero findings; an unrecognised derivative yields `unknown_distribution`; a release without downloaded data yields `missing_rules`, which the desktop resolves by synchronising and re-running; a timeout or size cap yields `packages_truncated`; an unreadable container image yields `container_not_covered`. A released rule without a fixed version yields `unknown`, not `fixed`. The coverage reason begins with the phrase "finding counts are not an all-clear". Each source also carries an age and a staleness flag (by default 2 days for distribution feeds, 35 days for MSRC, 7 days for the mapper), which warns without changing the exit code.

The desktop turns gaps into notices that "must not read as clean": platform not supported, distribution not identified, inventory unreadable, inventory incomplete, rules unavailable,

Gap	State	Raised when
`no_packages`	missing	no usable host package, or the packages module was unavailable or reported `no_packages`
`unknown_distribution`	missing	the distribution could not be identified
`missing_rules`	missing	packages exist but no rules are loaded for the release
`packages_truncated`	partial	a package limit, size limit or timeout cut the inventory
`invalid_package_records`	partial	malformed package records were skipped
`container_not_covered`	partial	an image's packages could not be evaluated or read
`not_synced`	partial	PyPI names present but PyPI data not downloaded

Table 6: Coverage gaps. Any non-partial gap makes coverage `missing`, and `resolve` then exits with status 1 after writing its output.

Python data not downloaded, and specific variants for missing privileges (no root, wrong sudo password, user not in sudoers). A finished job with a notice shows the reason in place of a clean status; a scan containing such jobs counts them as incomplete and is shown as "finished with errors". A server without a saved login receives only network checks and a notice that "nothing to install" does not mean "up to date".

What `covered` does not mean. Coverage states that rules for the host's release were loaded and every usable package was evaluated against them. It does not state that every package has advisories: packages without any vendor rule (40 of 101 in our Ubuntu 22.04 image, 95 of 141 in openSUSE Leap 15.6, §13.3) are counted in the metrics as packages without vendor rules, and a naming mismatch between evidence and feed would look the same as the absence of advisories. It does not state that the data is fresh: staleness is a warning, and data pinned by the user to an older version still reads as covered. And it inherits the scope of the feed: Canonical's USN OVAL describes only released fixes, so on Ubuntu a CVE without a fix yet is not reported, whereas the Debian tracker also lists open issues (the resolver's source registry has an entry for the CVE flavour of Ubuntu OVAL, which would add them, but it is disabled and its definitions are not consumed by the current parser).

Weaker signals. One honesty signal is weaker than the others. A *recognised but unsupported* platform (for example Debian 11 or CentOS) produces `platform_support: not_supported`, an entry in `unknowns`, a note and a desktop notice, but coverage `not_applicable` and exit status 0. A consumer that looks only at the exit code would miss it; consumers must read `platform_support` and `unknowns`.

Step	Default limit
SSH or WinRM connect	15 s (3 to 120)
SSH keepalive	15 s, 3 misses
Collector upload (SFTP)	5 min
Collector module / whole run	120 s / 900 s (+60 s grace)
Evidence download	5 min
Cleanup	60 s (SSH), 45 s (WinRM), fresh context
Resolver per job	600 s (60 to 3600)
Mirror sync / rules import	60 min / 10 min
Network port connect / probe	1.5 s / 5 s

Table 7: Default timeouts.

10 Robustness

Scanning many servers means meeting hung package managers, enormous inventories and hostile hosts. openness bounds every step.

Hard timeouts. Table 7 lists the defaults. In the collector, each module has its own deadline; if a module does not return within 3 s of its deadline, its result is abandoned and recorded as `timeout`, and modules not yet started when the global deadline passes are recorded as not started. The evidence collected so far is always written.

Process groups. Every external command the collector runs is started in its own process group on Unix, and on cancellation the whole group receives `SIGKILL`, so a hung `rpm` with children cannot outlive its deadline; a 2 s wait delay bounds pipe draining. On Windows only the direct child is killed. On the remote side the desktop starts the collector under `setsid` when available and terminates the group during cleanup. The desktop's own local resolver subprocess is bounded by a context deadline but is not placed in a separate process group.

Size limits. Package databases are streamed, and parsing stops at the record cap with the producing tool killed, so a hostile or corrupt database cannot exhaust memory. Other limits include 1 MiB per scanned line of package and process data (4 MiB for APT lists), 64 MiB per decompressed APT list file, 64 KiB for a container `os-release`, 256 MiB for the evidence document, 64 MiB for an evidence download to the desktop, 256 MiB for an evidence file read by the resolver, 64 MiB per entry of an OSV archive, and 4 GiB per file and 16 GiB in total for processed rule archives.

Hostile input. The desktop treats everything a scanned host sends as attacker-controlled, because the desktop holds credentials for every other server. Its security test fixture is an evidence file and a resolver output in which every string field carries one of 50 named attack payloads: 13 cross-site scripting variants, 8 CSV formula injections, 10 shell metacharacter forms, 4 SQL, 6 path traversal (including NUL and UNC), 4 Unicode tricks (bidirectional controls, homoglyphs, zero-width characters), 2 format strings, 2 oversize strings and 1 HTML entity break. Eight tests feed it through the scan JSON API, the HTML, CSV, PDF and DOCX exports, evidence display, the support bundle, the AI payload builder, the findings store and remediation commands. They assert that no live markup reaches an HTML export (no script element other than the page's own, no event-handler attribute, no `javascript:` or `data:` URL), that every CSV cell beginning with `=`, `+`, `-`, `@`, tab or carriage return is neutralised, that payloads round-trip as inert data, and that no payload becomes part of an executable command. The backend has an analogous corpus pushed through every public endpoint. The resolver's own robustness tests cover evidence with zero packages, a missing package key, only malformed records, truncated lists, unknown or missing distribution identifiers, and absent or mismatching integrity hashes. All of these are pass/fail tests (§13).

Remediation commands. The resolver emits remediation hints only from tokens matching a conservative character allowlist and drops and records any other token. The desktop can run an update over SSH, but it rebuilds the command from the stored hint through an allowlist of shapes (`apt-get install -only-upgrade`, `dnf upgrade`, `yum update`, `zypper update`, `apk upgrade`) with single-quoted package names that must match a strict pattern. The browser terminal types a command but does not press Enter unless the user enables it.

11 The Data Mirror

The resolver can fetch feeds directly from publishers, which is what the open-source command line does. The desktop instead obtains all data from a vendor mirror, which gives it one authenticated source, lets the backend validate files before any client sees them, and keeps clients working when an upstream is down.

Fetching and validation. The backend fetches each source on a schedule (Ubuntu OVAL and the Debian tracker every 6 h, the others every 24 h, MSRC hourly on Patch Tuesday and the day after). It accepts HTTPS only, refuses redirects to private, loopback or link-local address literals, and caps files at 2 GiB. Each file is validated to the end so that a truncated stream fails: OVAL must be bzip2 containing `oval_definitions`, Amazon data gzip containing `updates`, OSV a zip with at least one JSON document, and the JSON sources exactly one complete JSON value. A new file smaller than one third of the previous copy is not published (adversary A3). A failing source keeps serving its last good copy and is marked `stale`. Files are stored content-addressed and served unmodified.

Signed manifest. The mirror publishes a manifest `{payload, sig}`. The payload names the format, genera-

tion time, a `sequence` number, and for every source its identifier, vendor, release, status, fetch and upstream-change times, and its files with name, size, SHA-256 and upstream URL. The signature is Ed25519 [33] over the byte string `"openness-data-manifest-v1\n"` followed by the payload. The prefix provides domain separation, so a signature made for another document type (for example a catalog manifest, which uses its own prefix) can never verify as a data manifest. A new manifest, with a higher sequence number, is issued whenever source metadata changes (including fetch times); file digests tell clients which files actually changed.

Client verification. The desktop verifies the signature, refuses a manifest whose sequence is lower than the last one it accepted (rollback and replay, adversary A2), downloads only files whose digest changed, and checks each file's size and SHA-256 against the manifest. Files are staged and imported, then swapped into place directory by directory under a write lock that excludes concurrent resolver runs; an interrupted swap is repaired at the next start. A source whose file fails verification keeps its old data. Once data exists it is never updated without the user's action, except to fill a release missing during a scan.

Processed rule archives. Importing all raw files on a first sync took about 6 minutes for 356 MB in the vendor's measurement; the backend therefore also publishes deterministic, content-addressed archives of the normalised rules (about 34 MB compressed), produced by the same pinned resolver version. The desktop uses them only if its resolver reports the same rule-format identifier, and extracts them accepting only regular files at expected paths.

Versions and rollback. The desktop keeps the last data versions (three by default) as hard links, so a version costs little space, and offers "use this version", which pins an older version and suspends syncs. On the backend, an administrator can roll a source back to an earlier fetch, which republishes it and holds the source so that the faulty upstream file is not re-fetched; 30 versions per source are retained. A backend rollback publishes the old files under a *new*, higher sequence number, so clients accept it; goal G4 is thus "never older than what the backend last published", not "never older content".

Security of the update channel. Relative to the threats studied for package managers and software update systems [15, 34, 61], the design addresses arbitrary-content, mix-and-match and rollback attacks on manifests, and data older than its staleness limit is flagged to the user. Key rotation, threshold signatures and role separation of the kind TUF provides are outside its current scope.

Offline bundles. For air-gapped sites the backend produces an uncompressed tar, `openness-data-<sequence>.tar`, whose first entry is the signed manifest, followed by the raw files and rule archives. The importer verifies the signature first, then the size and SHA-256 of every entry; a missing, unlisted or mismatching entry, or an older sequence, rejects the whole bundle.

Application updates. Desktop updates use a separate Ed25519 key. The service only offers newer versions, verifies size and hash of the downloaded installer, and never installs anything itself; the user runs the installer. The release build script refuses to build without both production public keys.

12 AI Assistance

The desktop offers optional assistance from a large language model for four fixed tasks: explaining a finding, checking an update command, prioritising one server, and summarising a scan. There is no free-text prompt. AI is off by default, requires explicit consent, and never influences a finding's state: the resolver contains no model.

What is sent. The desktop builds a structured payload: package names, installed and fixed versions, CVE and advisory identifiers, severities, distribution and release, counts, remediation actions, platform facts (architecture, OS name and version, kernel, reboot required) and user-chosen context (internet-facing, environment, role). Servers are aliased `server-1`, `server-2`, and remediations `r1`, `r2`, resolved back locally. Addresses, host names, user names, credentials, tags, scan names, paths, machine identifiers, raw evidence and logs are never sent. The desktop calls a vendor proxy, never a model provider directly; the proxy forwards to a third-party hosted model, chosen by the user from the proxy's model list or automatically.

Redaction. Before sending, a guard replaces any remaining target address or host name (and its first label), the workstation's name, user names and e-mail addresses, tags, scan names and other IP addresses with placeholders, matching on word boundaries and ignoring values shorter than three characters. It then re-scans the redacted body and refuses to send (HTTP 422) if any of these known values remains; identifying text that is not in its lists (for example a host name embedded in a package's version string) is not detected. Text a user edits in debug mode is additionally scanned for private keys, cloud credentials, JSON Web Tokens and long key-like strings.

Command check. For the "check update command" task the model may propose a corrected command; that output is

never trusted. The backend checks it against an allowlist (one line, at most 500 characters and three steps joined by `&&`; package managers `apt-get`, `apt`, `dnf`, `yum`, `zypper`, `apk`, and `winget` or a single PowerShell `Install-`/`Update-` cmdlet; only refresh, install and upgrade verbs; no option that changes configuration, repositories or signature checking; no shell metacharacters) and returns `ok`, `changed` or `unsafe`, in which case the scanner's own command is returned. The desktop applies its own deny patterns (recursive deletion, `dd`, `mkfs`, pipes from `curl` to a shell, `chmod 777`, shutdown) and a similar allowlist, limited to the Linux package managers, before its terminal types anything, and labels AI output as such. Answers are rendered as text, never HTML.

Credits and storage. Usage is metered in credits based on the number of model tokens. The proxy reserves the estimated cost before calling the model, serialised per key, so concurrent questions cannot overspend; calls that receive no answer are not charged; repeated identical questions are answered from a local cache at no cost. The redacted input is processed by the selected third-party model provider, which may be located in another jurisdiction. Operators who cannot accept this should leave AI off, which is the default.

13 Evaluation

We report only quantities that we measured or that are recorded in the repositories, and we label single observations as such. All measurements were taken on the public development branches of the collector (commit `9fd423d`, version 0.8.0) and the resolver (commit `97cf3dc`, which contains the 0.8.0 release changes although its `VERSION` file still reads 0.7.0), and the private development branches of the desktop (`e7cbe4b`) and backend (`dd1926f`), on an Apple M3 Pro laptop with 18 GiB of memory.

13.1 Comparator correctness

Table 8 summarises the comparator corpora. The dpkg corpus was labelled by running native `dpkg --compare-versions` in an Ubuntu 22.04 container on pairs drawn from curated difficult versions (epochs, tildes, `~esm` suffixes, `+dfsg`, `really` versions, kernel ABI strings, 27-digit numbers) and mutations of real versions, with a fixed random seed. The apk corpus was labelled by `apk version -t` in an Alpine 3.19 container. The rpm vectors are those of rpm's own test suite (`tests/rpmvercmp.at`). All pass. No corpus exists yet for the PEP 440 comparator.

13.2 Test suites

Table 9 lists the automated tests. The resolver suite, which we ran, passed 314 top-level tests and 248 subtests with 2 skipped (nightly tests that need network access to the publishers) in 75 s, 68 s of which are the command-line end-to-end tests, which run 12 synthetic evidence files and compare against 19 golden outputs. The collector's per-change continuous integration runs on native Linux and Windows runners and in containers of Ubuntu 20.04, 22.04 and 24.04, Debian 12, Rocky 9, AlmaLinux 9, Amazon Linux 2023, openSUSE Leap 15 and Alpine 3.20; a nightly workflow adds fixed Ubuntu 22.04 and 24.04 and Windows Server 2022 and 2025 hosts and further images. A host checker compares the evidence against ground truth from native tools (`dpkg-query`, `rpm -qa`, `apk list`, Windows CIM and every machine-wide ARP entry). The collector's timeout tests require that a module sleeping one hour under a 200 ms deadline, or a run under a 300 ms global deadline, return in under 2 s, and that a grandchild process be dead after its process group is killed.

Comparator	Pairs	Disagreements	Oracle
dpkg	2,766	0	native `dpkg`
apk	2,712	0	native `apk`
rpm	91	0	rpm test suite
PEP 440	none	not measured	none yet

Table 8: Version-comparator validation.

Component	Go LOC (non-test)	Test LOC	`func Test`
collector	16,908	4,850	154
resolver	16,214	11,027	317
desktop	52,063	26,078	527
backend	28,522	14,246	239

Table 9: Code and test size (`wc -l`, including comments). The desktop UI adds 128 TypeScript files with 96 test cases.

13.3 End-to-end runs in containers

We ran the collector inside eight official container images (arm64, as root) and resolved each evidence file against feeds downloaded from the publishers on 7 November 2026 between 19:13 and 19:14 UTC, using the open-source `feed sync` path rather than the mirror. The container runtime was OrbStack on macOS. The complete command log, image digests and raw values are in the evaluation logs that accompany this paper (`eval/RESULTS.md` and `eval/NATIVE.md`). Table 10 reports the results; resolve was run three times per file and we give the median wall time and the highest peak resident set size.

Collection. Every run exited 0 and every file passed the collector's `validate`. Every file reported `collection.status partial`, which is the honest result for a bare container: there is no systemd (the `services` module is `unavailable`, or `error` on AlmaLinux, whose image ships `systemctl` without a running systemd), no listening sockets for the evidence chain, and on the dpkg images no APT lists, so `packages` is `partial`

Image	Pkgs	Collect	Evidence	Rules for	Decisions (package, vulnerability) by state						Resolve	Peak
		(ms)	(KB)	release	vulnerable fix	vulnerable no fix	fixed	n/a	pot. vuln.	unkn.	median (s)	RSS (MiB)
ubuntu:22.04	101	148	425	186,624	9	0	366	0	0	0	2.67	639
ubuntu:24.04	92	142	395	154,688	9	0	236	0	0	0	1.80	523
debian:12	88	505	367	70,164	0	143	1,162	187	64	4	0.53	213
alpine:3.20	14	29	93	8,371	2	0	248	21	0	0	0.08	34
rockylinux:9	148	451	550	7,748	265	0	333	0	0	0	0.12	40
almalinux:9	159	161	571	145,725	10	0	697	0	0	0	1.20	422
amazonlinux:2023	106	139	399	170,436	0	0	752	0	0	0	1.49	533
opensuse/leap:15.6	141	137	530	341,045	0	0	236	0	0	0	3.18	1,090

Table 10: End-to-end runs in official container images. "Collect" is the collector's own `duration_ms`; "Rules for release" is the number of normalised rules loaded for the image's release. All 24 resolve runs exited 0 with coverage `covered`, no gaps, no comparator errors and balanced reconciliation. "Vulnerable, no fix" are Debian tracker issues with no fix (102) or a deferred fix (41). Counts depend on the image build dates and the feed snapshot and are not comparable across distributions.

with the repository origin left unknown rather than guessed.

Resolution. The resolver's own matching took 2 to 108 ms; most of the wall time and memory is spent loading the normalised rules for the release from disk, which is why the openSUSE run, with 341,045 rules for its release, is the slowest and largest. Repeated runs produced identical summaries apart from the duration line. Ubuntu kernel advisories were skipped with an explicit note because the container sees the virtual machine's kernel, which is not an Ubuntu kernel (§7.4). Two data-handling details surfaced: for Alpine, 8,371 rules were loaded against 2,966 plus 5,409 synchronised from the main and community files, because entries present in both are merged.

Image	openness	Native	Both	Only openness	Only native
ubuntu:22.04	3	3	3	0	0
ubuntu:24.04	3	3	3	0	0
debian:12	0	0	0	0	0
alpine:3.20	2	4[a]	2	0	2
rockylinux:9	37	35	35	2	0
almalinux:9	4	4	4	0	0
amazon:2023	0	0	0	0	0
leap:15.6	0	0[b]	0	0	0
Total	49	49	47	2	2

Table 11: Packages with a pending security fix: openness versus the native package manager. [a]All upgrades (apk has no security flag). [b]zypper warned that the update repository metadata had expired.

13.4 Agreement with native package managers

To obtain an independent reference on the same images, we started a fresh container of each image digest and asked the distribution's own package manager which installed packages have a pending *security* update: `dnf updateinfo` and `dnf check-update -security` on Rocky, Alma and Amazon Linux, `zypper list-patches -category security` on openSUSE, upgradable packages from the `-security` pocket on Ubuntu and Debian, and `apk version -l '<'` on Alpine, which has no security flag and therefore lists every upgrade. The native checks ran about 40 minutes after the feeds were downloaded. We compared the result with the set of binary packages that openness reports `vulnerable` with a fix available (Table 11). The installed package counts matched the collector's on every image.

All four disagreements have a cause visible in the outputs. On Alpine, the two native-only packages (`apk-tools`, `ca-certificates-bundle`) have upgrades that are not tied to any CVE in the secdb snapshot, so they are not security findings by openness's definition. On Rocky, `libevent` has a fix that appears in Rocky's OSV export (advisory RLSA-2026:67910) but is not tagged as a security update in the repository's arm64 metadata, although the fixed build is available; and `openssl-fips-provider` receives findings because Rocky's OSV data names the source package `openssl`, which openness applies to every binary built from that source, while `dnf` lists only `openssl` and `openssl-libs`. The second case is a consequence of source-level rules (§7.1) and may be a false positive at the binary level. We therefore do not count these 34 pairs as correct detections.

At the level of (package, CVE) pairs on Rocky, 231 of openness's 265 vulnerable findings (87%) match `dnf`'s security list exactly, so with `dnf` as the reference the agreement on this image is 87%, not 100%; the remaining 34 are the `libevent` (8) and `openssl-fips-provider` (26) cases above. The two tools share 53 advisories and 148 CVEs; `dnf` additionally shows two CVEs only on bug-fix advisories, which openness does not report. On Alma all 10 pairs match. The native view confirms that the difference between Rocky and Alma is real: `dnf` reports 53 pending security advisories on the Rocky image, built in May 2026, and 2 on the Alma image, built in October 2026.

This comparison is small (49 packages), uses containers

Modification	Hash	Exit	Coverage, gap
package list removed	recomputed	1	missing, `no_packages`
package list removed	removed	1	missing, `no_packages`
distribution set to unknown	recomputed	1	missing, `unknown_distribution`
one byte of host name changed	stale	1	refused (integrity)

Table 12: Honesty probes on the Ubuntu 22.04 evidence.

rather than hosts, and measures agreement with the distribution's own view of available updates, which is not ground truth for exploitability or for vulnerabilities without a fix. On Debian, all 143 vulnerable findings concern issues without a fix, which the package manager cannot show at all; on openSUSE the expired repository metadata means the native check cannot distinguish a fully patched image from one that receives no updates.

Honesty probes. We modified the Ubuntu 22.04 evidence in four ways (Table 12). Removing the package list, with or without recomputing the integrity hash, produced an empty findings file but coverage `missing`, gap `no_packages` and exit status 1, with the message that finding counts are not an all-clear. Setting the distribution to an unknown identifier produced gap `unknown_distribution` and exit status 1. Changing one byte of the host name without recomputing the hash made the resolver refuse the file before matching. In no case was the host reported clean. A file whose hash was removed is processed with a note that it was not integrity-checked.

Caveats. Containers are not hosts: they have no kernel of their own, no init system and minimal package sets (14 to 159 packages), so these numbers are lower bounds for package volume and say nothing about kernel handling, services or evidence chains. Timings come from one laptop that was not isolated from other load, with three runs per file. The feed synchronisation for these eight releases took 1.5 to 24.6 s per evidence file and produced 1.5 GiB of raw and normalised data.

We also ran the collector's unit tests on macOS: 161 of 163 test results passed and 2 were skipped (one needs Linux `/proc/sys`, one is opt-in), in 10.9 s.

13.5 Observations from development

Several numbers in the project documentation are single observations, without repetition or variance, and we report them only as such. For Firefox ESR 115.3.1 on Windows Server 2022, introducing the `same_branch` rule raised the number of matched CVEs from 56 to 298. A first mirror synchronisation of 67 sources took 52 s and 34 MB with processed rule archives, against 6 minutes and 356 MB when importing raw files, with identical findings. One scan of an Ubuntu 22.04.4 virtual machine over SSH with sudo took 3 min 58 s. The desktop's resolver concurrency setting assumes 0.6 to 1 GB of memory per resolver run; this figure comes from the setting's documentation, not from a controlled measurement.

13.6 What we have not measured

Beyond the small native comparison above, we have not measured detection accuracy (precision and recall) against an independent ground truth on a representative population of hosts, nor compared openness with other scanners such as Nessus, OpenVAS, Vuls, Trivy or Grype on identical targets, nor deliberately induced each false all-clear cause from §1 on real virtual machines. The repositories contain analysis scripts for a lab validation, but their inputs are not public and their outputs are not committed, so we do not report them. The commits we measured are development states rather than tagged releases; the container evaluation can be repeated from the image digests and commands in its log. We also have not measured scan time and memory across host sizes, collector overhead on production systems, or the false-positive rate of unverified catalog entries. These are future work (§17).

14 Limitations

Platform coverage. Supported are Ubuntu 20.04, 22.04 and 24.04, Debian 12 and 13, RHEL, Rocky and AlmaLinux 8 and 9, openSUSE Leap 15.5, 15.6 and 16.0, Alpine 3.14 to 3.22, Amazon Linux 2 and 2023, and Windows builds that map to an MSRC product. For Windows this covers the operating system and third-party applications in the catalog, not other Microsoft products. CentOS, Oracle Linux (no parser for its advisories yet), SLES, Fedora, openSUSE Tumbleweed and Debian 11 are recognised and reported as not supported. Windows coverage is limited to the operating system build; .NET, Office, Edge, SQL Server and Exchange updates, Windows 7 and 8.1 extended updates, and hotpatch baselines are not yet evaluated from MSRC data. Snap packages, Livepatch and language ecosystems other than PyPI are not evaluated. On Ubuntu only Canonical's USN OVAL is used, which lists released fixes; CVEs that Canonical tracks but has not fixed yet (published in its CVE OVAL flavour) are therefore not reported, whereas on Debian the tracker's open issues are. The catalog's coverage of Windows applications rests mostly on unverified entries: 47 of the 722 application entries (6.5%) are verified.

Data and parsing. The Ubuntu OVAL parser is not a general OVAL interpreter (§5.1). In the RPM OSV exports, `last_affected` ranges are rejected. For PyPI, the highest fixed version across all ranges is used, so a host on an older, already fixed branch (1.9.6 when fixes are 1.9.5 and 2.0.1) is reported vulnerable. The PEP 440 comparator is simplified and not yet

validated against a corpus, and PyPI name normalisation does not collapse periods as PEP 503 does. MSRC supersedence is parsed but unused. The application catalog has 722 application entries and 10 device entries in its current build and depends on NVD's CPE data quality [7, 24]; most entries are unverified.

Security of the scan path. Host keys and WinRM certificates are trusted on first use, so the first connection is exposed to an active attacker. The evidence integrity hash is unkeyed and protects only against corruption. A compromised host can lie in its evidence, for example by reporting newer package versions; a credentialed scanner cannot defend against this, and openness treats evidence as untrusted data only in the sense that it cannot cause code execution or injection on the workstation. On Windows the collector does not kill descendant processes on timeout, and the desktop does not isolate its local resolver subprocess in a process group.

Honesty signals. A recognised but unsupported platform exits with status 0 (§9). Coverage reasoning cannot detect a package manager that is not modelled at all (for example software installed from source into `/usr/local`); the collector's software-inventory module detects 13 common server products by file evidence, and these are reported as inventory without a verdict until a product feed exists.

Evaluation. As stated in §13, detection accuracy at scale is unmeasured.

15 Related Work

Commercial and open-source host scanners. Nessus performs credentialed local checks with plugins that encode vendor advisories, and supports an agent [65]. Greenbone OpenVAS performs authenticated local security checks; its Notus scanner matches gathered package lists against advisory data, separating collection from matching much as openness does [30]. Qualys VMDR offers both scanner-based authentication and a resident Cloud Agent [52], and Rapid7 InsightVM similar options [54]. Wazuh's vulnerability detection correlates agent inventories with a vulnerability feed [70]. Vuls is the closest open-source analogue: an agentless scanner that logs in over SSH and matches packages against OVAL, the Debian tracker and other sources collected by companion fetchers, and stores scan results separately from the report step [26, 27].

Prior art for the individual ideas. None of openness's ideas is new in isolation. OVAL System Characteristics files already record collected items with per-item status (`exists`, `does not exist`, `error`, `not collected`) and per-object flags (`complete`, `incomplete`, `error`), and SCAP builds assessment on them [67, 68]; the openness evidence schema is in effect a JSON re-derivation of those semantics for package inventories. Nessus reports whether operating-system patch assessment was available, failed or is unsupported through dedicated plugins [64], so "a failed scan is not shown as clean" is not by itself a difference. Syft and Grype, and Trivy in SBOM mode, separate an inventory artifact from matching. Signed repositories are standard practice.

The differences lie in granularity, openness and the contract. Nessus's status plugins describe one host's assessment as a whole; openness reports coverage per release, per container image and per data source, each with a typed reason, the provenance and age of the rules used, and an exit status that automation can rely on. The coverage decision is made by an open-source, deterministic resolver whose rules, comparators and decision trail can be inspected and re-run on the same evidence file, whereas the logic of proprietary plugins cannot be audited. The evidence file states limits the collector hit (truncation, timeouts, unreadable databases) in its schema rather than in log text. And the comparators are validated against the native tools. What openness contributes is this combination, together with binary-specific rule identity across heterogeneous vendor feeds and a documented account of each vendor format's semantics. We regard this as an engineering and experience contribution rather than a research advance.

Image and SBOM scanners. Trivy [8], Grype [5] with Syft [6], Clair [53] and osv-scanner [28] analyse container images, file systems or lockfiles and match them against distribution and ecosystem advisories; Trivy and Grype also ship their vulnerability databases as signed or checksummed bundles. They target build and registry pipelines rather than live servers, and they do not log in to hosts. openness's container handling resembles theirs (os-release mapping, digest grouping) but operates on containers found on a scanned host.

Measurement studies. Holm et al. quantified the accuracy of network vulnerability scanners and found substantial false negatives, notably for unauthenticated scans [31]. Shu, Gu and Enck [62] and Zerouali et al. [71] studied vulnerable and outdated packages in Docker Hub images. Dong et al. [24] and Anwar et al. [7] documented inconsistencies in NVD version and product data, which motivates our preference for distribution data and our safety checks on NVD-derived catalogs. Li and Paxson [35] and Nappa et al. [40] studied how security patches are produced and deployed, and Dissanayake et al. surveyed socio-technical obstacles in security patch management [23]; reliable assessment of patch state is a precondition for that process. Dann et al. built a test suite for open-source dependency scanners and showed how differently they behave on the same inputs [20]; an analogous benchmark for host scanners is the most important missing piece of our evaluation.

16 Ethical Considerations

A vulnerability scanner is dual-use: the same inventory that guides patching could guide an attacker. Credentialed scans require valid credentials, so they reach only hosts for which the operator holds an account. The credential-free network checks, however, are enabled by default and probe all TCP ports of any address the operator enters; like any network scanner, openness relies on the operator to scan only systems they are authorised to test. A single bulk addition of targets expands to at most 1,024 addresses. Results and credentials stay on the operator's workstation. This paper reports no data about third-party systems: every measurement was taken on public container images, synthetic fixtures or the developers' own lab machines. We have described the data shared with the vendor (Table 1) and the AI payload (§12) so that operators can make an informed choice; both opt-out and offline operation are available. Vulnerability data is redistributed, and the obligations that follow from its publishers' terms are listed in §5.9; this paper carries the NVD notice in its Availability section.

17 Conclusion

openness shows that a credentialed scanner can make incomplete evaluation explicit without sacrificing the precision that vendor data and native version ordering provide. Its core decisions are simple and auditable: observation and judgement are separate programs connected by a typed evidence file; each advisory source is normalised into binary-specific rules with the vendor's own words preserved; each comparator is a port of the native one, validated against it; and the common reasons a host might not have been evaluated are typed gaps that prevent a clean report. The most important open work is empirical: a measurement of detection accuracy against an independent ground truth across distributions and releases, comparison with established scanners on the same hosts, a PEP 440 comparator with a native-labelled corpus, a general OVAL evaluator for vendors whose data requires it, and per-host performance characterisation.

Availability

Contact. Onur Aktaş (onur@s4e.io), Hacı Özdemir (haci@s4e.io) and Eren Akdağ (eren@s4e.io).

The collector and resolver are open source under the Apache License 2.0: https://github.com/s4e-io/openness-collector and https://github.com/s4e-io/openness-resolver. The application catalog bundle is released at https://github.com/s4e-io/openness-app-mapper. Product information and documentation: https://getopenness.com, with documentation at https://getopenness.com/openness/documentation and a machine-readable summary at https://getopenness.com/llms.txt. The desktop application and backend are proprietary.

Data notice. This product uses data from the NVD but is not endorsed or certified by the NVD. (NIST's published wording [43] refers to the NVD API; openness obtains NVD data from the NVD data feeds.)

References

[1] Alpine Linux. Alpine linux security database (secdb). secdb.alpinelinux.org. Accessed 2026-11-07.

[2] Alpine Linux. Alpine secdb license (cc by-sa 4.0). secdb.alpinelinux.org. Accessed 2026-11-07.

[3] Alpine Linux apk-tools developers. apk-package(5): APK package format, version field. apk-tools source repository (GitHub mirror). Accessed 2026-11-07.

[4] Amazon Web Services. Amazon linux security center (ALAS). alas.aws.amazon.com. Accessed 2026-11-07.

[5] Anchore, Inc. Grype: A vulnerability scanner for container images and filesystems. GitHub repository anchore/grype. Accessed 2026-11-07.

[6] Anchore, Inc. Syft: SBOM generator for container images and filesystems. GitHub repository anchore/syft. Accessed 2026-11-07.

[7] Anwar, A., Abusnaina, A., Chen, S., Li, F., and Mohaisen, D. Cleaning the NVD: Comprehensive quality assessment, improvements, and analyses. *IEEE Transactions on Dependable and Secure Computing 19*, 6 (2022), 4255–4269.

[8] Aqua Security. Trivy. GitHub repository aquasecurity/trivy. Accessed 2026-11-07.

[9] Baker, J., Hansbury, M., and Haynes, D. The OVAL language specification, version 5.11 revision 5. The MITRE Corporation, archived OVAL website, Dec. 2014. Accessed 2026-11-07.

[10] Canonical Ltd. Expanded security maintenance (ESM). Ubuntu Security. Accessed 2026-11-07.

[11] Canonical Ltd. Ubuntu OVAL. Ubuntu Security. Accessed 2026-11-07.

[12] Canonical Ltd. Ubuntu pro. ubuntu.com. Accessed 2026-11-07.

[13] Canonical Ltd. Ubuntu security metadata: OVAL data directory. security-metadata.canonical.com. Accessed 2026-11-07. Data licensed CC BY-SA 4.0 per the LICENSE file in this directory.

[14] Canonical Ltd. Ubuntu security notices. Ubuntu Security. Accessed 2026-11-07.

[15] Cappos, J., Samuel, J., Baker, S., and Hartman, J. H. A look in the mirror: Attacks on package managers. In *Proceedings of the 15th ACM Conference on Computer and Communications Security (CCS)* (2008), pp. 565–574.

[16] Cheikes, B. A., Waltermire, D., and Scarfone, K. Common platform enumeration: Naming specification version 2.3. NIST Interagency Report 7695, National Institute of Standards and Technology, Aug. 2011. Accessed 2026-11-07.

[17] Cichonski, P., Waltermire, D., and Scarfone, K. Common platform enumeration: Dictionary specification version 2.3. NIST Interagency Report 7697, National Institute of Standards and Technology, Aug. 2011. Accessed 2026-11-07.

[18] Coghlan, A., and Stufft, D. PEP 440: Version identification and dependency specification. Python Enhancement Proposals, 2013. Accessed 2026-11-07.

[19] CVE PROGRAM. CVE terms of use. cve.org. Accessed 2026-11-07.

[20] DANN, A., PLATE, H., HERMANN, B., PONTA, S. E., AND BODDEN, E. Identifying challenges for OSS vulnerability scanners: A study & test suite. *IEEE Transactions on Software Engineering 48*, 9 (2022), 3613–3625.

[21] DEBIAN POLICY EDITORS. Debian policy manual, section 5.6.12: Version. debian.org, Debian Policy version 4.7.4.1, 2026. Accessed 2026-11-07.

[22] DEBIAN SECURITY TEAM. Debian security bug tracker. security-tracker.debian.org. Accessed 2026-11-07.

[23] DISSANAYAKE, N., JAYATILAKA, A., ZAHEDI, M., AND BABAR, M. A. Software security patch management: A systematic literature review of challenges, approaches, tools and practices. *Information and Software Technology 144* (2022), 106771.

[24] DONG, Y., GUO, W., CHEN, Y., XING, X., ZHANG, Y., AND WANG, G. Towards the detection of inconsistencies in public security vulnerability reports. In *28th USENIX Security Symposium (USENIX Security 19)* (2019), USENIX Association, pp. 869–885.

[25] FIRST.ORG, INC. Common vulnerability scoring system v3.1: Specification document, revision 1. Forum of Incident Response and Security Teams, June 2019. Accessed 2026-11-07.

[26] FUTURE CORPORATION. Vuls: Agent-less vulnerability scanner. GitHub repository future-architect/vuls. Accessed 2026-11-07.

[27] FUTURE CORPORATION. Vuls documentation: Usage, report. vuls.io. Accessed 2026-11-07.

[28] GOOGLE. OSV-Scanner. GitHub repository google/osv-scanner. Accessed 2026-11-07.

[29] GOOGLE AND THE OSV COMMUNITY. OSV.dev: A distributed vulnerability database for open source. osv.dev. Accessed 2026-11-07.

[30] GREENBONE AG. Notus scanner. GitHub repository greenbone/notus-scanner. Accessed 2026-11-07.

[31] HOLM, H., SOMMESTAD, T., ALMROTH, J., AND PERSSON, M. A quantitative evaluation of vulnerability scanning. *Information Management & Computer Security 19*, 4 (2011), 231–247.

[32] ICASI. Common vulnerability reporting framework (CVRF) version 1.1. Industry Consortium for Advancement of Security on the Internet. Accessed 2026-11-07.

[33] JOSEFSSON, S., AND LIUSVAARA, I. Edwards-curve digital signature algorithm (EdDSA). RFC 8032, Internet Research Task Force, 2017. Accessed 2026-11-07.

[34] KUPPUSAMY, T. K., TORRES-ARIAS, S., DIAZ, V., AND CAPPOS, J. Diplomat: Using delegations to protect community repositories. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI)* (2016), pp. 567–581.

[35] LI, F., AND PAXSON, V. A large-scale empirical study of security patches. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS)* (2017), ACM, pp. 2201–2215.

[36] MICROSOFT. winget-pkgs: Windows package manager community repository. GitHub repository microsoft/winget-pkgs. Accessed 2026-11-07. Licensed MIT.

[37] MICROSOFT SECURITY RESPONSE CENTER. MSRC CSAF provider metadata. msrc.microsoft.com. Accessed 2026-11-07.

[38] MICROSOFT SECURITY RESPONSE CENTER. MSRC CVRF API v3.0. api.msrc.microsoft.com. Accessed 2026-11-07.

[39] MICROSOFT SECURITY RESPONSE CENTER. Security update guide. msrc.microsoft.com. Accessed 2026-11-07.

[40] NAPPA, A., JOHNSON, R., BILGE, L., CABALLERO, J., AND DUMITRAŞ, T. The attack of the clones: A study of the impact of shared code on vulnerability patching. In *2015 IEEE Symposium on Security and Privacy (S&P)* (2015), IEEE, pp. 692–708.

[41] NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY. NVD CVE API 2.0 developer documentation. National Vulnerability Database. Accessed 2026-11-07.

[42] NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY. NVD data feeds. National Vulnerability Database. Accessed 2026-11-07.

[43] NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY. NVD developers: Terms of use. National Vulnerability Database. Accessed 2026-11-07.

[44] OASIS CSAF TECHNICAL COMMITTEE. Common security advisory framework version 2.0. OASIS Standard, 2022. Accessed 2026-11-07.

[45] OPENSSF. Open source vulnerability (OSV) format, schema version 1.9.2. ossf.github.io, 2026. Accessed 2026-11-07.

[46] OSV.DEV. OSV data dumps: Per-ecosystem exports. osv.dev documentation. Accessed 2026-11-07.

[47] OVAL COMMUNITY. OVAL: Official repository for the open vulnerability and assessment language. GitHub repository OVAL-Community/OVAL. Accessed 2026-11-07.

[48] OVAL PROJECT. OVAL language 5.11.2: Schemas and specifications (official release tag). GitHub repository OVALProject/Language, tag 5.11.2 (read-only archive), 2016. Accessed 2026-11-07.

[49] PARMELEE, M. C., BOOTH, H., WALTERMIRE, D., AND SCARFONE, K. Common platform enumeration: Name matching specification version 2.3. NIST Interagency Report 7696, National Institute of Standards and Technology, Aug. 2011. Accessed 2026-11-07.

[50] PRESTON-WERNER, T. Semantic versioning 2.0.0. semver.org. Accessed 2026-11-07.

[51] PYTHON PACKAGING AUTHORITY. PyPI advisory database. GitHub repository pypa/advisory-database. Accessed 2026-11-07. Licensed CC BY 4.0.

[52] QUALYS, INC. Qualys cloud agent. qualys.com. Accessed 2026-11-07.

[53] QUAY (RED HAT). Clair: Vulnerability static analysis for containers. GitHub repository quay/clair. Accessed 2026-11-07.

[54] RAPID7, INC. InsightVM documentation: Configuring scan credentials. Rapid7 documentation. Accessed 2026-11-07.

[55] RED HAT, INC. OVAL v2 deprecation announcement. Red Hat Customer Portal. Accessed 2026-11-07.

[56] RED HAT, INC. Red hat CSAF v2 advisories and VEX files. security.access.redhat.com. Accessed 2026-11-07.

[57] RED HAT, INC. Red hat security data. Red Hat Customer Portal. Accessed 2026-11-07. Data licensed CC BY 4.0.

[58] RPM PROJECT. rpm-version(7): RPM version format and comparison. rpm.org documentation. Accessed 2026-11-07.

[59] RPM PROJECT. rpmvercmp.cc: RPM version comparison implementation. GitHub repository rpm-software-management/rpm. Accessed 2026-11-07.

[60] RUNDGREN, A., JORDAN, B., AND ERDTMAN, S. JSON canonicalization scheme (JCS). RFC 8785, Independent Submission, 2020. Accessed 2026-11-07.

[61] SAMUEL, J., MATHEWSON, N., CAPPOS, J., AND DINGLEDINE, R. Survivable key compromise in software update systems. In *Proceedings of the 17th ACM Conference on Computer and Communications Security (CCS)* (2010), pp. 61–72.

[62] SHU, R., GU, X., AND ENCK, W. A study of security vulnerabilities on Docker Hub. In *Proceedings of the Seventh ACM Conference on Data and Application Security and Privacy (CODASPY)* (2017), ACM, pp. 269–280.

[63] SUSE. SUSE and openSUSE OVAL data. ftp.suse.com. Accessed 2026-11-07. Data licensed CC BY 4.0 (since 2021-05-12) per the LICENSE file in this directory.

[64] TENABLE. Nessus plugins 117887 “os security patch assessment available”, 117886 “os security patch assessment not available”, 21745 “os security patch assessment failed” and 110695 “os security patch assessment checks not supported”. tenable.com. Accessed 2026-11-07.

[65] TENABLE, INC. Nessus user guide: Credentials. Tenable documentation. Accessed 2026-11-07.

[66] THE MITRE CORPORATION. OVAL: Open vulnerability and assessment language (archived website). oval.mitre.org. Accessed 2026-11-07.

[67] THE MITRE CORPORATION. OVAL system characteristics schema, version 5.11. archived OVAL website. Accessed 2026-11-07.

[68] WALTERMIRE, D., QUINN, S., BOOTH, H., SCARFONE, K., AND PRISACA, D. The technical specification for the security content automation protocol (SCAP): SCAP version 1.3. Tech. Rep. SP 800-126 Rev. 3, National Institute of Standards and Technology, 2018.

[69] WÅREUS, E., AND HELL, M. Automated CPE labeling of CVE summaries with machine learning. In *Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA)* (2020), Lecture Notes in Computer Science, Springer, pp. 3–22.

[70] WAZUH, INC. Wazuh documentation: Vulnerability detection. documentation.wazuh.com. Accessed 2026-11-07.

[71] ZEROUALI, A., MENS, T., ROBLES, G., AND GONZALEZ-BARAHONA, J. M. On the relation between outdated Docker containers, severity vulnerabilities, and bugs. In *Proceedings of the 26th IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)* (2019), IEEE, pp. 491–501.