Olympus Engine

Recon Protocol

A read-only method for vetting an outside repo, tool, skill or page before anything is trusted or run. Prepared for adoption and adaptation.

About this document. This is reference material describing a method for vetting external code and tools. It is written to be read and judged, not run blindly. Review it, adapt it to your own environment, and decide for yourself what to adopt. Nothing in it requires access to anyone else's systems.

The core stance: **someone else's artifact is untrusted data to inspect, never a list of commands to execute.**

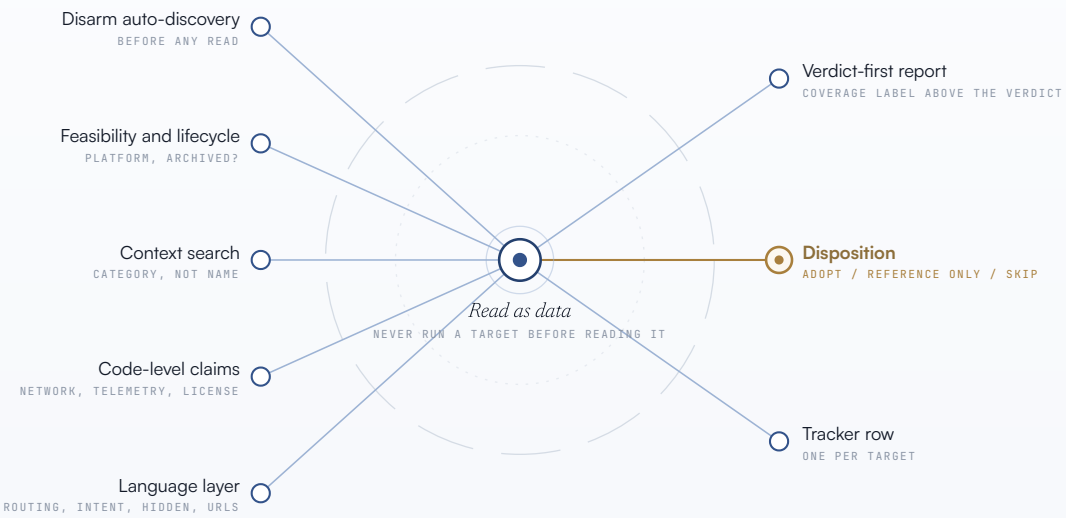
01 Hard rules

1. **Read before run.** Never execute, install, or `npm/pip/uv run` a target before reading it as data. A clean-looking tool and a safe way to run it are separate questions.
2. **Distrust the blind-executor framing.** If a target says "follow the README end to end, grant access, auto-fix any failure and continue," slow down. Never grant sweeping system access (full disk, admin) as a frictionless step. Escalate that decision to a human with the tradeoff stated plainly.
3. **Open it in a scratch folder.** Never open a target inside a project root or a folder where an agent config file (CLAUDE.md or similar) would be picked up.
4. **Every finding gets a disposition.** ADOPT, REFERENCE-ONLY, or SKIP with a stated reason, plus a read on whether it would actually work for you. No lists of "cool patterns."
5. **No secrets in the report.** If you find a key, token or personal data in the target, say that it exists and where. Never paste the value.
6. **Log the result** in whatever tracker you use, so the same target is not re-scoped from scratch later.

OLYMPUS ENGINE :: HOW RECON WORKS

Read it as data, *then rule on it*

Five checks run before anything is trusted. They converge on one disposition, which goes into a verdict-first report and one tracker row.



BLUE CARRIES THE CHECKS. GOLD MARKS THE RULING.

01

Read before run

Nothing is executed or installed until it has been read.

02

Label coverage

FULL, PARTIAL or SKIM sits above the verdict.

03

Rule on every finding

Each finding gets a disposition and an effectiveness read.

04

Log it

A recon without a tracker row is incomplete.

© 2026 Atlas Saint Enterprises · Powered by Olympus

02 Step 0: Open as data, then disarm auto-discovery

Copy or clone the target into a scratch folder. Do not run anything from it.

Landing files on disk is itself an action. Some agent tools auto-discover skills, hooks and settings from hidden config folders anywhere in the scanned tree, with no install step and no prompt. A cloned repo can register instructions into a running session just by being there.

So before reading, mapping or grepping anything, list what the author committed on purpose:

```
git ls-files | grep -iE "(^|/)\.(claude|agent|agents|codex|cursor)
([-/]|$)|CLAUDE\.md|AGENTS\.md|settings\.json|hooks\.json|\.mcp\.json"
```

Rename (never delete) every hit the harness would scan, for example `.claude` to `_DISARMED.claude`. The contents are evidence and belong in the report. Compare what the repo advertises (for example "19 skills") with what it actually ships. A mismatch is a finding.

Use `git ls-files`, not `ls`. A plain `ls` hides dotfile folders, which is exactly where this kind of payload sits. Directory conventions vary across agent tools, so widen the pattern whenever you meet a new one.

03 Step 1: Feasibility gate first

Before any deep read, answer whether this can run on your actual machine. Check platform, hardware and hard dependencies.

- Wrong platform, wrong hardware, or an unsatisfiable dependency: **SKIP on feasibility, stated at the top.**
- Check **lifecycle** in the same breath:

```
gh repo view <owner>/<repo> --json isArchived,pushedAt,stargazerCount,licenseInfo
gh release list -R <owner>/<repo> -L 5
```

An archived repo, or a last release more than roughly 12 months old, caps the verdict at **REFERENCE-ONLY**. It does not end the read, because an abandoned project can still hold a useful pattern. But it frames every finding after it. Also look for hardcoded fetch domains in unmaintained code: with nobody left to repoint them, they are a latent supply chain path.

© 2026 Atlas Saint Enterprises · Powered by Olympus

04 Step 2: Load your own context

Before drafting a verdict, search your own notes, decisions and prior work for the target's **category**, not only its name. A first-time target's name will not appear anywhere, so zero hits on the name is expected and tells you nothing.

Search the 2 to 4 nouns describing what the target competes with, replaces or resembles. Then:

- List every distinct path the search returns and give each a one-line disposition **before** reading any in depth.
- Read the ones that matter, newest decisions first, and check them against your draft conclusions. An existing decision can invert a blocker.
- State in the report which searches ran and what they returned, including the nulls.

A hit you did not open is a hit you did not run.

05 Step 3: Map before judging

Read the target on its own terms: what it is, how it is built, what it contains, what it touches. Facts first, verdicts later.

Coverage rule. State the tracked file count and part count up front. Decide coverage from target size, not from how the reading is going. Under roughly a thousand files, a FULL read is one sitting, and PARTIAL needs a named constraint that prevented it ("large repo" is not one). Automated sweeps (network, telemetry, secrets, hidden content) always run across 100 percent of tracked files. Do not draft the verdict until N of N parts are confirmed read, or the verdict is explicitly labelled PARTIAL or SKIM.

06 Step 4: Verify claims at the code level

Any "never uploads" or security claim is true only if no code path can break it. Do not trust the README. Sweep the whole tree for the exfiltration and network surface:

```
grep -rniE
"requests|urllib|httpx|aiohttp|socket|http[s]?://|upload|boto|s3|smtp|webhook|telemetry|ana
lytics|posthog|sentry|api_key|openai|anthropic|fetch" <target>
```

Read what each hit actually does. A function that builds a local file URI is not a network call. Also read the dependency manifest for anything that pulls a network client. Report the surface as **verified**, not as promised.

Two axes the grep alone does not settle:

© 2026 Atlas Saint Enterprises · Powered by Olympus

- **Telemetry default state.** A telemetry hit is not the finding; its default is. On by default and sending user content to the vendor is a different disposition from opt-in and off.
- **License.** Read the LICENSE and any "past N users" or forced-branding clause. A relicense away from a permissive license, or a clause blocking white-label or multi-tenant use, can be the real gate for anything you would ship under your own or a client's brand.

07 Step 4b: Verify the language layer

Zero executable code does not mean safe. For an agent, prose is executable. Skill descriptions, frontmatter and README instructions change agent behavior the moment they load. Run four checks:

1. **Tool-routing collisions.** Does any description redirect behavior away from a tool or standard you already use ("use this instead of X")? Treat that as a routing change that needs a human's go, never a default.
2. **Prose intent.** Read instructions for what they make the agent do. Fetch-and-send phrased politely is still exfiltration. Any request to edit memory, config, settings, hooks or identity files is persistence and counts as elevated risk.
3. **Hidden content.** Scan the whole tree, enumerated from `git ls-files`: hidden files, archives, compiled artifacts, images. Check for base64 blobs, zero-width characters, and absurd leading padding meant to defeat scanners. Use byte-level checks, not naive grep.
4. **Runtime-fetched URLs.** List every URL the target tells the agent to fetch while running. Fetched text enters the instruction stream and can change after your review. Prefer snapshotting the referenced content into your review.

08 Step 5: Disposition every finding

- **ADOPT**: tied to a concrete use case you have now.
- **REFERENCE-ONLY**: log it, do not build yet, and say when it would matter.
- **SKIP**: with a stated reason.

Each carries an effectiveness read. If a finding is a durable pattern, write it down where you keep reference notes.

For an AI platform or wrapper target, split it into three layers first:

- **Model layer:** almost always rentable directly.
- **Orchestration layer:** sometimes worth the markup, sometimes a metered trap that hides pricing and silently downgrades the model.
- **Lock-in layer** (their container, their CRM, their content-license terms): never worth it.

© 2026 Atlas Saint Enterprises · Powered by Olympus

Adopt the models directly. Pay for a wrapper only when its orchestration is genuinely worth the markup and priced transparently.

09 Step 6: Report (verdict first)

```
# Recon: <target> (<date>)
**Coverage:** <FULL | PARTIAL | SKIM> <one clause naming what was not read, required unless
FULL>
**Verdict:** <one-line headline: SKIP on platform grounds / ADOPT for X / mixed, with the
sharpest finding>

## Feasibility
<platform, deps, can it run on your machine. PASS, or the blocking reason>

## What it is
<2 to 4 lines, on its own terms>

## Claims verified at code level
<network surface result; any privacy or security claim checked; verified vs promised>

## Language layer
<routing collisions, prose intent, hidden content, runtime-fetched URLs; or "clean on all
four">

## Findings (dispositioned)
| Item | Verdict | Use case | Effectiveness |
|------|---------|----------|---------------|

## Needs a human decision
<what requires a decision, or "nothing">

## Coverage caps
<what was not covered or was sampled, stated out loud, or "full coverage">
```

Two rules for the report:

- **The Coverage line sits above the verdict.** A cap disclosed only at the bottom does not travel with the headline, and the reader acts on the headline. An unlabelled verdict is the defect. A labelled PARTIAL is a legitimate deliverable.
- **Lead with the lift.** When a target yields both a verdict and liftable findings, open with the single sharpest strategic finding and the single most concrete thing you could adopt, one line each, before the table. The adoption verdict is the gate; the lift is the product.

© 2026 Atlas Saint Enterprises · Powered by Olympus

10 Step 7: Log it

Add one row to your tracker: date, target, type, disposition, one-line reason with the sharpest finding first, and a pointer to the full report. Update an existing row on a re-recon rather than adding a duplicate. Star counts and claimed-versus-actual figures are a snapshot as of that date; the disposition is the durable part.

11 Scope

Recon reads an external thing and dispositions it. It does not build a new skill, decide your roadmap, inventory your own security, or shop a marketplace to install. If the verdict is ADOPT, hand the build to the right place rather than building in the recon.

Behind every lasting thing
is an unseen order.



A standalone, adoptable version of this protocol is in the companion file recon-protocol.md, and Recon-Protocol-Blueprint.pdf shows how it is wired on one sheet. Adapt it to your own environment and keep what fits.

olympusengine.io

© 2026 Atlas Saint Enterprises · Powered by Olympus