vge-cc-guard User Guide

Local security sidecar for Claude Code: runtime guardrails for prompts, files, tool calls, URLs, and tool outputs.

AT A GLANCE

» Requires Node.js >= 20.10.0

» Interactive TUI configurator

» 7 hook event types supported

» Per-tool gate: allow / ask / block

» Credential path protection on by default

» URL Access Baseline with 5 presets

» Session-scoped allow / block decisions

» JSONL audit log with built-in filters

» Config reload without Claude Code restart

Node.js >=20.10	7	10	5
prerequisite runtime	hook event types	built-in tool policies	URL baseline presets

User-scope | Project-scope | Interactive TUI | Per-tool gate policies | JSONL audit log

WHO THIS GUIDE IS FOR

Use this guide if you are:

» A developer using Claude Code with `vge-cc-guard` installed.
» A team lead setting the default policy for a project.
» A security or platform engineer helping users understand blocked actions.
» A support operator reviewing a user's configuration or audit events.

This guide focuses on observable behavior, configuration choices, and safe operating practices. It does not describe internal implementation details beyond what users need to operate the tool.

THE MENTAL MODEL

Claude Code can ask to run tools such as `Read`, `Bash`, `WebFetch`, `Edit`, or `Write`. Some of those tools can read sensitive files, fetch untrusted web content, run shell commands, or modify a repository. **vge-cc-guard** sits beside Claude Code and answers three practical questions:

#	QUESTION	HOW IT IS ANSWERED
1	Should this tool be allowed to run at all?	Gate policy: allow, ask, or block before execution.
2	Is the tool output safe enough to give to Claude?	VGE output analysis after execution: allow, frame, quarantine, or block.
3	If blocked, what should happen?	User chooses: block, allow once, or allow for session.

The guard is intentionally conservative. The default posture is to preserve security and auditability first. When a choice is required, the user receives a clear prompt with numbered options.

WHAT THE GUARD PROTECTS

SURFACE	WHAT IT MEANS	WHY IT MATTERS
Prompt text	Text you submit to Claude Code.	Prompt text can contain malicious instructions, copied web content, or data that policy does not allow into model context.
Prompt attachments	Files attached to a prompt.	Attachments can contain hidden instructions or sensitive data.
Tool execution	Whether a tool is allowed before it runs.	Some tools mutate files, fetch URLs, or run commands.
Tool output	Content produced by a tool after it runs.	Tool output can contain prompt injection, secrets, or untrusted instructions.
URLs	URLs found in tool inputs.	Some URL targets should never be fetched, such as cloud metadata endpoints.
Credential paths	Known local credential files and patterns.	These should not be read, edited, or written by Claude Code by accident.
Subagent output	Tool output produced on behalf of a subagent.	Subagents can collect risky output just like the parent session.

WHAT THE GUARD DOES NOT DO

The guard is not a replacement for normal source control, code review, or secrets hygiene. It does not guarantee that every bad command is detected before execution. Local tool gates and URL/credential protections run before execution, but content classification often happens after a tool has produced output. For example, a web fetch may run, then the fetched output may be quarantined before Claude can use it. It also does not remove the need for human judgment. When the guard asks you to decide, you should treat the prompt as a real security decision.

DAILY WORKFLOW

A normal protected session looks like this:

» You start Claude Code in a project.
» Claude Code invokes hooks registered by `vge-cc-guard`.
» The guard starts or contacts its local daemon.
» Tool calls are allowed, denied, routed to Claude Code's native approval prompt, or checked after execution depending on policy.
» VGE decisions and local guardrail decisions are written to a private local audit log.
» If content is blocked but recoverable, you choose `block`, `allow once`, or `allow for session`.

In a clean session, you may not notice the guard except for occasional approval questions. In a risky session, you may see a block message or decision prompt.

INSTALLATION

Install the package globally (published under the `@vigil-guard` npm scope):

```
npm install -g @vigil-guard/vge-cc-guard

npm install -g @vigil-guard/vge-cc-guard@beta # beta channel
```

Install hooks for your user:

```
vge-cc-guard install --apply --scope=user
```

Install hooks for only the current project:

```
vge-cc-guard install --apply --scope=project
```

Preview changes without writing them:

```
vge-cc-guard install --dry-run --scope=user

vge-cc-guard install --dry-run --scope=project
```

Choosing User Scope Or Project Scope

SCOPE	BEST FOR	SETTINGS FILE
User	A single developer who wants the guard in all Claude Code projects.	~/.claude/settings.json
Project	A repository where the team wants project-local hook settings.	/.claude/settings.json

User scope is convenient. Project scope is easier to reason about when a repository has its own Claude Code policy. After installation, restart open Claude Code sessions so they pick up the new hook settings.

What Installation Creates

The installer registers Claude Code hooks and creates local state under `~/.vge-cc-guard/`:

PATH	PURPOSE
~/.vge-cc-guard/config.json	Main guard configuration.
~/.vge-cc-guard/audit.log	Local JSONL audit log.
~/.vge-cc-guard/debug.log	Local diagnostic log when debug logging is enabled.
~/.vge-cc-guard/cc-settings-backups/	Backups of Claude Code settings edited by the configurator.

PATH	PURPOSE
~/.vge-cc-guard/install-records.json	Tracks installed scopes for safe uninstall.

API keys are stored in the config file. Keep the file private.

FIRST CONFIGURATION

Open the configurator:

```
vge-cc-guard config
```

Start with **API Keys & VGE Connection**. You need:

» VGE API URL.
» Input API key.
» Optional output API key.
» Optional client ID.

> Use **Test Connection** before saving. A successful test updates `verified_at` so future support sessions can tell whether the current URL/key pair has been validated.

TUI CONFIGURATOR

The configurator is a terminal UI. Common keys:

KEY	ACTION
Up / Down	Move selection.
Enter	Open, toggle, confirm, or edit.
Tab / Shift-Tab	Move between fields where supported.
Space	Cycle a compact option where supported.
s	Save on editable policy screens.
r	Restore defaults where supported.
Esc	Go back. If there are unsaved changes, press again to discard.
Ctrl-C	Exit immediately.

Main screens:

SCREEN	PURPOSE
API Keys & VGE Connection	Configure VGE endpoint and credentials.
Tools Policy	Decide which tools can run and which outputs are analyzed.
Native CC Permissions	Edit Claude Code's own allow/ask/deny permission rules.
IDE Compatibility	Tune enforcement for terminal versus IDE native panels.
Security Baseline	Configure credential path protection and URL baseline rules.
View Current Configuration	Review and export a redacted configuration summary.
Live Events	Watch hook events as they happen.
Decision History	Review recent blocking decisions.
Audit Log	Browse local audit events.

SCREEN	PURPOSE
Stats	View decision and health counters.

API KEYS & VGE CONNECTION

This screen answers: "Where is VGE, and which key should the guard use?"

FIELD	WHAT IT DOES	WHAT TO CHOOSE
VGE API URL	Base URL for VGE requests.	Use your organization's VGE URL. HTTPS is required for normal use.
Client ID source	`Auto`, `Manual`, or `Disabled` — controls what, if anything, is sent to VGE as `metadata.clientId`.	Auto sends your OS username. Manual sends a literal label you type. Disabled omits the field entirely.
Input API Key	Key for prompt text, attachments, and input-side checks.	Required. Use a key scoped for this installation.
Output API Key	Optional key for tool-output scans.	Leave blank unless your organization separates input and output scanning keys.
Test Connection	Checks that the URL and input key work.	Run before saving, especially on first setup.

> If `api_key_output` is empty, the guard reuses `api_key_input` for output scans.

How Test Connection Works

Test Connection routes through the local daemon, not directly from the TUI process. The path is: TUI -> local daemon -> VGE. This is the same path that real hook traffic uses, so a successful Test Connection proves the daemon can reach VGE with the keys entered.

Client ID Source Modes

MODE	WHAT IS SENT TO VGE	WHEN TO USE
`Auto` (default)	OS username, detected once per daemon process from `os.userInfo()`.	You want events attributed to a developer without manual config.
`Manual`	Literal value you type in the Client ID field.	You need a stable label such as `ci-runner-7` that helps with incident triage.
`Disabled`	`metadata.clientId` is omitted entirely from VGE payloads.	You explicitly don't want to expose the OS username or any identifier.

> **Privacy note.** Auto mode sends the OS username (indirect PII) to VGE on every analyze call. If your VGE deployment crosses an organizational or jurisdictional boundary, prefer Manual with a non-personal label or Disabled.

When To Change These Values

Change VGE settings when:

» Your organization moved VGE to a new URL.
» Your key was rotated.
» A support team asked you to use a different client label.
» You want output scans to use a separate key.

TOOLS POLICY

This screen answers two different questions for every tool: (1) Can the tool run before any output exists? (2) If it runs, should the output be analyzed before Claude uses it? Each tool has two fields.

FIELD	VALUES	MEANING
`gate`	`allow`, `ask`, `block`	Controls whether the tool can run before execution.
`analyze_output`	`on` / `off`	Controls whether output from that tool is sent to VGE and can be framed, quarantined, or blocked.

Gate Values

GATE	USER EXPERIENCE	USE WHEN
`allow`	The tool may run without a pre-execution question.	The tool is common and you rely on output scanning or other guardrails.
`ask`	Claude Code shows its native approval prompt before the tool runs.	You want a human check before execution but do not want a hard deny.
`block`	The tool is denied before it runs.	The tool is too risky for this profile or should be enabled only temporarily.

Output Analysis

When `analyze_output` is on, completed tool output can be checked by VGE. The guard can then:

» allow the output normally,
» add a security frame,
» quarantine the output and ask you what to do,
» block or fail closed when policy requires it.

When `analyze_output` is off, the guard does not send that tool's output to VGE. Pre-execution policy, credential protection, URL baseline checks, and Claude Code native permissions can still apply.

Subagent Quarantine Visibility

Text returned by a subagent is not scanned by the `Task` output policy by default. Instead, the guard tracks tools that the subagent runs. When a subagent-owned tool output is quarantined, the parent Agent output receives a structured notice:

```
[VGE_SUBAGENT_QUARANTINE_NOTICE]

notice_format_version: 1

session: <session_id>

agent_id: <agent_id>

total_quarantined: <count>

counts_by_status: pending=<n> resolved_block=<n>

resolved_allow_once=<n> resolved_allow_session=<n>

entries:

- decision=dec_<id> tool=<tool> resource="..." reason=<code> status=<status>

operational_note: <guidance>

[/VGE_SUBAGENT_QUARANTINE_NOTICE]
```

The notice tells Claude that the subagent worked on safe placeholders and the final response may be incomplete. It does not release the quarantined output. Resolve any pending decisions and re-invoke the Agent call if the response needs the quarantined material.

Default Tool Policy

TOOL	GATE	ANALYZE OUTPUT	WHY
`Bash`	allow	on	Shell output is high-value and high-risk. Output is checked.
`Read`	allow	on	File reads are common and can expose sensitive or adversarial content.
`Grep`	allow	on	Search output can reveal risky text snippets.
`Glob`	allow	off	File listings are lower-content and often noisy.
`WebSearch`	allow	on	Web results are untrusted and should be framed or blocked when risky.
`WebFetch`	allow	on	Fetched pages can contain prompt injection or unsafe instructions.
`Write`	block	off	Writes mutate files, so the default is conservative.
`Edit`	block	off	Edits mutate files, so the default is conservative.
`Task`	allow	off	Subagent-owned underlying tool output is handled separately.
`*`	ask	off	Unknown tools require native approval by default.

Practical Profiles

Use the defaults for a strict security posture.

For a trusted local coding workflow, teams often change `Edit` from block to ask or allow after they are comfortable with review and source control. Keep credential protection enabled.

For research-heavy work, keep `WebSearch` and `WebFetch` output analysis on. This allows the guard to treat retrieved web content as untrusted even when the tool itself is allowed.

For IDE environments where decision prompts are not visible, consider disabling output analysis only for the affected tools, or use terminal Claude Code for full decision handling.

NATIVE CC PERMISSIONS

This screen edits Claude Code's own permission arrays:

ARRAY	MEANING
`permissions.allow`	Claude Code can run matching commands without prompting.
`permissions.ask`	Claude Code prompts before running matching commands.
`permissions.deny`	Claude Code blocks matching commands before hooks run.

These are Claude Code permissions, not VGE decisions. They are useful for simple command patterns such as shell deny rules.

How Native Permissions Relate To Tools Policy

Native Claude Code permissions can block or ask before `vge-cc-guard` sees a tool call. The guard's Tools Policy still exists separately and still applies to hook events that reach it. Use native permissions for simple execution controls. Use Tools Policy and output analysis for guard-level policy and VGE-backed decisions.

Managed Default Baseline

The optional default baseline adds generic deny rules managed by `vge-cc-guard`. Managed rules are tracked separately so they can be removed without deleting user-owned rules. Use this when you want a known starting point for Claude Code's own permission system. Leave it off if your organization already manages Claude Code permissions elsewhere.

IDE COMPATIBILITY

Terminal Claude Code gives the clearest experience for blocking decisions. IDE native panels may not always show conversation-native decision prompts in the same way. The compatibility settings let you reduce hidden friction without turning off core local safety protections.

SETTING	DEFAULT	WHAT IT CONTROLS
Prompt text	`enforce`	Whether unsafe prompt text can block or ask before Claude uses it.
Prompt file attachments	`enforce`	Whether unsafe prompt attachments can block or ask before Claude uses them.
Subagent tool outputs	`enforce`	Whether subagent-owned tool output participates in enforcement.
Prompt text VGE failures	`fail_closed`	What happens if prompt text scanning cannot complete.
Prompt attachment VGE failures	`fail_closed`	What happens if attachment scanning cannot complete.
PostTool output VGE failures	`fail_closed`	What happens if tool-output scanning cannot complete.
PostTool overload backpressure	enabled	Short cooldown behavior for eligible fail-open web research scan failures.

`enforce` Versus `off`

`enforce` means the guard can stop the session or ask for a user decision. `off` means that stage does not block or ask. Use it only when the user experience would otherwise be broken, for example when an IDE hides a prompt. Other protections still apply where relevant.

`fail_closed` Versus `fail_open`

`fail_closed` blocks when a scan cannot complete. This is safer. `fail_open` allows work to continue when VGE is unavailable or overloaded. This is more convenient but weaker. It does not override real VGE `BLOCKED` decisions, credential path protection, tool policy blocks, or URL baseline blocks. Changing a failure mode to `fail_open` requires a second confirmation in the configurator.

SECURITY BASELINE

This screen handles local protections that do not depend on VGE.

Credential Path Protection

Credential path protection is enabled by default. It blocks `Read`, `Edit`, and `Write` for known credential paths and sensitive filename patterns.

Examples include:

- `.env`, `*.env`, and `*/.env`
- `~/.ssh/*`
- `~/.aws/credentials` and `~/.aws/config`
- `~/.kube/config`
- `~/.config/gcloud/*` and `~/.gcp/*`
- Private key filenames: `id_rsa*`, `id_ed25519*`, `id_ecdsa*`
- Filenames containing `credentials` or `secrets`.

> A model does not need raw credentials to help with most tasks. Only disable credential protection when you have a specific operational reason and understand that Claude Code may be able to read credential material into context.

URL Access Baseline

The URL Access Baseline blocks risky URL targets before fetch-like tool calls run. It is local and deterministic.

Default behavior:

» Normal public `http` and `https` URLs are allowed unless another rule matches.
» Cloud metadata targets are blocked.
» Unsafe URL shapes are blocked.
» More aggressive categories are available but off by default.

Built-in presets:

PRESET	DEFAULT	WHAT IT BLOCKS
`cloud_metadata`	on	Cloud metadata endpoints and related addresses.
`unsafe_url_shapes`	on	URL forms that are unsafe or ambiguous, such as embedded credentials.
`oob_callback_collectors`	off	Known callback or collection services used in out-of-band exfiltration patterns.
`strict_internal_network`	off	Internal/private network targets when your policy requires strict isolation.
`public_paste_and_file_drops`	off	Public paste and file-drop services.

Custom deny rules:

RULE TYPE	EXAMPLE	USE FOR
Host	`example.internal` or `*.corp.example`	Blocking exact hosts or host families.
CIDR	`10.0.0.0/8`	Blocking network ranges.
Scheme	`ftp`	Blocking URL schemes.
URL pattern	`https://example.com/private/*`	Blocking simple URL glob patterns.

CLAUDE CODE CONTRACT HEALTH

The guard's PostTool output replacement depends on whether the running Claude Code binary accepts the replacement shape. This is tracked as a live contract between the guard and the installed Claude Code binary. Normally you do not need to think about this. The configurator and `doctor` surface it when something changes.

WHERE	WHAT IT SHOWS
MainMenu header	Short status indicator when contract is degraded.
View Current Configuration	Contract state, reason, CC version, binary path, SHA prefix, last live probe, next action.
`vge-cc-guard daemon status`	Same fields in machine-friendly form.
`vge-cc-guard doctor --cc-contract`	Detailed report and recommended next action.

Typical causes of contract degradation:

» Claude Code was updated.
» The `claude` binary was moved to a new path.
» The SHA-256 of the binary changed.
» A previous live probe result was lost or never ran.

When the contract is degraded, L0 output replacement is disabled but PostTool blocking still works via a safer degraded path. You still receive decision prompts and the original blocked output is not released. The daemon schedules one auto-probe in the background when the contract is degraded. Run `vge-cc-guard doctor --cc-contract` to check status and next action.

VIEW CURRENT CONFIGURATION

Use this screen when you want to understand the effective setup without editing anything. It shows:

» config path,
» VGE URL and masked keys,
» client ID,
» tool gates,
» output-analysis settings,
» prompt and attachment enforcement,
» failure modes,
» URL baseline status,
» session and decision timing settings.

The export action writes a redacted summary suitable for support requests. API keys are masked.

LIVE EVENTS, DECISION HISTORY, AUDIT, AND STATS

Live Events

Live Events shows hook activity as it happens. Use it when Claude Code appears paused or when you want to see whether a tool is being allowed, denied, or checked.

FIELD	MEANING
Time	When the event occurred.
Hook	Which Claude Code hook fired.
Tool	Tool name when applicable.
Decision	The guard or VGE outcome.
Source	Whether the event came from prompt, attachment, tool input, or tool output.
Latency	How long the step took.

Decision History

Decision History shows recent blocking decisions. It is useful when a user says "Claude is waiting" or "I typed a reply and nothing happened." Look for:

» `pending` decisions that still need a choice,
» `resolved_block` decisions,
» `resolved_allow_once` decisions,
» `resolved_allow_session` decisions,
» owner information for parent sessions and subagents.

Audit Log

The audit log is local JSONL. It records security-relevant events without storing raw blocked tool output. Use it to answer:

» What was blocked?
» Which tool or stage produced the event?

» Which decision ID was involved?
» Was the user choice `block`, `allow_once`, or `allow_session`?
» Was the event produced by VGE, local policy, or local fallback behavior?

Stats

Stats gives a quick operational summary: decision counts, VGE signal counts, health indicators, and active sessions.

COMMANDS

`vge-cc-guard install`

Registers hooks in Claude Code settings. Use `--dry-run` first to review, then `--apply`:

```
vge-cc-guard install --dry-run --scope=user

vge-cc-guard install --apply --scope=user

vge-cc-guard install --dry-run --scope=project

vge-cc-guard install --apply --scope=project
```

`vge-cc-guard config`

Opens the configurator. Use it for normal configuration changes. Direct JSON edits are possible but not recommended.

`vge-cc-guard daemon`

Starts or controls the local daemon. Most users do not need to start it manually.

```
vge-cc-guard daemon

vge-cc-guard daemon status

vge-cc-guard daemon reload

vge-cc-guard daemon stop

vge-cc-guard daemon restart
```

`daemon status` includes `sidecarEnabled=`. `unknown` means the running daemon was started by an older build. Restart the daemon to clear it.

`vge-cc-guard doctor`

Runs local diagnostics. Flags:

FLAG	WHEN TO USE
(none)	Default. Reports local state and VGE connectivity check with next action when something is degraded.
`--no-vge`	Skip live VGE round-trip when you only want a quick local state check (offline or VGE known unavailable).
`--cc-contract`	Print detailed Claude Code contract status block.
`--assume-live-pass`	Manual trust override marking CC contract as healthy without running a live probe. Use only after separately verifying binary behavior.

`vge-cc-guard reset-session`

Clears session allow/block decisions. Use when a session is stuck, to forget session-scoped allows, or when support asks to clear local decision state.

`vge-cc-guard uninstall`

Removes installed hooks. Default removes only `vge-cc-guard` entries. `--restore` replaces the settings file with the install-time backup.

```
vge-cc-guard uninstall --yes --scope=user

vge-cc-guard uninstall --yes --scope=project

vge-cc-guard uninstall --yes --scope=user --restore
```

BLOCKING DECISIONS

The most important user interaction is the blocking decision prompt. A blocking decision means the guard has stopped a specific resource, prompt, attachment, URL, or tool output and needs a user choice.

Choices

CHOICE	MEANING	WHEN TO USE
`1` / `block`	Keep the item out of model context and remember this exact item as blocked.	Use when you do not trust the content or do not need it.
`2` / `allow once`	Allow this exact item once.	Use when you reviewed the situation and want to continue one time.
`3` / `allow for session`	Allow this exact item until the Claude Code session ends.	Use when the same exact item will be needed repeatedly in this session.

Accepted Replies

For a single active decision, these are accepted:

```
1

2

3

block

allow once

allow for session
```

For exact command style:

```
vge block dec_<id>

vge allow dec_<id>

vge allow-session dec_<id>
```

Some decisions accept a continuation instruction:

```
2 continue with the original request

3 continue with the original request

vge allow dec_<id> continue with the task

vge allow-session dec_<id> continue with the task
```

What Happens If You Type Something Else

If a decision is active and your reply is not a valid decision command, the guard re-shows the prompt. That reply is **not** sent to Claude as task content. This is deliberate: it prevents accidental task text from being interpreted while the session is paused for a security choice.

BLOCK MESSAGE FORMAT

Most block messages start with `VGE Agent Guard:`. Immediate policy blocks:

TYPE	MESSAGE FORMAT
Tool policy block	`VGE Agent Guard: is blocked by policy.`
Credential path block	`VGE Agent Guard: is on the credential protection deny list.`
Previously blocked resource	`VGE Agent Guard: resource was blocked by user decision.`
URL baseline block	`VGE Agent Guard: URL denied by local URL access baseline ().`

Decision Prompt Format

When VGE or local enforcement requires a user decision, the prompt shows: Stage, Decision ID, Owner, Tool, Resource (sanitized; URLs may have query strings redacted so secrets are not displayed), Reason, and VGE scores. Replies with `1`, `2`, `3` or exact `vge ... dec_` commands are accepted.

Inline Terminal Prompt Format

In terminal Claude Code, you may see a same-terminal decision screen:

```
VGE Agent Guard: decision required

Claude Code is paused. The blocked content is not released until you choose.

Kind: PostTool output Tool: Bash

Resource: Bash command: <command>

Reason: <reason>

VGE: score=<score> ruleAction=<action> decision=<decision>

Choose one option. Press 1, 2, or 3.

1) block 2) allow once 3) allow session
```

Redacted Tool Output Placeholder

For supported `Read` and `Bash` outputs, Claude may see a redacted placeholder instead of the original blocked output.

```
[VGE SECURITY REDACTION]

Tool: <tool> Resource: <resource id>

Router outcome: <outcome> VGE decision: <decision>

Score: <score>

Original tool output was removed and is not included in this message.

[/VGE SECURITY REDACTION]
```

For unsupported tool-output shapes, the guard fails closed instead of showing a malformed replacement. The original blocked output is not included.

Subagent Quarantine Notice

When a child subagent had tool outputs quarantined, the parent Agent output may contain a marker:

```
[VGE_SUBAGENT_QUARANTINE_NOTICE]

...

[/VGE_SUBAGENT_QUARANTINE_NOTICE]
```

The marker means the subagent response may be incomplete because one or more tool outputs were replaced with safe placeholders. Resolve each pending decision and re-invoke the Agent call if the response needs the quarantined material.

DECISIONS BY TYPE

Prompt Text And Attachments

Prompt text and attachments are checked before Claude uses them when their policy is `enforce`. If blocked: Claude has not used that prompt or attachment yet. You receive a decision prompt. `block` keeps it out of context. `allow once` allows the exact prompt or attachment one time. `allow for session` allows that exact item until session end. For attachments, credential path protection still applies even if attachment analysis is turned off.

Tool Output Decisions

Tool-output decisions happen after a tool has run but before Claude is allowed to rely on the blocked output. What to expect:

» Supported outputs can be replaced with a redacted placeholder.
» Unsupported outputs fail closed.
» Claude should not summarize, transform, or act on the blocked original output until you allow it.
» If you allow a blocked `Read`, Claude may need to re-read the file through the normal tool path before editing workflows continue.

URL Decisions

There are two URL-related behaviors:

» Local URL baseline denials.
» VGE-backed decision prompts for risky URL-related activity.

Local URL baseline denials are immediate. They normally do not ask a question. They show a denial reason and prevent the fetch. If a decision prompt appears, use the normal `1`, `2`, `3`, or `vge ...` decision commands.

SESSION STATE

Some decisions apply only to the current Claude Code session.

DECISION TYPE	LIFETIME
`allow once`	One exact retry or hook call.
`allow for session`	Until `SessionEnd` or session reset.
`block`	Exact-resource block for the session.

Use `vge-cc-guard reset-session` to clear session-scoped local decisions.

CONFIGURATION REFERENCE

The supported editing path is `vge-cc-guard config`. Direct file editing is for automation and support. Config file: `~/.vge-cc-guard/config.json`

FIELD	MEANING
`version`	Config schema version. Current value is `1.0.0`.
`vge`	VGE endpoint, API keys, client label, and scan timing budgets.
`tools`	Per-tool execution gates and output-analysis switches.
`policy`	Runtime safety policy and local guardrails.

`vge` Object

FIELD	TYPE	DEFAULT	WHY IT EXISTS
`api_url`	URL	`https://api.vigilguard`	Tells the guard where to send VGE checks.
`client_id`	string	empty	Adds an operator-friendly label to VGE events.
`api_key_input`	string	empty	Authenticates prompt, attachment, and input-side checks.
`api_key_output`	string | null	`null`	Optional separate key for tool-output checks.
`verified_at`	ISO datetime	`null`	Records the last successful Test Connection for support.
`posttool_budget_ms`	int 500-120000	120000	Maximum time budget for tool-output scanning.
`pretool_url_budget_ms`	int 200-120000	120000	Deprecated compatibility field. PreTool URL handling is local-only.
`pretool_url_total_deadline_ms`	int 500-120000	120000	Deprecated compatibility field. PreTool URL handling is local-only.
`userprompt_attachment_budget_ms`	int 500-120000	120000	Maximum time budget for attachment scanning.
`userprompt_text_budget_ms`	int 500-120000	120000	Maximum time budget for prompt text scanning.

Shorter budgets reduce waiting time but make scan timeouts more likely. Longer budgets reduce false operational failures but can make a blocked session feel paused for longer.

`tools` Object

Each tool entry has a `gate` (`allow`, `ask`, `block`) and an `analyze_output` flag (`true` / `false`). Known default entries: `Bash`, `Read`, `Grep`, `Glob`, `WebSearch`, `WebFetch`, `Write`, `Edit`, `Task`, and `*`. `*` is the fallback for unknown or custom tools.

`policy` Object

FIELD	DEFAULT	WHY IT EXISTS
`credential_protection`	`true`	Blocks known credential paths regardless of tool policy.
`prompt_text_analysis`	`enforce`	Controls whether prompt text can trigger blocks or decisions.
`prompt_attachment_analysis`	`enforce`	Controls whether prompt attachments can trigger blocks or decisions.
`subagent_output_analysis`	`enforce`	Controls enforcement for subagent-owned tool output.
`posttool_research_output`	quarantine_continue	Controls explicit VGE blocks for WebFetch/WebSearch outputs.
`vge_failure_mode`	fail_closed	Defines what happens when VGE scans cannot complete (per stage).
`vge_overload_backpressure`	enabled	Applies short cooldowns to eligible fail-open web research overload cases.
`session_idle_ttl_hours`	`24`	How long idle session files remain on disk (1-168 hours).
`enable_l0_output_replacement`	`true`	Legacy compatibility field retained for existing configs.
`url_access_baseline`	enabled	Local URL deny-list baseline configuration.
`decision_timeout_ms`	`120000`	Decision timing metadata and compatibility window.

`policy.posttool_research_output`

FIELD	VALUES	DEFAULT
`block_mode`	`quarantine_continue` or `stop_on_block`	`quarantine_continue`

`quarantine_continue` replaces a blocked WebFetch/WebSearch output with a safe placeholder and logs a non-blocking output decision. `stop_on_block` restores strict blocking for WebFetch/WebSearch as well.

`policy.vge_failure_mode`

FIELD	VALUES	DEFAULT
`prompt_text`	`fail_closed`, `fail_open`	`fail_closed`
`prompt_attachment`	`fail_closed`, `fail_open`	`fail_closed`
`posttool_output`	`fail_closed`, `fail_open`	`fail_closed`

Use `fail_closed` when security matters more than availability. Use `fail_open` only when operational continuity is more important for that stage and your team accepts the risk.

TROUBLESHOOTING

Claude Code Does Not Seem Protected

Run `vge-cc-guard doctor`. Check:

» Was `install --apply` run for the right scope?
» Did you restart Claude Code after installing?

» Are you in the project that has project-scoped settings?
» Does the settings file contain `vge-cc-guard hook` entries?

VGE Connection Fails

Open `vge-cc-guard config` and run Test Connection. Check:

» API URL is correct.
» API key is active.
» Network or proxy allows the VGE endpoint.
» The key has the expected environment or tenant access.

Claude Code Is Waiting For A Decision

Look for a prompt containing `Decision ID: dec_`. Reply with `1`, `2`, `3`, or an exact `vge ... dec_` command. If the session appears stuck:

```
vge-cc-guard reset-session
```

A Tool Is Always Blocked

Open Tools Policy and check the tool `gate`. `Write` and `Edit` are blocked by default. Change them only if your workflow requires it and you accept the risk.

Web Fetches Are Blocked

Check Security Baseline:

» URL Access Baseline may be enabled.
» A preset may match the URL.
» A custom deny rule may match the host, CIDR, scheme, or pattern.

If the URL is allowed locally but fetched content is blocked, check the tool-output decision. That is a content decision, not a URL-target decision.

PostTool Replacement Is Rejected Or L0 Is Disabled

If Claude is not receiving a redacted placeholder for a blocked `Read` or `Bash` output, the Claude Code contract is likely degraded. Run:

```
vge-cc-guard doctor --cc-contract
```

Check the reported state, reason, CC version, binary path, SHA prefix, and next action. In most cases auto-probe will recover the contract on its own. PostTool blocking still works via the degraded path when L0 is disabled.

IDE Panels Hide Decisions

Use terminal Claude Code for the full HITL experience. If your team must use an IDE native panel, review IDE Compatibility and consider turning off only the affected enforcement stage or affected tool-output analysis.

You Need To Share Configuration With Support

Use View Current Configuration and export a redacted config summary. Do not send raw API keys.

SAFE OPERATING RECOMMENDATIONS

Keep these defaults unless you have a specific reason to change them:

» `credential_protection: true`
» Prompt text analysis: `enforce`
» Prompt attachment analysis: `enforce`

» Subagent output analysis: `enforce`
» All VGE failure modes: `fail_closed`
» URL Access Baseline: enabled
» `WebSearch` and `WebFetch` output analysis: on
» `Read` and `Bash` output analysis: on

Relax policy gradually. Prefer changing one setting, testing the workflow, and reviewing audit events before changing more. For project teams, document any intentional deviations from defaults in your repository's onboarding notes so users understand why the policy differs.

Questions? Contact us at contact@vigilguard.ai or visit vigilguard.ai