

astrapatch.

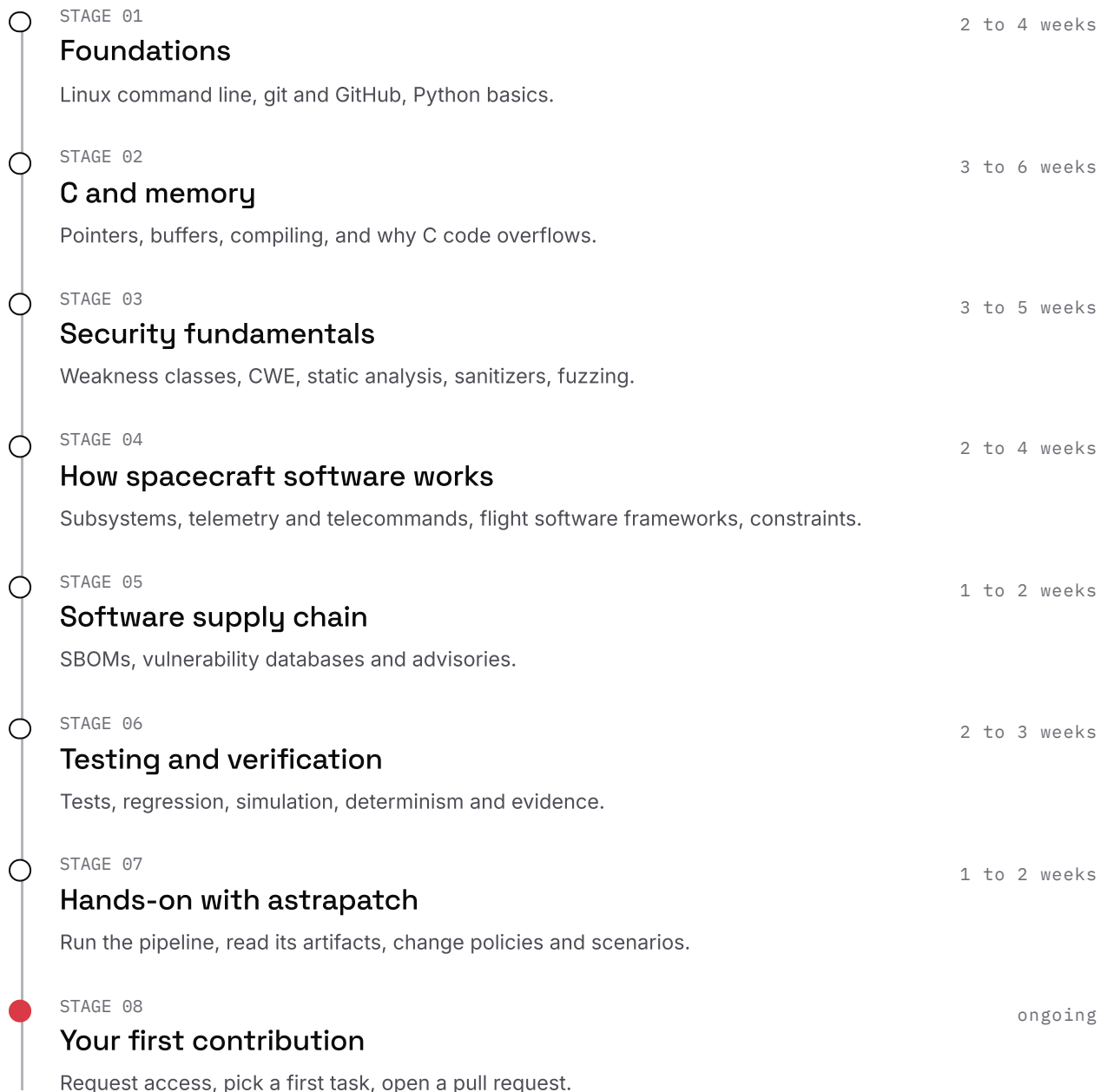
• LEARNING PATH

Beginner Roadmap

A self-paced path into spacecraft software security, from your first terminal command to your first astrapatch contribution.

From zero to your first contribution.

A self-paced path for people who are new to security, spacecraft software, or both. Each stage builds on the last and ends with hands-on work on the real astrapatch demo.



Durations are rough guides for a few hours a week, not requirements. Skip any stage you already know; each one ends with a checkpoint so you can tell.

How to use this roadmap.

Who it is for

- Students and new engineers curious about space cybersecurity.
- Software developers who know a language but not C or security.
- Security people who have never looked at flight software.
- Aerospace engineers who want to understand software vulnerabilities.

How each stage works

Learn

The ideas to understand, and a short list of resources. You do not need to finish every resource; stop when the checkpoint makes sense.

Practice

Exercises on your own machine, most of them on the astrapatch demo. Every exercise in this guide has been run and behaves as described.

What you need

- A computer running Linux, macOS, or Windows with WSL (Ubuntu).
- Python 3.11 or newer, git, `make` and a C compiler (gcc or clang).
- A GitHub account. Turn on two-factor authentication now; you will need it to contribute.
- Curiosity, and a habit of reading error messages slowly.

About the demo. `examples/demo-spacecraft` is a small, deliberately vulnerable C program built for learning. It is a host-side test harness, not real flight software, and its vendored library and advisory are fictional.

A note on ethics

Practice only on code you own or are explicitly allowed to test, like the demo. Spacecraft and ground systems are critical infrastructure; probing systems you do not own is illegal in most countries and can cause real harm.

Tools every contributor uses.

LINUX

GIT AND GITHUB

PYTHON

2 TO 4 WEEKS

Learn

- The shell: moving around, files, pipes, environment variables, exit codes.
- git: commits, branches, pulling, pushing, reading a diff.
- GitHub: forks, issues, pull requests, reviews.
- Python: functions, classes, modules, virtual environments, `pip`.
- Running tests with `pytest`.

Practice

```
git clone
https://github.com/astropatch/astropatch.git
cd astropatch
python3 -m venv .venv && .
.venv/bin/activate
pip install -e ".[dev]"
pytest
echo $? # 0 means every test passed
```

Then open `src/astropatch/cli.py` and find where each command is defined.

CHECKPOINT

You can clone the repository, create a virtual environment, run the tests, make a branch, commit a change, and explain what an exit code is.

Resources

RESOURCE	WHY
Pro Git (free book) git-scm.com/book	Chapters 1 to 3 cover everything you need to start.
The Python Tutorial docs.python.org/3/tutorial	The official tutorial; enough to read astropatch's code.
GitHub Docs: Get started docs.github.com	Forks, pull requests and two-factor authentication.

The language spacecraft run on.

C

POINTERS

BUFFERS

MAKE

3 TO 6 WEEKS

Most flight software is written in C or C++. To find and fix its weaknesses you have to understand how it uses memory.

Learn

- Types, arrays, strings and the terminating zero.
- Pointers, and the difference between a buffer and a pointer to it.
- The stack and the heap; what a local array really is.
- `memcpy`, `strcpy`, `sprintf` and their bounded versions.
- Compiling and linking with `gcc` and a `Makefile`.

Practice

```
cd examples/demo-spacecraft
make && make test
./build/flight_sim
scenarios/nominal.pkts
cat telemetry.out metrics.json
./build/flight_sim
scenarios/malformed.pkts
echo $? # non-zero: it crashed
```

Read `src/tlm.c`. Find the line that copies the payload and explain, in one sentence, why the malformed packets crash it.

CHECKPOINT

You can explain why copying `len` bytes into a 64-byte local array, when `len` comes from the packet, can overwrite the stack.

Resources

RESOURCE

WHY

The C Programming Language
(Kernighan and Ritchie)

Short and still the clearest introduction to C.

Your compiler's warnings

`gcc -Wall -Wextra`

Turn them on for everything you write and read every one.

How software breaks, and how we find it.

CWE

STATIC ANALYSIS

SANITIZERS

FUZZING

3 TO 5 WEEKS

Learn

- Weakness classes and the CWE catalog: CWE-120, CWE-121, CWE-787 for memory, CWE-78 for command injection.
- Static analysis: finding bugs by reading code, and why it reports false positives.
- Dynamic analysis: sanitizers that catch bugs while the program runs.
- Fuzzing: feeding generated inputs to find crashes.
- The difference between a weakness, a vulnerability and an exploit.

Practice

Catch the demo's bug with AddressSanitizer:

```
make clean
make CFLAGS="-O0 -g -fsanitize=address"
./build/flight_sim
scenarios/malformed.pkts
# stack-buffer-overflow, WRITE of size
4000
# in tlm_handle src/tlm.c:44
make clean
```

Then look up CWE-121 and compare its description with what you just saw.

CHECKPOINT

Given a short C function, you can name the likely weakness and its CWE, and say whether a static analyzer or a sanitizer would catch it.

Resources

RESOURCE

WHY

CWE, Common Weakness Enumeration

cwe.mitre.org

The shared vocabulary astrapatch uses in every finding.

AddressSanitizer documentation

clang.llvm.org/docs/AddressSanitizer.html

How the sanitizer you just used works.

cppcheck

cppcheck.sourceforge.io

An open-source C/C++ analyzer astrapatch already integrates.

Why space changes the problem.

SUBSYSTEMS

TELEMETRY

CFS AND F'

2 TO 4 WEEKS

Learn

- Spacecraft subsystems: on-board computer and flight software, power, attitude control, communications, payload.
- Telemetry and telecommands, and how packets carry them; the CCSDS space packet format is the common standard.
- Flight software frameworks: NASA cFS and NASA F' are open source and widely studied.
- Constraints: limited memory and CPU, deadlines, radiation, intermittent links, no physical access after launch.
- Why a patch must preserve timing, memory use and telemetry formats, not just remove a bug.

Practice

- Open `scenarios/nominal.pkts`. Decode one line by hand: which bytes are the APID, sequence count and length?
- Add a new well-formed packet with a 10-byte payload, rebuild, and run the nominal scenario. Check the new line in `telemetry.out`.
- Browse the NASA cFS or F' repository and find one component that handles incoming commands.

CHECKPOINT

You can explain why "it compiles and the bug is gone" is not enough to accept a patch for software already in orbit.

Resources

RESOURCE	WHY
NASA core Flight System github.com/nasa/cFS	Open-source flight software framework.
NASA F' (F Prime) github.com/nasa/fprime	Component-based flight software framework.
NASA NOS3 github.com/nasa/nos3	Spacecraft software simulation environment.
CCSDS published standards public.ccsds.org	Space packet and telemetry standards.
SPARTA sparta.aerospace.org	A framework describing threats and countermeasures for spacecraft.

These are independent external resources. astrapatch is not affiliated with or endorsed by NASA, CCSDS or The Aerospace Corporation.

Knowing what is inside your software.

SBOM

CYCLONEDX

OSV

1 TO 2 WEEKS

Learn

- What a software bill of materials (SBOM) is, and why operators ask for one.
- The CycloneDX and SPDX formats.
- Vulnerability identifiers (CVE) and advisory databases such as OSV.
- Version ranges: how an advisory says which versions are affected and which are fixed.

Practice

```
cd examples/demo-spacecraft
astrapatch analyze
ls .astrapatch/analysis/*/
# findings.json run.json sbom.cdx.json
```

- Open `sbom.cdx.json` and find the `minicrc` component and its hash.
- In `astrapatch.toml`, change the minicrc version to `0.9.3` and analyze again. The fictional advisory disappears, because it is fixed in 0.9.3.

CHECKPOINT

You can read an SBOM, explain how an advisory's version range decides whether a component is affected, and say why this matters for software you cannot easily update.

Resources

RESOURCE

WHY

CycloneDX

cyclonedx.org

The SBOM format astrapatch writes.

OSV

osv.dev

The advisory format astrapatch reads offline.

Evidence, not confidence.

TESTS

REGRESSION

SCENARIOS

DETERMINISM

2 TO 3 WEEKS

Learn

- Unit tests, integration tests and regression tests.
- Baseline versus candidate: comparing behavior before and after a change.
- Scenarios and fault injection: testing the inputs that matter for the mission.
- Determinism and reproducibility: the same inputs must give the same result.
- Why "unknown" must never be counted as "pass".

Practice

- Read `tests/test_tlm.c`, then add one more check: a packet with a 1-byte payload is stored with length 1. Run `make test`.
- Read `tests/test_policy.py` and find the test proving that a missing metric gives `unknown`.
- Read `policy.toml` and say, for each rule, what evidence it needs.

CHECKPOINT

You can explain what each rule in the demo policy checks, and why a candidate that passes every rule is still only accepted for review.

Run it, read it, change it.

ASTRAPATCH

SHELL

POLICIES

1 TO 2 WEEKS

```
cd examples/demo-spacecraft
astrapatch          # opens the interactive shell
> /analyze
> /show FINDING-001
> /fix FINDING-001
> /verify
> /report
```

Exercises

EXERCISE

WHAT YOU SHOULD SEE

Full pipeline

Baseline crashes on malformed packets; the candidate survives; verdict `accepted_for_review`.

Incomplete fix

`/fix FINDING-002` then `/verify`: rejected, because fixing only the `sprintf` leaves the crash.

Tighten the policy

In `policy.toml`, set `small-change` to `max = 2`, then fix and verify again: rejected, 3 changed lines against a limit of 2.

Write your own bug

In a new folder, run `astrapatch init`, add a C file that calls `strcpy(dst, src)`, and analyze: finding `AP-C003`, CWE-120.

Read the evidence

Unpack the evidence package and match each `SHA256SUMS` line to a file.

Reset the demo at any time with `rm -rf .astrapatch build telemetry.out metrics.json`.

CHECKPOINT

You can predict the verdict of a change before running `verify`, and explain it from `verification.json` afterwards.

Join the project.

1. Request contributor access

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

Include your GitHub username, your name and affiliation if any, the area you want to work on, relevant experience, and why you are interested. Mention which stage of this roadmap you have reached; beginners are welcome.

2. Pick a first task

GOOD FIRST TASK	SKILLS FROM STAGE
Fix or clarify documentation	01
Add a test case that proves a pattern is found, or correctly ignored	02, 03, 06
Add a scenario to the demo, such as an empty or truncated packet	04, 06
Add a SARIF fixture from another analyzer and test the import	03, 05
Improve the CWE or severity mapping in an adapter	03

3. Open a pull request

- Read `CONTRIBUTING.md` and the contribution guide (PDF) in `docs/`.
- Branch, make one focused change with a test, run `pytest` and `ruff`, and open a pull request using the template.
- Expect review comments; they are how everyone, including maintainers, learns the codebase.

Where the field is heading.

Once you are comfortable with the pipeline, these are the open questions astrapatch is built to help research.

R1

Autonomous remediation

How software agents can propose and refine candidate patches, and how to measure whether they help.

R2

Mission-aware verification

How functional and operational constraints should decide whether a patch is acceptable.

R3

Resource-constrained security

Security improvements measured against memory, computation, timing and link budgets.

R4

Evidence-driven autonomy

Producing auditable, reproducible evidence for automated engineering decisions.

Next skills to build

- Fuzzing with libFuzzer or AFL++ against packet handlers.
- Building and running NASA cFS or F' locally, and writing a simulator adapter.
- Real-time systems: scheduling, deadlines and worst-case execution time.
- Writing a deterministic policy rule kind, with tests for pass, fail and unknown.

astrapatch.

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