

astrapatch.

• CONTRIBUTOR DOCUMENTATION

Contribution Guide

How to contribute code, adapters, research and documentation to astrapatch, and the standards that keep it credible.

Building astrapatch, together.

This guide explains how to contribute code, adapters, research and documentation to astrapatch, and the standards that keep the project credible to the engineers who rely on it.

01 Welcome

02 Joining the project

03 Principles

04 Ways to contribute

05 Development setup

06 Repository layout

07 Contribution workflow

08 Code standards

09 Writing plugins

10 Policy rules

11 Documentation and accuracy

12 Commits and pull requests

13 Security, conduct and license

14 Pre-submission checklist

Secure software. Beyond Earth.

astrapatch is an open-source framework for spacecraft software security. It finds vulnerabilities in flight code, proposes candidate patches, and verifies each one against mission constraints before an engineer reviews it.

The project is built around one question: **can we remediate a software vulnerability while preserving the behavior and operational constraints of the spacecraft?** Every contribution, from a new analyzer to a corrected sentence in the docs, should move that question forward.

Project status: pre-alpha (Phase 01, Foundation). The pipeline runs end to end on the included demo application. Nothing astrapatch produces is evidence of flight safety or a substitute for mission-specific qualification.

Contributors from aerospace engineering, embedded security, software verification and AI research are all welcome. You do not need to be an expert in all of them; most good contributions start from one.

Request contributor access.

To join astrapatch as a contributor, send a short introduction to the maintainers. Approved contributors are invited to the astrapatch GitHub organization.

```
EMAIL    contribute@astrapatch.online
SUBJECT  Contributor access: <your GitHub username>
```

What to include

ITEM	DETAILS
GitHub username	Required. This is the account that will be invited to the organization.
Name and affiliation	Your name, and your university, lab or company if you have one. Independent contributors are welcome.
Area of interest	Analyzers, remediation, simulation and scenarios, policies, research, or documentation.
Relevant experience	Links where possible: repositories, papers, flight software or security work.
Why astrapatch	Why you are interested in the project and what you would like to build or improve.

Requirements

- A GitHub account with two-factor authentication enabled.
- You have read this guide and the Code of Conduct, and agree to follow them.
- You agree that your contributions are licensed under the Apache License 2.0.
- You only submit work you have the right to contribute.

Requests are reviewed by the maintainers, who reply by email.

What every change must respect.

These principles are the project's design constraints. A change that weakens one of them will not be merged, however useful it looks.

P1

Patches are untrusted

Whatever produced a candidate, a rule or a model, it must pass verification and human review. Nothing is trusted because of where it came from.

P2

Policies are deterministic

Acceptance rules give the same answer for the same evidence. Model output can inform engineers but can never override a rule.

P3

Unknown is not pass

A missing metric, a scenario that did not run or a check without data makes the verdict inconclusive. It is never rounded up.

P4

Evidence over confidence

Every result is backed by artifacts a reviewer can inspect and reproduce: logs, hashes, diffs, outputs.

P5

Humans approve

astrapatch classifies candidates for review. It has no deploy step, and contributions must not add one.

P6

Local first, no lock-in

Analysis and verification run offline. No dependency on a single model, cloud provider or vendor tool.

P7

Wrap, do not reinvent

Adapters integrate existing analyzers and simulators rather than reimplementing them.

P8

Say only what is true

Never describe planned work as shipped, and never claim certification, endorsement or flight heritage.

Where help matters most.

AREA	EXAMPLES	START WITH
Analyzer adapters	Wrap a static analyzer, parse its output into normalized findings, map severities and CWEs.	<code>docs/plugins.md</code>
Remediators	Deterministic fixes for well-understood patterns, or a command bridge to a model you run.	<code>remediators/rules.py</code>
Simulators and scenarios	Scenario runners, packet replays, fault injection, metric reporting.	<code>examples/demo-spacecraft</code>
Policy rules	New deterministic rule kinds for timing, interfaces, resource budgets.	<code>verification/policy.py</code>
Test cases	Fixtures that prove an analyzer finds, or correctly ignores, a pattern.	<code>tests/</code>
Research	Benchmark datasets, reproducibility packages, remediation experiments.	Open an issue first
Documentation	Clarify setup, configuration and architecture; fix anything inaccurate.	<code>docs/</code> , <code>README.md</code>

Finding something to work on

- Browse open issues, or propose an adapter with the **Adapter proposal** issue form.
- For anything larger than a small fix, open an issue before writing code so the approach can be agreed.
- Comment on an issue to say you are working on it, so effort is not duplicated.

Running astrapatch locally.

You need **Python 3.11 or newer**. The end-to-end tests and the demo also need `make` and a C compiler. `cppcheck` is optional; its adapter is skipped when it is not installed.

```
# Clone and install in editable mode with development tools
git clone https://github.com/astrapatch/astrapatch.git
cd astrapatch
python -m venv .venv && . .venv/bin/activate
pip install -e ".[dev]"

# Run the checks CI runs
pytest
ruff check .
ruff format --check .
```

Try the full pipeline

```
cd examples/demo-spacecraft
astrapatch analyze
astrapatch remediate --finding FINDING-001
astrapatch verify --candidate candidate-001
astrapatch report --candidate candidate-001
```

Or run `astrapatch` with no arguments to open the interactive shell and drive the same steps with `/analyze`, `/fix`, `/verify` and `/report`.

Artifacts land in `.astrapatch/` inside the project, which ignores itself in git. Delete the folder at any time to start fresh.

Where things live.

PATH	CONTENTS
<code>src/astrapatch/analysis.py</code>	Runs analyzers, imports SARIF, normalizes and de-duplicates findings
<code>src/astrapatch/analyzers/</code>	Built-in pattern analyzer, cppcheck adapter, SARIF import
<code>src/astrapatch/remediators/</code>	Rule-based and external-command remediators
<code>src/astrapatch/verification/</code>	Runner, policy engine and verification orchestration
<code>src/astrapatch/sbom.py</code> , <code>advisories.py</code>	CycloneDX SBOM and offline OSV correlation
<code>src/astrapatch/report.py</code>	Patch manifest, summary and evidence package
<code>src/astrapatch/cli.py</code>	Command-line interface
<code>examples/demo-spacecraft/</code>	Deliberately vulnerable demo application, scenarios and policy
<code>tests/</code>	Unit and end-to-end tests
<code>docs/</code>	Architecture, configuration and plugin guides

The weaknesses in `examples/demo-spacecraft/src` are intentional. Do not fix them in place; they exist so the pipeline has something real to find and verify.

From idea to merged change.

01	02	03	04	05
DISCUSS	BRANCH	BUILD	VERIFY	REVIEW
Open or pick an issue. Agree the approach for larger changes.	Fork, then branch from <code>main</code> with a short descriptive name.	Make one focused change with tests and docs.	Run <code>pytest</code> and <code>ruff</code> . Check behavior on the demo.	Open a pull request using the template. Respond to feedback.

```
git switch -c adapter/flawfinder
# ...make your change, add tests...
pytest && ruff check . && ruff format --check .
git commit -m "Add flawfinder analyzer adapter"
git push -u origin adapter/flawfinder
```

Review expectations

- A maintainer reviews every pull request; code owners are requested automatically.
- CI must pass on Python 3.11, 3.12 and 3.13, including the demo pipeline job.
- Reviewers check correctness, tests, and whether any claim in code or docs goes beyond what the change does.

How astrapatch code is written.

STANDARD	WHAT IT MEANS
Python 3.11+	Use modern syntax: <code>X None</code> , <code>StrEnum</code> , <code>tomllib</code> .
Zero runtime deps	The core installs with nothing but the standard library. Propose any new dependency in an issue first.
Ruff	Code passes <code>ruff check</code> and <code>ruff format</code> with the project configuration. Line length is 110.
Type hints	Public functions and plugin interfaces are fully annotated.
Determinism	Same inputs, same outputs. Sort anything written to disk; never depend on dictionary or file-system order.
JSON artifacts	Everything a stage produces round-trips through JSON so it can be audited without the tool.
Failure isolation	An adapter that fails reports its error and lets the run continue; it never hides other results.
Tests	Every behavior change comes with a test. Tests that need a toolchain are marked to skip cleanly without one.

Error messages

Say what went wrong and what to do next, for example: `No astrapatch.toml found. Run 'astrapatch init' to create one.` Remediators that cannot help raise `RemediationUnsupported` with a specific reason instead of guessing.

Analyzers and remediators.

Plugins are Python classes registered through entry points, so they can live in their own packages without changes to astrapatch.

```
class MyAnalyzer(Analyzer):
    name = "mytool"

    def available(self):
        # (False, reason) when the tool is missing; the run continues
        return True, ""

    def analyze(self, ctx: AnalysisContext) -> list[Finding]:
        # read only from ctx.root; return findings without ids
        ...
```

```
# your plugin's pyproject.toml
[project.entry-points."astrapatch.analyzers"]
mytool = "my_plugin.analyzer:MyAnalyzer"
```

Rules for adapters

- Set `location.snippet` to the source line so fingerprints stay stable when unrelated lines move.
- Analyzers run on the project and on both verification copies; read only from `ctx.root`.
- Remediators return a minimal, reviewable diff and record how it was produced in `provenance`.
- Include a fixture-based test showing both a true positive and a case the adapter must ignore.

The full interfaces are documented in `docs/plugins.md`.

Pass, fail, or unknown.

Policy rules decide whether a candidate is accepted for review. They are the most safety-relevant code in the project, so they follow the strictest rules.

RESULT	RETURN IT WHEN
pass	The rule had the data it needs and the condition holds.
fail	The rule had the data it needs and the condition does not hold.
unknown	Anything else: missing metric, scenario not run, failed baseline, unparseable input.

Requirements for a new rule kind

- Pure and deterministic: no network, no randomness, no model calls.
- Never returns pass on missing data; add a test proving it returns unknown.
- Explains itself: the `detail` string says what was compared and the result.
- Records the values it compared in `evidence` so reviewers can check them.
- Is documented in the rule table in `docs/configuration.md`.

A verdict of `accepted_for_review` means the configured checks passed. It is not a statement about flight safety, and wording in code or docs must never suggest otherwise.

Credibility is a feature.

astrapatch is a young project in a field where overstatement is dangerous. Accurate documentation is as important as working code.

Do

- Mark work as planned, experimental or available, and keep the status table current.
- Label demo, fictional and illustrative material clearly.
- State limits plainly, such as "runs in copies, not a sandbox".
- Document only what exists; mark drafts as drafts.
- Cite real sources for any research claim.

Do not

- Describe planned features as implemented.
- Add metrics, adoption numbers or benchmarks without real data.
- Claim NASA, ESA or any agency's certification or endorsement.
- Write installation steps for things that do not install yet.
- Imply that passing verification means flight safety.

Writing style

Plain, specific English. Short sentences. Prefer a concrete example over an adjective. Use the same terms as the code: finding, candidate, verdict, scenario, evidence package.

Small, clear, reviewable.

Commit messages

- A short summary in the imperative mood, under about 72 characters: `Add flawfinder analyzer adapter`.
- A body, when needed, explaining **why** the change was made and anything a reviewer should know.
- One logical change per commit. Keep refactors separate from behavior changes.

Pull requests

- Fill in the template: summary, type of change, how it was tested, checklist.
- Link the issue it closes.
- Keep it focused; split large work into a series of pull requests.
- Include sample output for changes that affect what users see.

Licensing of contributions

astrapatch is licensed under the Apache License 2.0. As stated in section 5 of that license, any contribution you intentionally submit for inclusion is provided under the same license, unless you explicitly state otherwise. Only submit work you have the right to contribute.

Working safely and respectfully.

Reporting vulnerabilities

Do not open public issues for security problems in astrapatch. Use GitHub's private vulnerability reporting: open the repository's **Security** tab and choose **Report a vulnerability**. Include the impact, steps to reproduce and the affected commit.

Vulnerabilities found *using* astrapatch in other software belong to that software's maintainers. The deliberate weaknesses in the demo application are out of scope.

Trust assumptions to keep in mind

- Build, test and scenario commands come from the project's configuration and run with the user's privileges.
- Verification copies are not a security sandbox. Do not write code or docs that suggest they are.

Code of conduct

The project follows the Contributor Covenant, version 2.1. Be respectful, assume good faith, welcome people of every background and level of experience, and keep disagreement focused on the work. Harassment is not tolerated.

License

Apache License 2.0. See `LICENSE` in the repository.

Before you open a pull request.

- ☐ The change is focused, and linked to an issue if it is not trivial.
- ☐ `pytest` passes locally.
- ☐ `ruff check .` and `ruff format --check .` pass.
- ☐ Tests were added or updated for every behavior change.
- ☐ No new runtime dependency, or it was agreed in an issue first.
- ☐ Policy rules return unknown, never pass, when data is missing.
- ☐ Documentation matches the change, including status tables.
- ☐ Nothing planned is described as implemented; no unverifiable claims.
- ☐ Demo and illustrative material is labeled as such.
- ☐ The pull request template is filled in, with sample output where relevant.

astrapatch.

contribute@astrapatch.online
github.com/astrapatch/astrapatch
astrapatch.online