PRODUCT DOCUMENT · SECURITY

# Security: what an agent can reach, and what you can prove afterwards

An agent is treated as untrusted even when it is not malicious. Five independent controls bound what it can reach, a person approves every merge, and a signed record proves what happened.

PUBLIC EDITION

| 5 | 0 | Closed | Every |
|---|---|---|---|
| independent controls on every agent session | merges an agent can make | network egress by default | refusal named and recorded |

FOR

a CISO, a security architect, an auditor and the platform team that runs it

PUBLISHED

5 November 2026

Saphan Studio · saphan.ai/product

CONTENTS


01 Summary

02 The risk, stated plainly

03 Five controls, each independent

04 Confinement

05 Egress: the network is closed by default

06 Identity, signatures and the human gate

07 Secrets

08 Recording and guardrails

09 Supply chain and integrity

10 What this document does not claim

11 The Saphan ecosystem and this document set


## HOW TO READ THIS DOCUMENT

Section 1 gives the four decisions behind the design and section 2 the risk. Section 3 lists the five controls and what each does not do; sections 4 to 9 describe them one by one. Every limit is named where it applies.

SECTION 01

## Summary

An agent that writes code on your infrastructure is a program you did not write, running with your credentials, against your repositories. Saphan Studio treats it that way. Its security rests on four decisions:

| DECISION | IN PRACTICE |
|---|---|
| **A person decides, always** | The engine routes work and checks it. It never approves a merge, and a timeout is never consent. |
| **Nothing is trusted because it says so** | Machines are admitted by a person. Every order an agent receives carries a person's approval. A machine's claim about what it can do is graded by where the claim came from. |
| **The record is the system** | Every act is written to a signed, append-only record. Every screen, report and export is a view of it. |
| **When a control cannot be enforced, the run does not happen** | A missing sandbox, an unreadable policy, an unadmitted machine or an ambiguous target is a refusal with a named reason, never a quiet fallback to a weaker posture. |

Where a control ships with a limit, the limit is named next to it, with what to do about it: `SHIPPED, WITH LIMITS`.

SECTION 02

## The risk, stated plainly

If you run continuous integration, you know most of the shape: a control plane sends work to machines, and the machines run code. Agents differ in three ways.

- **The instructions are not known in advance.** An agent decides its next command from what it sees. Controls that depend on reading a command list beforehand do not apply.
- **The agent is persuadable.** Text it reads (a source file, a README, an error message, an issue comment) can change what it does. **An agent is treated as untrusted even when it is not malicious.**
- **The agent is helpful about obstacles.** Blocked, it tries to route around the block, with no bad intent.

**So the question is not "will the agent behave?" but "what is reachable from where the agent runs, and who can prove what happened afterwards?"**

Throughout, Saphan Studio separates controls enforced by a kernel or a database from controls that ask the agent to cooperate. The second kind is documentation, not security.

SECTION 03

## Five controls, each independent

| CONTROL | THE QUESTION IT ANSWERS | WHAT IT DOES NOT DO |
|---|---|---|
| **Confinement** | what the session may read and write on the machine | it bounds the filesystem, not the network |
| **Tool policy** (permission classes) | which commands the session may start | it bounds what starts, not what a command then does: a policy that admits a shell admits everything a shell can reach |
| **Egress** | which hosts the session may reach | it is one layer: your network boundary remains yours |
| **Credential isolation** (seats) | whose keys the session holds | it separates identities, not intentions |
| **The human gate** | what may be merged, and by whom | it is a decision on evidence, so the evidence must be there: that is what the record is for |

**Read the five as a whole. A deployment that adopts three of them is not “mostly protected”: it is unprotected along the two it skipped.**

SECTION 04

## Confinement

- **Every agent runs in its own working copy**, under the engine’s confinement: Landlock and bubblewrap on Linux, Seatbelt on macOS. It can write only where its order allows.
- **Separate seats for agents that share a machine.** Each seat has its own configuration home, so two agents never share sessions, tokens or history.
- **A model called in process has no agent process to confine.** Under closed egress it is refused; the product’s own agent loop reaches the same endpoints, confined.
- **Refusal rather than degradation.** A machine that cannot enforce confinement refuses the run, by name.

| AGENT | LINUX | MACOS |
|---|---|---|
| **Claude Code** | confined by the engine | confined by the engine |
| **Codex · Qwen Code · Factory Droid** | confined by the engine | `SHIPPED, WITH LIMITS` run in their own sandbox; the record says so. To confine them by the engine, run them on a Linux runner or the Solo runner container |
| **Cursor** | confined by the engine | `SHIPPED, WITH LIMITS` confined by the engine on the machine Saphan Studio runs on; on a remote Mac not confined by the engine, and the record says so |
| **Saphan’s own agent loop** | its own confinement | its own confinement |

SECTION 05

## Egress: the network is closed by default

- A run reaches only its agent's model hosts and the package registries its class declares. Everything else is refused, and the refusal is recorded.
- A wider policy is a written document a person approves. A run is opened beyond it only by a person, with a recorded reason and actor.
- A machine that cannot enforce the egress rules refuses the run rather than running it open.
- Saphan Studio makes few outbound connections of its own, each named in the operator documentation.

> Example from the record: a minute into its task, an agent ran `git clone` against a public repository nobody had asked for. The egress gateway answered 403, the run stopped, and the refusal went into the record with the rule that refused it.

SECTION 06

## Identity, signatures and the human gate

- **People and agents act under one registry** with delegated, time-bounded, scoped authority. Nothing gains permission by the passage of time. Retiring an actor takes effect at once.
- **Machines are admitted by a person**, bound to a certificate, and graded by what they can prove about themselves.
- **Orders, gates and evidence are signed**, and every signature is checked when the document is opened.
- **Approvals are a person's act**, confirmed with a presence code or the person's own passkey. No agent can approve or merge; the permission is denied to every agent at the grant. `SHIPPED, WITH LIMITS` the approval travels with the merge as commit trailers, not as a git signature
- **Sign-in is standard OAuth 2.1 / OIDC**, and fail-closed: an identity without an explicit grant reads zero rows.

SECTION 07

## Secrets

- **Configuration, the record and the evidence carry names, paths and fingerprints, never secret values.** A credential is configured as the name of an environment variable or the path to a file.
- **Key files are under enforced custody**: owner-only permissions, checked before the contents are read.
- **Evidence is redacted before it is written**, by known patterns and by an entropy threshold that catches token-shaped strings no pattern knows.
- **Remote runs get a minimal environment**, never the control plane's.
- **A seat's vendor API key** is kept owner-only in that seat's profile and given to its agent at run time. `SHIPPED, WITH LIMITS` it is not hidden from other seats' agents on the same machine; give seats with different keys their own machines or operating-system users

SECTION 08

## Recording and guardrails

- **Every run is recorded by default.** Who, what and where in the run record; the agent's output, redacted, in a live journal; every exchange with the model in a hash-chained transcript. A run that was not recorded says so, with the reason.
- **The guardrail** is a local proxy between the agent and its model. It always records. With a plugin attached, it judges each request and response (allow or block) in observe or enforce mode. It never rewrites a message: it forwards the original or stops the call.
- **Refusals are a metric.** Prometheus counts refusals by class, because on this architecture a refusal is a security signal, not noise.

SECTION 09

## Supply chain and integrity

- A machine receives its payload only after a person admits it, and verifies the checksums before anything lands.
- The control binary checks its own custody.
- Skills reach agents with a fingerprint and signature anyone can verify (Skills Studio).
- Dependency changes are judged by the supply-chain skills for Go, npm and Python, and read by the reviewer as a separate act.

### Reporting a vulnerability

Write to security@saphan.ai. The operator documentation describes the process.

SECTION 10

## What this document does not claim

- It is not a certification. Standards certify an organisation's management system; Saphan Studio produces the evidence your auditors ask for.
- No control here replaces your network boundary, your host hardening or your own review of what you admit.
- A tool policy is only as strict as the commands it admits. Admitting a shell admits everything reachable from a shell.
- Agent vendors change their tools and sandboxes. Each run records the version it ran with.

The full operator chapter, with every limit and the commands to check each control yourself: docs.saphan.ai/docs/security.

SECTION 11

# The Saphan ecosystem and this document set

Saphan Studio is one product in a small ecosystem. The tools around it are open source under Apache 2.0 and each works on its own; together they cover a change from the first read of the code to the evidence that it is right.

`Discover reads the code` → `Skills Studio checks the skills` →
`Saphan Studio runs the agents under a person's approval` → `Mocks Studio stands in for real systems` →
`Tests Studio and Proven judge the result`

| TOOL | ITS PART |
|---|---|
| **Saphan Studio** saphan.ai | The control plane: signed orders, confined agents, independent review, human approval, one signed record. |
| **Discover** github.com/saphan-ai/discover | Reads a codebase before anyone changes it, with a source for every claim; proposes the skills that fit it. |
| **Skills Studio** github.com/saphan-ai/skills-studio | Checks and signs the skills agents are given, so anyone can verify them again. |
| **Mocks Studio** github.com/saphan-ai/mocks-studio | Stand-in systems agents and tests work against instead of the real ones. |
| **Tests Studio** github.com/saphan-ai/tests-studio | Shows what a test run really proved, in the framework the team already uses. |
| **Proven** github.com/saphan-ai/proven | An AI code reviewer that reports a problem only with evidence. |

## The document set

| DOCUMENT | WHAT IT ANSWERS |
|---|---|
| **Saphan Studio for developers and DevOps** | start here: what Saphan Studio is and how the pieces fit |
| **Skills and craft** | what every agent carries into the work |
| **Testing and review** | how a change earns its merge |
| **Orchestration rules** | a deterministic engine, model work inside it |
| **Security** (this document) | what an agent can reach, and what you can prove afterwards |

For partners: a founding customer onboarding guide and an Enterprise continuity plan, at saphan.ai/for-partners.

All documents: saphan.ai/product · Code quality against published baselines: saphan.ai/research/proven-effort · Operator documentation: docs.saphan.ai · Contact: hello@saphan.ai