claude-cluster: security audit

Date: October 21, 2026 · **Version**: 1.4.62 · **Commit read**: `2139c37` (integration tree, branch `integ`) · **Sources**: `ai/claude-cluster/src/cluster/` (`Web.cpp`, `Control.cpp`, `Auth.cpp`, `Attachments.cpp`, `Settings.cpp`, `Session.cpp`, `Manager*.cpp`, `Orchestrator*.cpp`, `Display.cpp`, `Tools.cpp`, `Mcp.cpp`), `templates/config.toml`, `README.md`

> **How to read this document** This is a **code-reading audit**: the findings come from reading the sources listed above and following each guard from its entry point to the action it protects. Nothing was exploited on a running instance and no dynamic tool was run.
>
> 1. Summary and severity scale.
> 2. Scope, method and what is not covered.
> 3. Threat model: who can reach what.
> 4. Findings, from the most to the least severe, each with its evidence and a fix.
> 5. What was checked and holds (the controls that work as documented).
> 6. Recommendations in order of priority.
> 7. Privacy and the GDPR.
>
> **Not included**: a dynamic test (fuzzing, a penetration test of the web access), a review of the agents themselves (Claude Code, Codex, opencode, Qwen Code), of the Android code beyond its trust model, and of libutils. The wider security audit of the whole repository by the Claude Code *mythos* audit is **coming soon** and is not redone here.

Contents

1. Summary
2. Scope and method
3. Threat model
4. Findings
5. Controls that hold
6. Recommendations
7. Privacy and the GDPR
8. Sources

1. Summary

Point	Conclusion
Overall	the design is careful: every local and network entry point has a guard, most of them correct by reading
Findings	**1** high, **5** medium, **4** low, **3** informational
Most important	a sub-session can run **any shell command through `heavy_run` without a permission prompt**, whatever its permission mode (finding F1)
Documented promise broken	three settings listed as *protected from the global session* are not (F2), and a guard on file downloads checks the wrong field (F3)
Web access	authentication, binding of a device to its address, Host / Origin checks, cookies, CSP and rate limits are sound; hardening points on the certificate authority and on `[web] settings` (F5, F7)
Control socket	protected by a `0700` folder and a `0600` socket; one weakness when `XDG_RUNTIME_DIR` is unset (F4)
Secrets	kept out of `config.toml`, logs and transcripts; one path puts an API key on a command line (F6)
Not done	no dynamic testing; the *mythos* audit is coming soon

Severity scale used below:

Level	Meaning
High	a boundary the user relies on can be crossed with no extra privilege and by default
Medium	a documented guard fails, or a boundary can be crossed under a realistic precondition
Low	limited impact, needs an authenticated device or an unusual setup
Info	hardening, or a trust-model remark with no direct exploit

2. Scope and method

Scope: the core (`claude-cluster serve`), its control socket, the web access (https page, terminal page, API), the MCP server and its tools, the settings and their protection, the attachments, the displays, the resource orchestrator and `cluster-run` , and the storage of secrets. The terminal, window and phone interfaces are covered as clients of these.

Method: for each entry point the code was read from the place a request enters (the accept loop, the control handler, the MCP tool call) down to the guard and the action. Guards were then compared with what `README.md` and the settings help promise. Where a promise and a guard differ the finding says which one is wrong. Function names are given because line numbers move.

Assumptions: a single-user desktop on a trusted LAN is the main case; the agents run with the rights of the user; the user can read all of their own files (so an attacker who already runs code as the user is out of scope, except where a guard claims to stop them).

3. Threat model

Actor	Can reach	Trust placed in them
The user at the keyboard	everything	full
Any program running as the same user	the control socket, the config and data folders	the same trust as the account: it can drive claude-cluster
Another local user	nothing, **if** the runtime folder is private (see F4)	none
A device on the LAN	the https page, public paths (`/` , `/ca.crt` , `/app.apk` , `/api/version`)	none until: the 6-digit code **and** the user's approval
A device from outside (Tailscale / router)	the same, https only	same, approval always asked
An approved web device	what the keyboard can do, minus a shell and minus the settings not opened to it	high: a lost phone is a lost claude-cluster
The Android app	the same page; pins the authority of the computer	as an approved device
The global session (a model)	the MCP tools, bounded by protected settings	**low**: it reads outside content; a prompt injected in a sub-session returns to it
A sub-session	its own small MCP (tasks, schedules, jobs of its own)	low to medium: depends on its permission mode
An agent provider (Anthropic, OpenAI...)	whatever the sessions send (prompts, files, the map of the computer)	the user's choice

The core assumption of the design is that **the global session and the sub-sessions can be manipulated by text they read** (a web page, a file, a reply). The guards below are judged against that.

4. Findings

Id	Level	Area	Finding
F1	**High**	permission gates	a sub-session runs arbitrary shell commands through `heavy_run` / `jobs_add` with no permission prompt
F2	Medium	protected settings	`sessions.global_fork` , `global_edit` , `global_answers` are not protected: the protection list names the wrong section
F3	Medium	file access	the hidden-file guard of displays reads the wrong field, so `path` / `paths` are never checked

Id	Level	Area	Finding
F4	Medium	control socket	the `/tmp/claude-cluster-<uid>` fallback is predictable and its owner is not verified
F5	Medium	web access	the local certificate authority is valid 20 years, has no name constraint and its key is unencrypted
F6	Medium	secrets	the describe / chat one-shot run puts an API key on a command line
F7	Low	web access	`[web] settings = all` (or `config`, `commands`, `web`) is equivalent to code execution on the computer
F8	Low	web access	the `web.shell` gate stops `!command` but not an agent told to run it
F9	Low	web access	the session token is compared with `==`, and any device can rotate the code
F10	Low	web access	files up to 512 MB are read whole in memory per request
F11	Info	files	credentials, keys and the config are created then `chmod`ed; `config_write` is not atomic
F12	Info	trust model	protections bound the MCP tools, not the global session process itself
F13	Info	attachments	attachment ids come from a non-cryptographic generator

F1. A sub-session can run any shell command through `heavy_run`

Where: `Manager-Tasks.cpp` (`subArgs_`), `Manager.cpp` (global arguments), `Orchestrator-Control.cpp`, `Orchestrator.cpp` (job start), `Control.cpp` (`subGate_`), `templates/config.toml` (`[orchestrator] sub_add = true`).

Evidence: the arguments of a sub-session are `--mcp-config mcp-<id>.json --allowedTools mcp__claude-cluster`: every tool of the claude-cluster MCP is pre-approved in Claude Code's own permission system. That MCP exposes `heavy_run` and `jobs_add` to sub-sessions when `[orchestrator] sub_add` is on, which is the default. `subGate_` only checks that the tool is in the list opened by the settings. The orchestrator then starts the job with `argv = {"sh", "-c", job.command}` (the job start in `Orchestrator.cpp`), as the user, with no look at the permission mode of the owner session.

Impact: the permission mode of a sub-session (`manual`, `acceptEdits`, `auto`, and, depending on the Claude Code version, `plan`) is the main safety boundary between a prompt-injected agent and your account. Through `heavy_run` the agent runs a command with no prompt and no classifier, so that boundary does not hold. It is reachable by default and needs no privilege: only text the agent reads.

Fix: do not pre-allow the job tools for sub-sessions (leave them out of `--allowedTools` so Claude Code asks), or make the core refuse `heavy_run` / `jobs_add` from a session whose mode is not `bypassPermissions`; and default `[orchestrator] sub_add` to off. The `cluster-run` wrapper is not affected: it runs inside the agent's own shell, which is already gated.

F2. Three "protected" settings are not protected

Where: `Settings.cpp` (`cluster::setting_protected`), `Config.cpp` (keys read from `[sessions]`), `Control.cpp` (`settings_set`, `config_write`).

Evidence: the protected list contains `global.answers`, `global.fork` and `global.edit`. The settings are stored and read as `[sessions] global_answers`, `global_fork` and `global_edit`. The section and the key names do not match, so `setting_protected("sessions", "global_fork")` is false. `settings_set` from the global session therefore only asks `setting_danger`, which has no entry for them, and `config_write` calls `protected_changes`, which uses the same list. The README and the settings help state these are changed by the user only.

Impact: the global session (or a prompt injected into it) can switch on `global_edit` (rewrite what a sub-session's model sees, a persistent prompt injection into it), `global_fork` and `global_answers` (answer the agents' questions in the user's place), with no confirmation.

Fix: use the real names (`sessions.global_answers`, `sessions.global_fork`, `sessions.global_edit`) in the list and add a test that every key marked as protected in the README exists in `config.toml`.

F3. The hidden-file guard of displays checks the wrong field

Where: `Control.cpp` (the `display` command), `Display.cpp` (the `file`, `image`, `audio` types).

Evidence: `display` refuses a hidden path only for `display["files"]`. A `file` display is *built* by `Display.cpp` from `display["path"]` / `display["paths"]` and the result is stored in `files`, overwriting whatever the caller sent. The caller's `files` is checked, the caller's `paths` is not. The `image` and `audio` types read `path` directly (4 MB and 10 MB) and embed the bytes in the display. `/api/file` then serves any regular file recorded in a `file` display, and `/api/display` returns the embedded bytes, to any connected web device.

Impact: `session_display {type: "file", path: "~/.ssh/id_ed25519"}` from the global session is accepted and becomes a downloadable item on the web page, contradicting "never gives a hidden file to the page as a download". The global session in `bypassPermissions` can already read that file itself, so the extra exposure is a new exfiltration channel to the page and to the transcript on disk, not new read access.

Fix: apply `hidden_path` to `path` / `paths` before the display is built, and again in `/api/file` against the recorded path.

F4. The `/tmp` fallback for the runtime folder

Where: `Tools.cpp` (`runtime_dir`), `Control.cpp` (the constructor), `Manager-Tasks.cpp` (`mcp-<id>.json`).

Evidence: without `XDG_RUNTIME_DIR` the folder is `/tmp/claude-cluster-<uid>`: a predictable name in a world-writable folder. The code calls `create_directories` and `chmod(0700)` and does not check afterwards that the folder is a real directory, owned by the user. On a shared machine another user can create it first (as a directory with mode `0777`, or a symlink). The core then binds `control.sock` there, and writes `mcp-<id>.json`, the file a session reads to launch its MCP server (`"command"`), in a folder the attacker controls. The client side (`connect_`) does not verify the owner of the socket either.

Impact: reading and replying to the control requests (prompts, permission answers), and replacing the MCP configuration of a session, which is code execution as the victim. The precondition (no `XDG_RUNTIME_DIR`, a multi-user machine) is uncommon on a systemd desktop and realistic on a shared server or a container.

Fix: after creating the folder, `lstat` it and refuse unless it is a directory owned by `getuid()` with mode `0700`; prefer a path under the data folder to `/tmp`; check the owner of the socket with `SO_PEERCRED` on the client.

F5. The local certificate authority is broad and long lived

Where: `Web.cpp` (`tls_`).

Evidence: the authority is made once with `openssl req -x509 ... -days 7300 -nodes` (a P-256 key, 20 years, unencrypted `ca-key.pem` in `~/.config/claude-cluster/web/`), `basicConstraints=CA:TRUE,pathlen:0` and no `nameConstraints`. Every device is invited to trust it (the login page, `web trust-ca` in the system store).

Impact: any process of the user, or a backup, that reads `ca-key.pem` can issue a certificate for any site that the trusting devices will accept, including this computer's system store. The key file is `0600` in a `0700` folder, which is the right baseline; the blast radius is what is large.

Fix: add `nameConstraints` (permit only the IP ranges and names the server certificate uses, including the tailnet name), shorten the validity and make the CA renewable, and document that `web trust-ca` widens trust system-wide.

F6. An API key on a command line

Where: `Manager-Describe.cpp`, `one_shot_` (the run of one question to a model, used for the descriptions and the *chat about this* of a question).

Evidence: the run is built as `argv = {"env", "KEY=value", ..., "<agent>", ...}` from `auth.env(backend)`, so for a provider with a key (`anthropic-api`, `openrouter`, `deepseek`, `openai` ...) the key is in the arguments of the `env` process (and of `setsid` when present) for the duration of the call. `Session.cpp` does the opposite for the sessions: the environment travels on an inherited pipe (fd 3) "so nothing secret is in `/proc/<pid>/cmdline`". The README states the pipe rule without exception.

Impact: `/proc/<pid>/cmdline` is readable by every local user by default; a short window on every description and every *chat about this* message. Login providers (no key) are unaffected.

Fix: reuse the pipe mechanism of `Session.cpp` for this runner.

F7. `[web] settings` can mean code execution

Evidence: `settings_set` and `config_write` from a web device are allowed for the sections named in `[web] settings`; only `web.settings` itself is excluded, and the protected list is applied to the global session only. With `all`, `config`, `commands` or `web` a device can set `commands.claude`, `web.shell` or `agent.permission_mode`.

Impact: a feature that is off by default, documented as dangerous in the settings. Still, a trusted phone with `all` is a shell by another name.

Fix: refuse the protected keys from the web as well, whatever `[web] settings` says.

F8. `web.shell = false` is a prompt gate

Evidence: `Control.cpp` blocks a `send` whose text starts with `!` (or `:` with `!`). `spawn` (with a mode and a prompt), `mode` and `send` of plain text are allowed from the web. A device can open a session in `bypassPermissions` or switch one to it, and ask the agent to run a command.

Impact: the shell switch does not stop an approved device from executing commands; it only removes the direct path. The README says a device can do everything except a shell, which is true literally and misleading practically.

Fix: document it, and gate `mode: bypassPermissions` and `spawn` modes from the web behind their own setting.

F9. Smaller points on the login and the connections

- The code and the trust secret are compared in constant time (`same_secret_`); the session token (`cc_web`) is compared with `==`. The token is 24 random bytes, the exposure is very small, but the same helper should be used.
- Five wrong codes from anyone make the core rotate the code (`_failures`); a device on the LAN can keep rotating the code and lock a legitimate login out. Per-address blocking (15 minutes) limits it to one address at a time.

F10. Whole files in memory

`/api/file` and `/api/attach/file` read a file of up to 512 MB into a `std::string` before answering. Eight connections per address and 64 in all are allowed, so an authenticated device can cost several gigabytes of memory. Stream the file instead.

F11 to F13. Informational

- `credentials.json`, the CA key and the server key are created with the default umask and `chmod`ed right after; the folders are `0700`, so the window has no practical exposure. `config_write` writes with `std::ofstream` directly (not the atomic temporary file used by `write_json`).
- The protections (protected settings, the `.md` / `.txt` rule of `export`, the hidden-path rules, the `rewrite_*` switches) bound the **MCP tools** of the global session. The session itself runs in `bypassPermissions` by default with its own shell and file tools, so it can edit `config.toml` and the prompt files directly. The README says so for the permission mode; the protections should be described as a limit on the tool surface, not on the process.
- Attachment identifiers use `std::mt19937_64` seeded by `std::random_device`: not a secret, but a cryptographic source is cheap.

5. Controls that hold

Area	Verified in the code
Web login	a code of 6 random digits (`std::random_device`), compared in constant time; a wrong code answered 1 s later; 5 wrong codes from an address block it 15 minutes and 5 from anyone rotate the code; the code is new at each opening
Device identity	cookies `HttpOnly`, `SameSite=Strict`, `Secure` over https; the session is bound to the address of the device and checked on each request; a trusted device uses a 32-byte secret in a separate cookie; banned devices refused by id
Approval	the user approves each new device (always from outside the LAN, even with `approve = false`)
Network scope	only private addresses of a subnet of an interface, the tailnet range, or any address with `remote = router` (https only); the `Host` must be an address, `localhost` or the name of the computer (DNS rebinding); a `POST` with an `Origin` must match
Headers	`X-Content-Type-Options`, `X-Frame-Options: DENY`, `Referrer-Policy: no-referrer`, HSTS over https, `Cache-Control: no-store`; the page script runs only with a per-response CSP nonce; attachments are served with `Content-Security-Policy: sandbox` and non-images as downloads
Limits	headers 16 KB / 10 s, body 1 MB (a recording 16 MB), 8 connections per address, 64 at once; the long polls are bounded
API surface	`/api/cmd` accepts a fixed list of commands and forces `origin: "web"`; the shell, the settings and the services have their own switches
TLS	TLS 1.2 minimum, no renegotiation, server cipher preference; a P-256 key per certificate, 397 days, made again 30 days before the end and when the addresses change
Control socket	the folder is `chmod 0700`, the socket `0600`; any exception of a request is answered, never fatal; a request from the MCP carries `origin` and the settings can turn each tool off
Attachments	uploads stored `0600`, never executable; names reduced to letters, digits, `.`, `_`, `-`; the type is read from the content and a program (ELF, PE) is refused; images re-encoded without EXIF; links resolved with `canonical`, compared component by component, never into the claude-cluster folders, and for the global session never into a hidden folder of the home; a file is linked only from a loopback peer

Area	Verified in the code
Prompts rewriting	`config_write` refuses the global session unless the matching `rewrite_*` switch is on; the protected keys of `config.toml` are compared before the write; `export` writes `.md` / `.txt` outside hidden folders only; `context` refuses hidden files
Secrets	API keys in the keyring or in a `0600` file, never in `config.toml`, logs or transcripts; the agents of the sessions receive them on a pipe; the token of Claude Code is read for the usage endpoint only
Orchestrator	the wrapper protocol runs on the same socket; the slot is released when the connection drops; limits, pressure rules and the per-session scope (`sub_kill_own`, `sub_see_others`) are enforced by the core

6. Recommendations

Priority	Action	Finding
1	stop pre-approving the job tools for sub-sessions, or require `bypassPermissions` for them; default `sub_add` off	F1
2	fix the protected keys (`sessions.global_*`) and add a test that compares the README list with `config.toml`	F2
3	check hidden paths on `path` / `paths` / `image` / `audio` and in `/api/file`	F3
4	verify the owner and mode of the runtime folder; avoid `/tmp`	F4
5	give the API key to the describe run on a pipe	F6
6	name constraints and a shorter validity for the authority	F5
7	apply the protected list to the web as well	F7, F8
8	constant-time token compare, stream large files, atomic `config_write`	F9, F10, F11
9	state in the docs that protections bound the MCP surface, not the global process	F12

7. Privacy and the GDPR

claude-cluster is software the user runs on their own computer. Its conversations, history, attachments, keys and the map of the computer stay in the user's folders; there is no server operated by the project, no telemetry and no account. The **GDPR does not apply to claude-cluster itself**, because no personal data is collected, transmitted to or processed by the project. It does still apply to what the user chooses to send to the providers they connect (an API provider reads the prompts, the files and the map of the computer that a session sends it), and the user remains responsible for that processing and for the choice of provider. The web access serves only the user's own devices.

8. Sources

Code read (commit `2139c37`): `src/cluster/Web.cpp`, `Control.cpp`, `Mcp.cpp`, `Settings.cpp`, `Config.cpp`, `Auth.cpp`, `Session.cpp`, `Manager.cpp`, `Manager-Tasks.cpp`, `Manager-Describe.cpp`, `Orchestrator.cpp`, `Orchestrator-Control.cpp`, `Attachments.cpp`, `Display.cpp`, `Tools.cpp`; `templates/config.toml`; `README.md`; `docs/orchestrator.md`.

Assumptions: the behaviour of `plan` mode with pre-allowed MCP tools depends on the Claude Code version and was not tested; the exposure in F4 depends on the environment of the user.