TECHNICAL SPECIFICATION

VAREK

Version 1.21.1 (current) · September 29, 2026

Deterministic runtime verification of autonomous AI agents using formal methods.

Author: Kenneth Wayne Douglas, MD
Published by Sober Agentic Infrastructure, Inc.
Security contact: kenneth.douglas@soberagents.ai
License: MIT · Project: varek-lang.org · Source: github.com/kwdoug63/varek
Supersedes the v1.12 edition; corrections to earlier editions are folded in and listed in Appendix B.

VAREK is an open-source runtime, with a companion language, for verifying the behavior of autonomous AI agents before that behavior takes effect. Policy is discharged by a satisfiability-modulo-theories (SMT) decision procedure that returns one of three verdicts — SATISFIED, UNSATISFIED, or UNKNOWN — and the runtime is fail-closed: an action proceeds only on an explicit SATISFIED. The stack is vertical, from a declared plan checked before the agent starts to each system call the agent makes at the kernel boundary.

The Warden, VAREK's supervisor, holds each file open, file lookup, outbound connect, program launch and datagram send at the kernel boundary through seccomp-BPF and seccomp user-notify, and decides it before it runs. From v1.13 the decision is made in the enforcement path by a purpose-built SMT decision procedure over bounded strings (the object) and 32-bit bitvectors (the open flags), cross-checked against an off-the-shelf SMT solver; from v1.14 it matches exact, prefix, suffix, contains and glob patterns on the resolved canonical path. From v1.15 every authorization carries a certificate that a small, independently written checker must accept before the action takes effect, so the checker, not the procedure, is the trusted decision code for authorizations. From v1.16 the verdict stream is hash-chained, signed with Ed25519 and anchored off the host, so it cannot be rewritten by whoever holds it. From v1.17 the agent runs unprivileged, file lookups are decided like opens, and the Warden's own key, anchor and log are refused by identity through any path. From v1.18 the pre-execution plan gate runs the cross-action data-flow check, the bounded-refusal breaker and the load-time progress-safety proof, which certifies that an unattended, human-out-of-the-loop (HOOTL) deployment always has an automated next move. From v1.21 the Warden decides each outbound connect on the destination it copied once, dials it itself outside the agent's empty network namespace, and hands the agent the connected socket — the same resolve-then-decide-then-deliver discipline it has applied to

files since v1.12. v1.21.1 lets a plan step declare how it opens a file, so the plan gate can authorize a declared read that the runtime would allow.

No release from v1.9.1 through v1.21.1 changes the three-state decision semantics or the symmetric-suppression invariant: no extension may move a genuinely unsafe action to SATISFIED.

---

1 Motivation

Agentic AI systems act in the world: they call tools, read and write data, and chain those actions toward goals. The dominant safety posture is probabilistic — a system that is "usually right." Medicine does not deploy systems that are usually right, because the tail is where people get hurt. VAREK applies that standard to agent behavior.

The governing principle is **authorization before execution**: an action is checked against an explicit policy, with a determinate decision, before it is allowed to take effect. The decision procedure does not guess. When it cannot prove an action satisfies policy, it returns UNKNOWN, and the runtime refuses to proceed. A refusal is a safe outcome; an unverified action is not.

---

2 Architecture overview

VAREK is a single vertical stack. Before the agent starts, an optional plan gate checks the agent's declared action-graph — a directed acyclic graph of planned actions — against policy and, with a flow policy, the data moving along its edges. While the agent runs, the Warden holds each policy-relevant system call at the kernel boundary, lifts it into a structured action, decides it, and enforces the verdict before the call completes. System calls the kernel filter admits without asking (memory, time, threads, and reads and writes on descriptors the agent already holds) run undecided; everything else is refused outright.

2.1 Surface language and compilation

Policies are written in the `var::` surface language and compiled to a formal representation suitable for a decision procedure. The compiler is the trust boundary between human-authored intent and machine-checked obligation. The VAREK pipeline language, in which unsafe operations are not expressible, is stable at v1.0; active development is in the Warden runtime.

2.2 Decision procedure (v1.13, v1.14)

Compiled policy is discharged by an SMT decision procedure returning SATISFIED, UNSATISFIED, or UNKNOWN. UNKNOWN is never coerced to a pass; the runtime treats it as fail-closed.

In the Warden (from v1.13) the procedure is purpose-built for a quantifier-free fragment of two theories: bounded strings for the object an action names (the canonical path, `host:port`, or the exec path) and 32-bit bitvectors for the open flags (`(flags & mask) == value`, faithful to the syscall argument, the kernel's `int flags`, rather than to the descriptor's later state, which `fcntl` can partly change). A policy compiles to one ordered formula per action kind. The same procedure answers concrete per-syscall queries (where satisfiability reduces to evaluation), symbolic-flag queries for the plan gate (SATISFIED only if every admissible flags value is), and rule-reachability queries at policy load (is there any action for which this rule fires first). Policies can say read-only (`readonly`, `access=ro`, `+O_…` / `-O_…`); before v1.13 a path rule admitted every open flag.

From v1.14 a path or exec rule takes a matcher before its constant — `exact`, `prefix` (the default), `suffix`, `contains` or `glob` (`?`, `[...]`, `*`, `**`, and `/**/` for any number of segments) — each a regular language over bytes, matched on the resolved canonical path, so a symlink with an innocent name is decided as its target. A policy can therefore deny a kind of file wherever it appears under an allowed tree: private keys, dotenv files at any depth, a patient's psychotherapy notes. Globs compile to a small automaton stepped word-parallel.

The fragment boundary is a soundness obligation: an over-length string (more than 4,095 bytes), a flag bit outside the ABI set, access mode 3, or a query beyond the enumeration bound is UNKNOWN. Rule reachability is decided exactly over the abstract domain (any byte string up to the bound) by a breadth-first search of the product of the rules' automata, and each reachable rule comes with a shortest witness; at load the Warden reports every rule that can never fire. A general SMT solver is kept off the hot path (v1.5 measured tail latencies of tens of milliseconds for its string theory) and used instead as an independent differential oracle over every repository policy and seeded random policies — zero disagreements over 26,624 checks at v1.13 — joined from v1.14 by a derivative-based procedure that settles bounded reachability where the solver's regular-expression theory does not finish.

Cost. At v1.14 the procedure alone decided a representative path in about 155–165 ns on the finance and healthcare policies. In the live Warden on a 2-vCPU host (`varek bench`, v1.22.0), the Warden's own time for a decision is 13–17 µs for a refused file open and 56–66 µs for an authorized one, which includes resolving and opening the file and checking its

certificate; what the agent waits for the call, notification round trip included, is 53–57 µs and 70–80 µs, against 1–2.4 µs natively (`bench_results_v1_22_0.txt`). From v1.16 a policy's globs are capped at 4,096 tokens, which bounds the work of one decision: on adversarial policies built to hit the worst case, one decision in the live Warden takes about 26 ms (median); real policies are unchanged.

2.3 Certificates: every authorization independently checked (v1.15)

Through v1.14 the Warden authorized an action when the procedure said SATISFIED, so the procedure — about 1,400 lines of C with fast paths, word-parallel automata and a reachability search — was trusted. From v1.15 every SATISFIED verdict carries a certificate naming the deciding rule and a witness that the rule's constant matches, and the Warden authorizes the action only if an independently written checker accepts it. The checker (about 540 lines including SHA-256) has its own policy parser and its own matchers, shares no source with the procedure, and has none of its optimizations. If it refuses, the action is denied and recorded as `certificate_refused`. A test build with a deliberately planted procedure bug shows the effect: the procedure wrongly says SATISFIED for a file under an explicit deny rule, the checker refuses, and the open is denied.

For the property that matters most — no action is authorized that the policy does not allow — the decision procedure moves from *trusted* to *checked*, for logic bugs. The two run in the same process, so a memory-safety bug in the procedure could in principle corrupt the checker's state; this is stated in the trusted-computing-base document. Certificates are recorded in the verdict stream with the SHA-256 of the policy, so `tools/varek_audit.py` can re-check every authorization in a saved run later, from the stream and the policy file alone. Building and checking a certificate adds about 1 µs per authorized open on the example policies.

2.4 Pre-execution action-graph verification and the plan gate (v1.6; in the Warden from v1.12.4)

Agent plans are represented as action-graphs and verified compositionally before any constituent action runs. As a plan is revised mid-task it is re-verified, so the authorization guarantee holds across plan changes rather than only at the start.

In the Warden this is the optional `--plan` gate. It decides each `file_open` step on the lexically canonical form of its declared absolute path (from v1.12.4; symlinks are not followed, because there is no agent yet, and every open is still decided again at run time). From v1.18.0, with `--flow-policy`, it runs the whole v1.7–v1.9 pipeline — the cross-action data-flow check (§2.6), the bounded-refusal breaker and the progress-safety proof (§2.7) — which

through v1.17.0 were a library with tests that the Warden did not call. From v1.21.0 it decides `net_connect` steps like the connect they name (§2.14), where through v1.20.0 it refused them all.

Fields on plan steps (v1.20). A step can carry up to 16 `key=value` fields after its target (quoted values may hold spaces, e.g. `contains="a customer record"`), and the flow policy's rules match them like any named argument. Fields are the agent's declarations, like the plan's edges: the node check and the runtime see only the target, and nothing compares the agent's later calls with its fields. Because the agent chooses them, the Warden refuses a flow policy whose rules match a field unless the policy declares `trust_declared_fields`. Targets are limited to 4,095 bytes; the node check calls a longer one UNKNOWN.

Declared open flags (v1.21.1). See §2.15.

2.5 Kernel-boundary integration (v1.7)

The verification layer binds to runtime enforcement at the system boundary. Observed low-level behavior is intercepted, lifted into structured actions, checked against policy, and gated before the action completes. The same policy a developer reasons about is the policy enforced against the agent's actual runtime behavior — not a weaker approximation of it.

2.6 Cross-action data-flow verification (v1.8)

v1.8 extends verification from single actions to data flow across sequences of actions, tracking where data originates and where it is permitted to travel. Policies can express constraints over information movement — for example, that data drawn from a sensitive source must not reach a particular sink — rather than evaluating each action without regard to what came before it. v1.8.0 adds operator-designated, audited declassification (sanitize-then-send), the only mechanism that can bypass the read-secret-then-exfiltrate guarantee, governed by four test-pinned safety properties. From v1.18.0 flow rules can match a URL's host component (`match url.host`), and a plan whose URL a rule cannot read is refused rather than skipped.

2.7 Progress-safety and the bounded-refusal breaker (v1.8.2, v1.9, v1.19)

v1.6–v1.8 prove safety: nothing unauthorized executes. They do not by themselves prove *liveness* — that the system always has a legal, automated next move. An unattended deployment needs both, or "never requires a human" is a hope: a policy could admit a reachable state in which an action is refused and no authorized fallback exists, a deadlock only a human could break.

v1.9 discharges that obligation once, at policy load, before anything runs. It certifies:

> *For every non-authorizing verdict (UNSATISFIED or UNKNOWN) the policy can produce, the deterministic refusal resolution reaches an automated terminal outcome in finitely many steps, with no point requiring human intervention.*

The proof decomposes into four obligations — bounded refusal, disposed UNKNOWN, disposed exhaustion, and an authorized-fallback reachability proof that composes the underlying decision procedure as its authorization oracle. The result is three-state like every other VAREK verdict: SATISFIED (certified human-out-of-the-loop), UNSATISFIED (a concrete gap, failing obligation named), UNKNOWN (could not decide; fail closed, not certified). From v1.18.0 the Warden runs it on the `--flow-policy` at startup and refuses to start if it fails, so "no human at run time" is provable: if no automated terminal is guaranteed, the system never reaches run time.

The bounded-refusal breaker (v1.8.2) is a loop bound in the trusted boundary, keyed by (session, action-graph signature), so a stuck or adversarial planner cannot resubmit a refused action-graph forever: after the policy's refusal budget, the outcome is a declared terminal (deny, or a pre-authorized action). Each verdict stays a pure function of plan and policy; the breaker only interprets the sequence of verdicts. From v1.18.0 its counts persist across runs in a state file the agent cannot reach. From v1.19.0 a flow policy must also declare `session_refusal_budget N`: the breaker counts every refused plan in a session, whatever the plan, so a planner that changes one step each time is stopped too.

2.8 Enforcement hardening (v1.9.1)

io_uring bypass closed. io_uring dispatches operations from kernel worker threads off the syscall entry path, where a seccomp filter — and therefore the Warden's user-notification mediation — cannot observe them. The Warden's filter refuses io_uring instance creation; there is no sound way to mediate it at this layer. From v1.21.0 `io_uring_setup` answers `ENOSYS` rather than killing the agent, so runtimes such as Node.js fall back to epoll; no ring can be created, and `io_uring_enter` and `io_uring_register` stay in the hard-deny set. The live filter's test (`test_v14_filter`) asserts this.

TOCTOU discipline, measured. When a mediation decision depends on a pointer argument — a path, a socket address — letting the original syscall proceed after approval is unsafe: a second thread in the target can rewrite the argument between the check and the kernel's use of it, a time-of-check-to-time-of-use (TOCTOU) race. For file opens, the supervisor resolves the approved path itself and injects the resulting descriptor into the target, so the

target's syscall never runs against mutable memory. On the race harness (`tests/seccomp_toctou_harness.c`), approve-then-continue leaked the protected target 1,848 to 1,889 times in 20,000 attempts and resolve-and-inject leaked 0 (three runs each on a 2-vCPU host, `tests/toctou_results_v1.18.0.txt`; the count depends on the host). Connects, for which there is no file descriptor to resolve, were deny-only from v1.9.1 through v1.20.0; v1.21.0 applies the same discipline to them (§2.14). Agent-initiated `execve` remains deny-only.

2.9 Mediation completeness (v1.9.2)

v1.9.2 inverts the Warden's enforcement model and wires the result into the live supervisor. The filter is *default-deny*, so the residue — every syscall not explicitly admitted — is refused by construction rather than by enumeration. Completeness of mediation is treated as a closure property: for each policy-relevant effect, the obligation is to have enumerated all the syscalls that produce it, across all application binary interfaces.

Default-deny allowlist. Unknown syscalls, variant syscalls (`clone3`, `openat2`, `faccessat2`, the `pidfd_*` family), and the 32-bit multiplexers (`socketcall`, `ipc`) are denied unless admitted — a bypass that was not enumerated becomes one that was not admitted.

Native-ABI lockdown. The deny default applies across every architecture, and no secondary ABI is admitted, so the same operation re-issued through the 32-bit compat ABI (`int 0x80`) or the x32 ABI (`__X32_SYSCALL_BIT` set) hits the deny path — closing the most common real-world seccomp escape, asserted on the live kernel by a release-blocking test.

Hard-deny set and namespace denial. Syscalls with no legitimate use inside a mediated agent — `ptrace`, `bpf`, `userfaultfd`, `process_vm_readv`/`writev`, `pidfd_getfd`, the mount/FUSE family, the kernel-module and `kexec` family, `perf_event_open`, and the key-management calls — are denied with process termination in strict mode. `clone` and `unshare` are filtered on their scalar flags argument (a register value, so the check is race-free) to deny `CLONE_NEWUSER` and the namespace-creation set. `clone3`, whose flags live behind a pointer, never executes: from v1.12.2 it is answered `ENOSYS`, so the C library falls back to `clone` and every thread or child passes the same flag filter.

Integration and validation. The default-deny baseline replaces the filter the Warden installs. A conformance target — a running, agent-shaped workload — exercises the boundary end to end: it opens an allowed file (served by descriptor injection), is refused a denied path, creates a socket, and is refused an outbound connection (deny-only through v1.20.0; from v1.21.0 refused because the conformance policy has no host rule allowing it). Under live enforcement it passes every phase. An observe mode logs rather than blocks, so a target's required syscalls can be harvested before the deny default is turned on, while the hard-deny

set still terminates on a dangerous call. The conformance target also surfaced, and verified the fix for, a regression in which deny-only exec mediation refused the target's own bootstrap `execve`; the Warden authorizes exactly the operator-specified launch, once per run.

2.10 Supervisor/target lifecycle coupling (v1.9.3)

Enforcement assumes the supervisor is alive. If it stops, mediated calls fail closed, but an agent that keeps running can still act through capabilities it already holds. v1.9.3 removes that window in the live Warden.

The target requests a kill signal on supervisor death before it installs its filter, and first confirms, through a pipe only the supervisor holds open, that the supervisor is still alive. It runs as the first process of a dedicated PID namespace, so when it dies the kernel kills every other process in the namespace; on orderly shutdown the supervisor kills the whole tree. The supervisor watches the target through a process descriptor. The namespace requires `CAP_SYS_ADMIN`; the Warden checks for it at startup and refuses to run without it unless explicitly told otherwise. A crash test kills a live supervisor with `SIGKILL` and, separately, stops it with `SIGTERM`, and asserts no agent process survives either; against the pre-fix Warden it fails (two survivors after a crash, one after orderly shutdown).

2.11 Mediation correctness (v1.12.0–v1.12.4)

A decision is only as sound as the identity of the object it is made about, and a record is only evidence if the process it describes cannot forge it.

Resolve-then-decide. Through v1.9.3 the Warden matched policy against the pathname string the agent supplied and then, separately, opened whatever that string resolved to — two steps that could name two different objects: `/allowed/../../denied` matched an allowed prefix as text, a symlink planted inside an allowed directory was followed to a denied target, and `/proc/self`, which the kernel resolves in the caller's context, resolved to the supervisor rather than the agent. From v1.12 the Warden resolves the object once, reads the canonical path back from the resolved descriptor, decides policy on that path, and hands the agent the same descriptor, so decision and delivered capability refer to the same inode by construction. From v1.12.1 it pins the object with `O_PATH`, decides, and only after an ALLOW opens it with the agent's flags, so a denied open has no side effect (no truncation, no creation, no FIFO that wedges the supervisor). From v1.12.3 ordinary symlinks are followed and decided as their targets, so dynamically linked agents (CPython, a JVM, Node.js) can load their libraries; a leading `/proc/self` or `/proc/thread-self` is mapped to the agent's own process, and any other process's `/proc` is refused after resolution. Because the decision is on the canonical path, `allow path` prefixes name canonical locations (`/usr/lib/`, not `/lib/`).

Record integrity. Every agent-controlled field is escaped, so no agent string can begin, end or fabricate a record, and each record reports the resolved canonical object it was decided on. From v1.12.1 the agent's own stderr is relayed with an `[agent]` prefix and every record carries a per-run id and a contiguous sequence number, so a forged, missing or foreign record is detected.

Threads, children and inbound networking. From v1.12.1 `bind`, `listen` and `accept` are refused and the agent runs in its own network namespace, holding only a loopback interface. From v1.12.2 agents can start threads and wait for their children, and the launch exec is granted once per run, not once per process, closing a race in which a thread could change the launch path.

2.12 Lookups, protected objects and an unprivileged agent (v1.17)

Lookups are decided. `stat` (`newfstatat`), `statx`, `access`, `faccessat`, `faccessat2`, `readlink` and `readlinkat` were admitted and unlogged, so an agent could learn whether any file existed and read where any link pointed. From v1.17 the Warden resolves the name like an open, decides it with the SMT decision procedure as a read-only open of the same object (`access(W_OK)` as a write), certifies an ALLOW, records it, and answers the call itself. A name that does not exist is decided on where it would be — `ENOENT` inside the policy, `EACCES` outside it — so a lookup says nothing about what exists elsewhere. Lookups of the directories leading to an allowed path are answered with existence and type only, and recorded as `metadata_ancestor`; lookups of descriptors the agent already holds are answered like `read()`.

Protected objects, refused by identity. Every open and lookup is checked after resolution, on the object actually pinned and before the policy is asked, against the device and inode of the signing key, the anchor and the verdict stream, so a bind mount, overlay or hard link into an allowed tree cannot reach them (`protected_object`). Raw storage and memory devices — block devices, `/dev/mem`, `/dev/kmem`, `/dev/port`, the sg, bsg and NVMe command devices, `/proc/kcore` — are refused the same way whatever the policy says (`raw_device`).

The agent runs unprivileged. Through v1.16.3 the agent ran as root with every capability, held back only by the filter. From v1.17 it runs as `nobody` or the user `--run-as` names; before the filter is loaded, the Warden's child clears its supplementary groups, empties the capability bounding and ambient sets, switches group and user, and verifies that no capability remains. File access does not depend on the agent's rights: the Warden opens allowed files on its behalf only after the policy allows it.

2.13 A verdict stream its holder cannot rewrite (v1.16, v1.18)

From v1.12.1 the supervised agent cannot forge, drop or reorder a record. From v1.16 neither can a person who holds the log afterwards. Every record is hash-chained; with `--sign-key`, run start, a checkpoint every 64 records (and at least once a second) and run end are signed with Ed25519; with `--anchor`, each checkpoint is appended to storage the log's holder cannot rewrite. `tools/varek_audit.py --pubkey --anchor` catches any edit, insertion, removal or reordering before the last signature, and a stream cut short, by anyone without the key, and any rewrite of anchored history even by someone with it. It verifies signatures with its own pure-Python RFC 8032 verifier. The Warden refuses to start if the policy would let the agent open the key, the anchor, the verdict stream or a raw disk.

An anchor on the Warden's own host does not protect against that host's root, who also holds the key. From v1.16.2 `tools/varek_anchor_forward.py` sends each checkpoint off the host, normally within a second, spooling through outages, to an append-only receiver set up by `tools/varek_anchor_receiver.sh` — an SSH account restricted to one forced command that appends well-formed anchor lines to a `chattr +a` file. `tools/varek_preflight.sh` (v1.16.1) checks a deployment before it runs, including every location the Warden refuses at startup, and with `--run` performs an audited trial run.

Evidence export. The verdict stream exports as a Bill of Materials in the CycloneDX 1.6 format: the runtime as a tool component, the supervised run as the subject, each authorized object and connected destination as a component, and an Authorization-Before-Execution attestation. From v1.18.0 the attestation is derived from the stream, the exporter checks the policy file against the recorded SHA-256, the BOM can be signed (JSON Signature Format, Ed25519), and output is tested against the published CycloneDX 1.6 schema. The exporter refuses a stream that does not verify, so a corrupted log cannot be laundered into an attestation. (CycloneDX is a trademark of the OWASP Foundation; VAREK is not affiliated with or endorsed by the OWASP Foundation or the CycloneDX project, and uses the name only to describe interoperability with the openly published CycloneDX format, standardized as ECMA-424.)

2.14 Decided connections (v1.21.0)

Through v1.20.0 a supervised agent had no network: every connect was refused, whatever the policy said, because letting an allowed connect continue in the kernel would let a second thread change the destination between the Warden's check and the kernel's read of it. From v1.21.0 the Warden decides each outbound connect and carries it out itself:

1. The destination is copied once from the agent's memory.

2. It is spelt canonically: `a.b.c.d:port`, `[IPv6]:port` (an IPv4-mapped address as its IPv4 form), or `unix:<canonical path>` for a Unix socket named by a path, resolved like a file open.
3. The SMT decision procedure decides it against the policy's `host` rules, and the independent checker must accept the certificate of any ALLOW.
4. The Warden, in the host's network namespace, makes a socket of the agent's kind, copies over the socket options the agent set (from a list of 58), and connects it to the copy it decided on; a Unix connect is made with the agent's uid and gid.
5. `SECCOMP_IOCTL_NOTIF_ADDFD` puts the connected socket in place of the agent's descriptor, with the same number and the agent's close-on-exec and non-blocking state, and the agent's `connect` returns what its own would have.

The kernel never reads the agent's socket address, so the race stays closed: in 2,000 attempts to swap the destination from an allowed one to a denied one after the check, the denied side was reached 0 times. The agent's own network namespace stays empty; the only way out is a socket the Warden made. Covered: TCP and connected UDP over IPv4 and IPv6, and Unix stream, datagram and seqpacket sockets named by a path (IPv6 decisions are tested; IPv6 dialing was not exercised on the release host).

A send that names a destination is still refused, because an unconnected datagram cannot be decided without a race on its address; a send with no destination goes to the peer of a connect the Warden decided. Routing options that could redirect a connected socket's packets (`IP_OPTIONS`, `IPV6_RTHDR` and two legacy IPv6 forms) are refused on every socket. `bind` is refused except the wildcard, port-0 form libuv uses before connecting a UDP socket; `listen` and `accept` stay refused.

Cost. On a 2-vCPU host a connect took about 75 to 125 µs longer than a native one at the median; a whole Python `requests.get` to a local server took about 0.17 ms longer (1.21 → 1.38 ms). A profile put about 2 µs of the Warden's part in the decision and certificate check.

Scope. VAREK decides *where* an agent may connect, not what it sends: an allowed host is a channel. Rules match the numeric address dialed; host names are the planned second stage (§8).

2.15 Declared open flags at the plan gate (v1.21.1)

A plan step has no open flags of its own, so through v1.21.0 the plan gate decided a `file_open` step with the flags unknown. A path allowed only by a rule with a flag clause (`readonly`, `access=ro`, `-O_TRUNC`) was therefore UNKNOWN at the gate, although the runtime

allowed the same open — and all five shipped sector policies grant their libraries read-only. From v1.21.1 a `file_open` step may declare how it opens its file:

```
action load  file_open  /srv/agent/reference/in.json  open=read
action save  file_open  /srv/agent/work/out.json      open=O_WRONLY|O_CREAT|O_TRUN
edge   load  save
```

`open=read` means `O_RDONLY` and nothing else; otherwise the value is one access mode followed by any of 17 named flags, each at most once, joined by `|`. A name has the value an agent's `open()` passes for it (`O_LARGEFILE` is the kernel's bit, as in the policy language). Anything else — another word, a lowercase name, a repeated flag, an `open` field on a step that is not a `file_open` — makes the step UNKNOWN, never SATISFIED, with the reason logged. The node check decides and certifies the step with exactly the declared flags, as the runtime decides an open with the flags the agent passes. The field is a declaration: it does not bind the agent, and an open with other flags is decided on its own flags at run time.

---

3 Three-state decision semantics

The three-state result is the core of VAREK's safety claim and the reason the system is honest about its own limits.

RESULT	MEANING	RUNTIME BEHAVIOR
SATISFIED	Provably compliant	Proceed
UNSATISFIED	Provably non-compliant	Deny
UNKNOWN	Not provable within bounds	Fail closed (deny)

A two-state system is forced to convert every UNKNOWN into either a false pass or a false block. VAREK refuses that conversion: it reports UNKNOWN as UNKNOWN and lets the fail-closed runtime resolve it safely. In the running agent an UNKNOWN is refused, like an UNSATISFIED (symmetric suppression). At the plan gate with a flow policy, it goes to the deterministic disposition that policy declares, and the progress-safety proof certifies that every such refusal ends in an automated outcome.

3.1 Soundness, and why UNKNOWN is the honest residue

VAREK is **sound but deliberately incomplete**. Soundness: no action is reported SATISFIED unless it provably satisfies policy. Incompleteness: some safe actions cannot be proved safe within the decision procedure's bounds and are reported UNKNOWN. The asymmetry is intentional — over-refusing a safe action is a utility cost; wrongly authorizing an unsafe one is a safety failure. The system is built to never make the second trade.

This is also why the input space being effectively infinite is not a problem the way enumerating edge cases would be. The decision procedure reasons over whole domains symbolically rather than sampling points, and the three-state verdict is *total*: every input lands in exactly one of SATISFIED / UNSATISFIED / UNKNOWN, deterministically, with UNKNOWN as the fail-safe residue. Coverage of the infinite space is by construction, not by enumeration.

3.2 UNKNOWN diagnostics and resource bounds

An UNKNOWN record's `rule` field says why: `fragment_escape_flags`, `fragment_escape_length`, or `default_deny_unknown` when no rule matched (from v1.13.0). The procedure's work is bounded by construction — a length guard on the object (4,095 bytes), a bounded enumeration of symbolic flag bits (16), and from v1.16.0 a cap of 4,096 glob tokens per policy, which bounds the steps of one decision. There is no wall-clock ceiling and no memoization cache; a bound is structural, so a decision that would exceed it is UNKNOWN before it starts rather than cut off mid-way.

3.3 Measuring the residue

The verdict-distribution harness (v1.13.0) measures what fraction of safe actions a policy clears and gates every change on `unsafe_satisfied == 0`. On its synthetic seed corpus of file opens, v1.13 cleared 85.4% of safe opens with no unsafe open authorized, where the example policies as v1.12.4 shipped them cleared 75.6% and authorized 24 unsafe opens. These are figures on a synthetic seed corpus, not on a customer workload; a baseline on a customer workload is future work.

---

4 Verification scope and guarantees

VAREK verifies that agent actions and plans satisfy explicitly authored policy, that data flow across declared plan steps respects explicitly authored information-flow constraints, and

that an unattended policy is progress-safe. The guarantees are relative to the policy as written and to the fidelity of the action model derived at the system boundary.

What the Warden decides. File opens, file lookups, outbound connects, datagram sends and the agent's launch are held and decided. Agent-initiated program launches after the first are refused whatever the policy says, as are sends that name a destination and inbound networking. System calls the filter admits without asking (memory, time, threads, and reads and writes on descriptors the agent already holds) run undecided, and every other system call is refused outright.

Independence from the agent platform. The decision procedure and the Warden run outside the supervised agent and do not depend on its model provider, agent framework, or orchestration platform: the action model is derived at the system boundary, and enforcement uses seccomp-BPF and seccomp user-notify, which apply to any supervised process. The verdict stream and its CycloneDX 1.6 export are produced by the Warden, not by the agent or its vendor, and can be signed and anchored, so a third party can inspect and verify the authorization record without relying on the vendor that built the agent. The dependency is on the host kernel (Linux with seccomp user-notify) and on the Warden's trusted computing base, stated in `docs/security/TRUSTED-COMPUTING-BASE.md`; it is not on any particular model or agent platform.

Declarations are not observations. Plan edges, plan-step fields (v1.20) and declared open flags (v1.21.1) are the agent's declarations. The plan gate decides what the agent declares; the runtime decides what the agent does. Nothing compares the agent's later calls with its declarations, and every call is decided again at run time.

Explicit information flow. The data-flow subsystem covers explicit flows — data that moves through observable action inputs and outputs. Implicit flows (information conveyed through control structure rather than data movement) are on the roadmap and are not claimed. An allowed network destination is a channel; what the agent sends over it is outside the Warden's decision. The published `docs/security/threat-model.md`, `docs/security/threat-model-dataflow.md`, `docs/security/bypass-classes.md`, and `docs/security/TRUSTED-COMPUTING-BASE.md` are the authoritative statement of assumptions, adversary models, in-scope and out-of-scope threats, the per-class mediation status, and the per-component trusted-versus-verified status of the verification chain. Every release's notes list its known limits.

5 Quality and testing

The testing posture mirrors the runtime posture: where a guarantee cannot be established, the build fails closed rather than presenting an unverified result as a passing one. Each release adds a regression suite that fails against the previous Warden, and every earlier suite must still pass.

- Progress verifier: `test_v19_progress.c`, 10/10, clean under `-fsanitize=address,undefined`.
- Enforcement, measured directly: the TOCTOU race harness reports 1,848 to 1,889 leaks in 20,000 attempts for approve-then-continue versus 0 for resolve-and-inject (v1.18.0 re-run). io_uring refusal under the live filter is asserted by `test_v14_filter`.
- Mediation completeness on the live kernel: `test_v192_abi_lockdown.c` admits the native call and kills the x32 call (release-blocking); `test_v192_baseline_deny.c` confirms the hard-deny set and `CLONE_NEWUSER` denial. `target_conformance` passes every phase under the live Warden.
- Lifecycle coupling: `test_v193_lifecycle.c` kills a live supervisor and asserts no agent process survives.
- Mediation correctness: `make test-v112` through `make test-v1124` — an adversarial target exercising traversal, symlink, `/proc/self` and record-injection escapes; side-effect-free denial; threads and children; dynamically linked agents; the plan gate.
- Decision procedure: `make crosscheck` (zero disagreements with the off-the-shelf SMT solver over 26,624 checks at v1.13), the derivative-based reachability oracle (v1.14), and the verdict-distribution harness's `unsafe_satisfied == 0` gate. `make test-v1130`, `test-v1140`, `test-v1150` (including a planted procedure bug the checker must stop).
- Record integrity and deployment: `test-v1160` through `test-v1162`, `tools/varek_preflight.sh`, and a CI job that builds the Warden with libseccomp and libsodium, lints the shipped policies and runs the preflight on each.
- v1.17.0: 48 checks (protected objects by identity, mediated lookups, unprivileged agent). v1.18.0: 73 checks plus the v1.7 layer's test. v1.19.0 and v1.20.0: breaker session limit and plan-step fields.
- v1.21.0: `make test-v1210`, 90 checks, running curl, Python `requests` (including verified TLS), raw Python TLS, Python Unix and UDP sockets, and Node.js `http`, `https`, `tls`, `net` and `dgram` as the agent under the Warden, plus the 2,000-attempt destination-swap race.

- v1.21.1: `make test-v1211`, 33 checks, 23 of which fail against the v1.21.0 Warden.
- From v1.18.0 each release's own code was reviewed independently before tagging, and the findings fixed in the release are listed in its notes.

---

6 Demo

A narrated demo walks through the stack end to end: authorization on a compliant action, denial on a violating action, fail-closed behavior on UNKNOWN, and cross-action data-flow scenarios where a sequence is blocked on the basis of where data originated (youtu.be/6E1Yrt02GNM). A browser visualization of the three-state verdict is published on the project site; the runnable C demo is in the repository (`v1_7`, `make demo`; `demo_hootl.c` for the HOOTL walkthrough), and the conformance target runs a real workload under the live Warden. The VAREK Verdict Service at `api.varek-lang.org` answers plan-verification requests with the v1.6 compositional evaluator; its front end, `v1_6/plan_verify_cli.c`, is in the repository from v1.16.3 and remains a demonstration with a built-in policy.

---

7 Documentation

- `README.md` — what VAREK is, the verdict model, and a summary of every release.
- `RELEASE-v1.21.1.md`, `RELEASE-v1.21.0.md`, and the release notes for each earlier version — what changed, how it was tested, what an independent review found, compatibility, and known limits.
- `CHANGELOG.md` — the full change record, with dated corrections.
- `docs/security/threat-model.md` and `docs/security/threat-model-dataflow.md` — adversary models, in-scope guarantees, and out-of-scope non-goals.
- `docs/security/bypass-classes.md` — the bypass-class checklist and mediation-completeness argument, with per-class status.
- `docs/security/TRUSTED-COMPUTING-BASE.md` — per-component trusted-versus-verified status of the verification chain.
- `docs/security/v1.21-stage2-host-names.md` — the design for host names without agent DNS.
- `docs/security/v1.10-architecture-roadmap.md` — the model- and TCB-changing track.

- `docs/verification/` — the bitvector, bounded-string and bounded-sequence fragments, the verdict-distribution harness and its corpus schema.
- `docs/security/v1.9.2-baseline-allowlist.md` — the default-deny allowlist rationale and class-to-syscall map.
- `docs/adr/0001-syscall-layer.md` — the syscall-layer architecture decision.
- `v1_7/INTEGRATION-hotl.md` — using the progress verifier as an unattended-startup gate.
- `SECURITY.md` — supported versions, private vulnerability reporting, and the security contact (kenneth.douglas@soberagents.ai).

---

8 Roadmap — shrinking UNKNOWN without weakening soundness

The v1.10/v1.11 verification program has one goal: move cases out of UNKNOWN into a provable SATISFIED or UNSATISFIED, raising the clear rate on safe actions, under a soundness obligation that forbids ever turning an unsafe action into SATISFIED. Its first three releases have shipped:

1. **v1.13.0** — the SMT decision procedure in the enforcement path, the bitvector flag fragment, the prefix and equality part of the bounded string fragment, and the verdict-distribution harness with its `unsafe_satisfied == 0` gate.
2. **v1.14.0** — the rest of the bounded string fragment: suffix, contains and glob matchers, exact rule reachability with witnesses.
3. **v1.15.0** — certificates for every authorization, checked by an independent checker.

Next, stated as direction and not claimed as shipped:

- **Host names without agent DNS (v1.24.0, planned).** Policies name hosts (`allow host api.example.com:443`); the Warden resolves the allowed names itself and answers the agent's `/etc/hosts` lookup with a view of only those names, so the agent never sends a DNS query, a name the policy does not allow does not resolve, and DNS cannot carry data out. The decision is made on `name:port`, certified like any host rule, and recorded with the resolution it relied on.
- **Bounded sequence fragment (v1.11, candidate).** Element-level reasoning for the cross-action data-flow subsystem, composed on top of the string and bitvector fragments so its soundness inherits theirs.

- **A customer baseline** for the verdict-distribution harness, on a design partner's workload and policy.
- **The v1.10 architecture track:** race-free filesystem enforcement and trusted-base reduction via a kernel-native sandbox (Landlock); acquisition-time mediation tiering with a latency gate; an optional post-grant re-mediation mode; an escalation ladder for UNKNOWN at the kernel edge; and a trusted base reduced to a minimal supervisor, a small proof checker and the kernel.
- Expanded information-flow coverage including implicit flows; surface-language ergonomics for `var::`; and continued external audit and independent assurance engagement.

External validation context: the DARPA/NSF AI Forge program (June 2026) names provably secure-by-construction agent sandboxes with verifiable action and information-flow bounds and low-latency runtime intervention as a national priority — the problem class VAREK's shipped architecture addresses. This is cited as third-party validation of the problem, not as a claim of program involvement.

---

9 Licensing and intellectual property

VAREK is released under the **MIT license**. Three provisional patent applications are on file covering the formal-verification (SMT decision procedure) layer, the Warden kernel-level enforcement architecture, and action-graph compositional policy decision. The project is **patent-pending**; nothing in this release is granted. Non-provisional conversions begin in 2027. Relicensing considerations are deferred pending conversion.

A Appendix A — Version history

VERSION	FOCUS
v1.0 (Apr 2026)	Public launch. Formal-verification layer, MIT license.
v1.1	Security release: pluggable isolation backend; subprocess-boundary fix.
v1.4	The Warden: a seccomp user-notify supervisor in C (kernel-boundary enforcement).
v1.5	Fast-path matcher and SMT feasibility study for the decision layer.
v1.6	Pre-execution verification of agent action-graphs; compositional three-state decision.
v1.7	Kernel-boundary integration; vertical stack from system boundary to formal decision.
v1.8	Cross-action data-flow verification; audited declassification (v1.8.0); bounded-refusal breaker (v1.8.2).
v1.9	Progress-safety verification. Load-time liveness proof; certified human-out-of-the-loop.
v1.9.1	Enforcement hardening. io_uring refused; TOCTOU-safe file-open mediation; connect and execve deny-only; threat model and trusted-computing-base published.
v1.9.2	Mediation completeness. Default-deny allowlist; native-ABI lockdown (x32 closed); hard-deny set; scalar-flag `CLONE_NEWUSER` denial; conformance target.
v1.9.3	Lifecycle coupling in the live Warden. PID namespace, death-signal coupling, fork-race guard, process-descriptor watch, crash test.
v1.12.0	Mediation correctness. Resolve-then-decide; escaped records carrying the resolved object; datagram egress refused; CycloneDX 1.6 evidence export.
v1.12.1	Side-effect-free denial (`O_PATH` pin, then open); authenticated verdict stream (run id, contiguous sequence); inbound networking refused;

VERSION	FOCUS
	agent in its own network namespace.
v1.12.2	Threads and child processes (`clone3` answered `ENOSYS`); launch exec granted once per run.
v1.12.3	Dynamically linked agents: symlinks followed and decided as their targets; `/proc/self` mapped to the agent.
v1.12.4	The `--plan` gate decides file opens on the lexically canonical declared path.
v1.13.0	SMT decision procedure in the enforcement path (first release of the v1.10 program). Bounded strings and bitvectors; read-only flag clauses; load-time rule reachability; solver cross-check; verdict-distribution harness (85.4% of safe opens cleared, 0 unsafe, on the seed corpus).
v1.14.0	Bounded string fragment: exact, suffix, contains and glob matchers on the resolved path; exact reachability with witnesses; derivative-based oracle.
v1.15.0	Certificates: every authorization checked by an independent checker before it takes effect; `tools/varek_audit.py`.
v1.16.0	Hash-chained, Ed25519-signed, anchored verdict stream; 4,096-token glob cap bounding every decision.
v1.16.1	Deployment preflight (`tools/varek_preflight.sh`); CI build with libseccomp and libsodium.
v1.16.2	Off-host anchor: forwarder and append-only receiver.
v1.16.3	The Verdict Service's plan checker in the repository; escaped JSON output.
v1.17.0	Protected objects refused by identity; raw devices refused; `stat`/`access`/`readlink` decided; agent runs unprivileged (`--run-as`, default `nobody`).
v1.18.0	The claims and the code agree. Plan gate runs the data-flow check, breaker and progress-safety proof; URL-host flow rules; signed, schema-tested CycloneDX export; `EACCES` in records; figures re-measured or restated.

VERSION	FOCUS
v1.19.0	A refusal limit per session (session_refusal_budget).
v1.20.0	Fields on plan steps; trust_declared_fields.
v1.21.0	Decided connections: each connect decided on the destination copied once, dialed by the Warden and handed to the agent; TCP, connected UDP, Unix sockets; tested with curl, Python and Node.js.
v1.21.1	**Plan steps say how they open a file (open=), so the plan gate can authorize declared reads on read-only paths.** Corrects the sample plan and the data-flow threat model.
v1.22.0	The varek command (doctor, init, run, refusals, audit, export, ...) and varek bench, which measures the cost of mediation per call on the host it runs on and checks every verdict.
v1.23.0 (planned)	VAREK Enterprise on AWS Marketplace.
v1.24.0 (planned)	Host names without agent DNS.
v1.11 (candidate)	Bounded sequence fragment for cross-action data flow.

B Appendix B — Corrections to earlier editions

Earlier editions of this paper (v1.9.1 through v1.12) made statements a later review of the code did not bear out. They are corrected in the text above; this list records what changed.

1. **UNKNOWN diagnostics and resource bounds.** Specified for v1.9.1 but not implemented then. §3.2 describes what exists, from v1.13.0. (Corrected in v1.18.0.)
2. **TOCTOU figure.** "510 leaks in 20,000 attempts" has no recorded run. The v1.18.0 re-run gave 1,848 to 1,889 leaks for approve-then-continue and 0 for resolve-and-inject. (Corrected in v1.18.0.)
3. **io_uring test.** v1_7/tests/test_v191_io_uring.c does not check io_uring under the Warden filter on its own; test_v14_filter does, from v1.18.0. (Corrected in v1.18.0.)
4. **Data-flow, breaker and progress-safety in the Warden.** Through v1.17.0 these were a library with tests that the Warden did not call; the --plan gate runs them with --flow-policy from v1.18.0. (Corrected in v1.18.0.)

5. **Connect deny-only.** True from v1.9.1 through v1.20.0; from v1.21.0 connects are decided and dialed by the Warden. (Corrected in v1.21.0 and v1.21.1.)
6. **`/proc/self`.** It is an ordinary symlink, not a magic link, and disabling magic-link resolution does not refuse it. The no-symlinks flag refused it in v1.12.0 to v1.12.2; from v1.12.3 a leading `/proc/self` is mapped to the agent and any other process's `/proc/<pid>` is refused after resolution. (Corrected in v1.21.0.)
7. **Benchmark figures.** A v1.4 P99 of 57 µs is 44 µs in the recorded run; "zero false negatives" is a count of 0 in one benchmark, not a proof. (Corrected in v1.18.0.)
8. **Decision latency.** The v1.21.1 edition gave the median across all decisions as 8 µs "including the notification round trip". That figure, from the v1.14 to v1.16 release notes, is the Warden's own time (its clock starts when the notification is received, so the round trip is not in it), over the v1.4 `bench_target` mix, in which every connect was refused with nothing dialed. §2 now gives `varek bench` figures per kind of call, both the Warden's time and what the agent waits. (Corrected in v1.22.0.)

Copyright Sober Agentic Infrastructure, Inc. VAREK is open source under the MIT license.