Security Architecture — claude-config-template

Honest Risk Assessment

This system gives an AI agent significant access to your machine. That is inherently dangerous.

However, the risk profile varies dramatically by deployment model:

	Claude Code + this config	OpenClaw (self-hosted)
Runs as	Interactive CLI, user present	24/7 daemon, unattended
Human in the loop	Yes — user approves destructive ops	No — fully autonomous
Attack surface	Local machine, user's permissions	Server + all connected channels (Slack, Teams, WhatsApp)
Credential exposure	`.mcp.json` on local disk	API keys in running service, accessible from network
Blast radius of compromise	One user's files	All connected channels + server + all users who interact
Sandbox	bubblewrap (Linux namespaces)	None (runs as service user)
Permission gating	Explicit allow-list per tool	No tool-level permission model
Prompt injection risk	Low — input is the user typing	High — any message in Slack/Teams can be crafted input
Audit trail	Git history + session context	Application logs only
Recovery from bad action	`git revert`, files are local	Messages sent, API calls made — not reversible

Bottom line: Claude Code with this config is a powerful tool with real risks, but the human-in-the-loop model + sandbox + permission gating + git audit trail make it **categorically safer** than any unattended AI agent. OpenClaw's always-on, multi-channel exposure is a fundamentally different (and higher) risk class.

Residual risks even with this setup

- Claude runs with **your user permissions** — it can read/write anything you can
- `Write(*)` is pre-approved — a hallucinating agent could overwrite important files (git history is your safety net)
- MCP servers extend reach to external services (GitHub, Twitter) — a confused agent could post or commit unwanted content
- `docker` is in the allow list — container operations have their own blast radius
- The config repo itself is a high-value target — anyone with push access controls Claude's behavior

This is not "safe." It is "managed risk with recovery options."

Sandboxing (Claude Code Built-in)

Layer	Mechanism	What it does
Process sandbox	bubblewrap (`bwrap`)	Linux namespace isolation — Bash commands run in a restricted container with limited filesystem/network access
IPC proxy	socat	Proxies communication between Claude Code and sandboxed processes — prevents direct system access
Platform	Linux namespaces + seccomp	Kernel-level syscall filtering. No privilege escalation from within the sandbox

Required packages: `socat` + `bubblewrap` (installed by `setup/install-base.sh` and `setup/bootstrap-fedora.sh`)

Without these, **sandboxed tool calls fail silently** — Claude Code falls back to unsandboxed execution or auto-denies.

Permission Model

How it works

Claude Code uses a grant-based permission system in `settings.json`:

```
permissions.allow = [ "ToolName(glob_pattern)", ... ]
```

What's pre-approved (shipped in template)

Category	Allowed	Examples
Read-only tools	All	`Read(*)`, `Glob(*)`, `Grep(*)`, `WebSearch`, `WebFetch(*)`
File modification	All	`Write(*)`, `Edit(*)`
Safe Bash commands	40+ patterns	`git`, `npm`, `node`, `python3`, `docker`, `gh`, `curl`, `mkdir`, `cp`, `mv`
Orchestration	Skill only	`Skill(orchestration)`

What's NOT pre-approved (prompts user)

Command	Why blocked
`rm` / `rm -rf`	File deletion — must confirm
`sudo`	Privilege escalation — must confirm
`chmod` (partially)	`chmod` is allowed but `chown` is not
Arbitrary Bash	Commands not matching any allow pattern trigger a prompt

Subagent permissions

Background subagents **cannot prompt the user**. If a tool isn't in the allow list, the call is **auto-denied silently**. This is the #1 cause of mysterious subagent failures. Fix: add the minimal matching pattern to `permissions.allow`.

Credential & Secret Management

Layer	Mechanism
API keys	Environment variables loaded from `secrets.env` (never committed)
MCP credentials	Stored in `.mcp.json` (per-user, not in repo)
Git authentication	`git-credential-mcp` helper reads PAT from `.mcp.json` — single source of truth, no embedded tokens
VPS secrets	`secrets.env.template` tracked, `secrets.env` gitignored

.gitignore protection

```
.env, .env.*, *.key, *.pem, secrets/, credentials/
```

Both repos enforce this. The template ships with a conservative gitignore.

Trust Boundaries

Layer	Role	Key controls
USER (interactive)	Approver	Can approve/deny any tool call in real-time
CLAUDE CODE (main process)	Executor	Runs with user's filesystem permissions. Bash sandboxed via bubblewrap. Permission model gates destructive ops.
SUBAGENTS (background)	Workers	Same sandbox. CANNOT prompt user – auto-denied if not in allow list. Isolated context windows.
MCP SERVERS (external tools)	Integrations	Each server has its own credentials in .mcp.json. Scoped access. No cross-server credential sharing.
VPS (remote, headless)	Infrastructure	secrets.env loaded at bootstrap only. SSH tunnel for services. Let's Encrypt TLS.

What's NOT Covered (Gaps)

Gap	Risk	Mitigation
No dedicated security doc	Security rules scattered across 10+ files	**This document** consolidates them
Write(*) is fully open	Claude can overwrite any file the user owns	Sandbox + git history provide recovery
No network egress filtering	Sandboxed processes may still make outbound connections	MCP servers are the primary external access; Bash `curl` is allowed
No file integrity monitoring	Modified files not detected between sessions	Git diff at session start (hook) shows changes
Subagent blast radius	A subagent with `Write(*)` can modify any file	Git-based audit trail; session hooks commit state
MCP token scope	GitHub PAT with `repo` scope = full repo access	Use fine-grained PATs where possible

---

Platform-Specific Security

WSL (Windows Subsystem for Linux)

Feature	Detail
Sandbox deps	`sudo apt install socat bubblewrap`
Windows Defender	Exclude WSL paths to avoid scanning overhead (see wsl-environment.md)
Line endings	`core.autocrlf input` prevents CRLF injection
Performance boundary	`/mnt/c/` is slow — keeping files in `/home/` also reduces Windows-side exposure

Native Linux (Fedora, etc.)

Feature	Detail
Sandbox deps	`dnf install socat bubblewrap` (via bootstrap-fedora.sh)
No additional hardening needed	bubblewrap uses kernel namespaces natively

---

Audit Trail

Mechanism	What it captures
Git history	Every file change, every session, every machine
session-context.md	What was done, when, by which session
Session hooks	Auto-commit + push at session end; auto-pull at session start
Registry	Which projects exist, where, what status

The combination of git + session hooks means **every action is version-controlled and recoverable**.

Zone Identifiers / Windows Security Tools

Not currently documented in the config template.

On Windows/WSL, two security mechanisms may apply to downloaded files:

Tool	What it does
NTFS Alternate Data Streams (ADS)	Windows attaches a `:Zone.Identifier` stream to downloaded files marking their origin (Internet, Intranet, etc.) — the "Mark of the Web" (MOTW)
Unblock-File (PowerShell)	Removes the Zone.Identifier ADS, telling Windows the file is trusted

These are **Windows-side** mechanisms. Within WSL's Linux filesystem (`/home/`), NTFS ADS don't exist. They only matter for files accessed via `/mnt/c/` or when executing scripts from the Windows filesystem.

If you were using tools that left Zone Identifiers during WSL setup, those were likely Windows-side script downloads. They don't affect the Claude Code sandbox (which is pure Linux bubblewrap).

Generated 2026-02-25. Source: claude-config-template + claude-config analysis.