GITHUB COPILOT AGENTIC FRAMEWORK

Hooks: intercepting the agent lifecycle.

Intercept, validate, and control agent behavior at every lifecycle point. The governance layer the agent cannot bypass: from the pre-hook that blocks to the post-hook that chains the next agent.

AUTHOR
Paula Silva

ROLE
Software Global Black Belt

DURATION
45 to 60 minutes

DATE
2026-06-11

AGENDA

Five parts, one mental model.

PART I	Concept, what hooks are, and the two analogies that unlock everything: middleware and assembly line
PART II	Types, pre-generation, post-generation, validation and custom hooks
PART III	Lifecycle, the complete flow, exit codes, the 3 handlers and multi-agent chaining
PART IV	Implementation, the real configuration, security scanning, the execution mechanics and where hooks run
PART V	Production, three real scenarios, the impact, the token economy with AI Credits, and best practices

PART

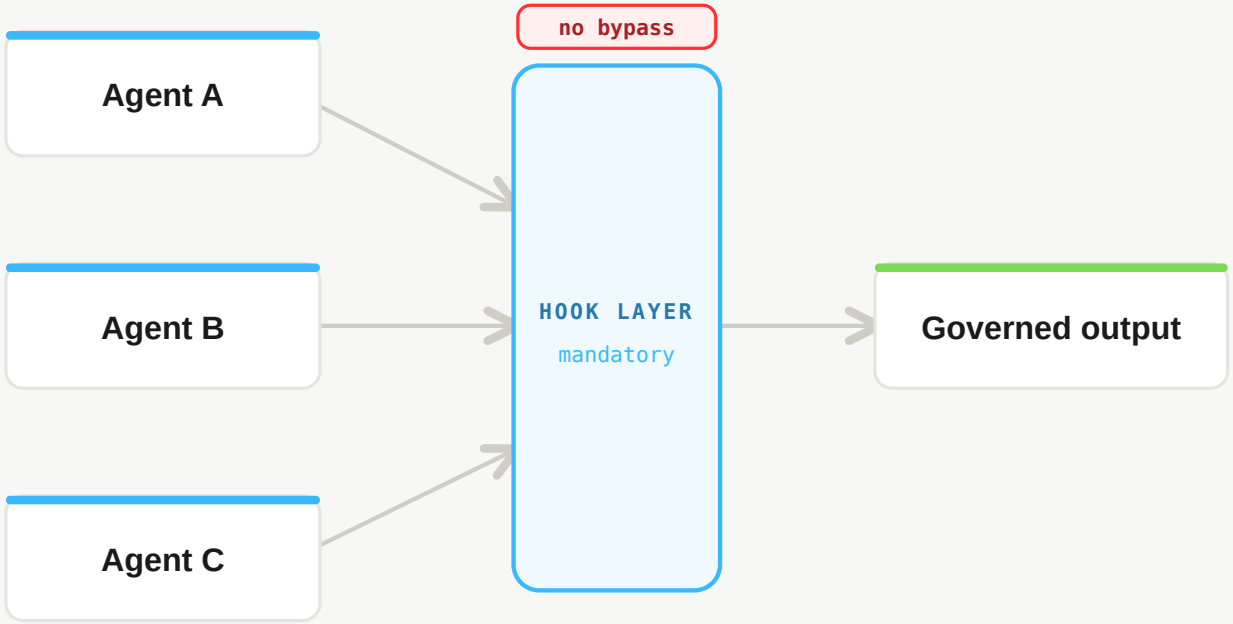
I

The concept.

The silent sensors of agentic automation. They sleep until triggered, and the agent cannot bypass them.

CONCEPT · WHAT HOOKS ARE

Passive mechanisms that respond to events in the agent lifecycle.



EVENT INTERCEPTORS

They respond to specific events in an agent's lifecycle. They sleep until triggered automatically.

MANDATORY BARRIERS

Agents CANNOT bypass them. Hooks are enforced by the system, ensuring every action follows governance standards.

ZERO OVERHEAD

Hooks only consume cycles when they actually need to run. When dormant, overhead is virtually zero.

Without hooks, each agent would manage its own validation and error handling. Hooks centralize these responsibilities, ensuring consistency across the fleet.

CONCEPT · ANALOGY 01

If you know Express.js, you already know hooks.

HTTP MIDDLEWARE

```
app.use(authMiddleware);    // PRE
app.get("/api", handler);   // EXECUTION
app.use(loggerMiddleware);  // POST
```

Intercepts HTTP requests before and after the main handler. Validates auth, logs, transforms data.

AGENT HOOKS

```
pre_hook:  validate_permissions  // PRE
execution: agent_task            // EXECUTION
post_hook: quality_check         // POST
err_hook:  fallback_handler      // ON ERROR
```

Intercepts agent actions before and after execution. Validates permissions, scans security, ensures quality.

The same middleware intuition applies: intercept, validate, transform. But for AI agents instead of HTTP requests.

CONCEPT · ANALOGY 02

Hooks as an assembly line: quality checkpoints at every stage.

ENTRY SENSOR · PRE

Checks raw material meets criteria before entering production. If it fails, blocks entry.

→

WORKSTATION · EXEC

The agent executes its main task using available skills and tools.

→

EXIT SENSOR · POST

Checks the quality of the final product. If approved, it moves on. If rejected, it reprocesses.

→

ALARM · ERROR

If something goes wrong, it stops the line, logs the incident and alerts supervisors.

Just as sensors make a factory efficient and reliable, hooks make agents governed, auditable and secure.

PART

II

The types.

Pre-generation, post-generation, validation and custom hooks. Each type guards a different moment in the cycle.

TYPES · PRE-GENERATION

Pre-generation hooks validate context and permissions BEFORE the agent acts.

hooks.json · pre-generation

```
{
  "hooks": {
    "pre-generation": [{
      "handler": "command",
      "script": "check_permissions.sh",
      "on_failure": "block",
      "timeout": 5000
    }]
  }
}
```

Validates user permissions before any generation.

Loads repo guidelines, such as the .instructions.md files.

Checks environment resources and preconditions.

Can BLOCK the action if it fails. It is the only moment to say no.

TYPES · POST-GENERATION

Post-generation hooks validate results AFTER the agent completes the task.

hooks.json · post-generation

```
{
  "hooks": {
    "post-generation": [{
      "handler": "prompt",
      "action": "Review output quality",
      "min_coverage": 80,
      "next_agent": "SecurityAgent"
    }]
  }
}
```

Validates the quality of generated results.

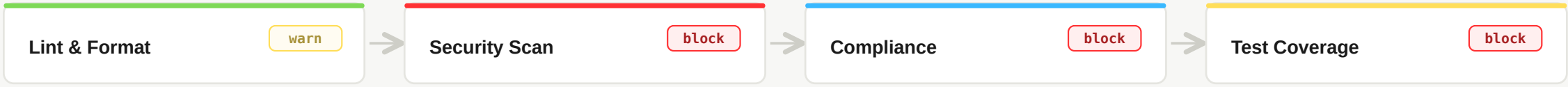
Records metrics and logs for each run.

Notifies interested systems on completion.

Triggers the next agent via chaining, creating multi-agent pipelines.

TYPES · VALIDATION

Validation hooks: quality, security and compliance as gates.



LINT & FORMAT

Validates code style, formatting and naming standards automatically.

SECURITY SCAN

Scans generated code for vulnerabilities, exposed secrets and insecure patterns.

COMPLIANCE CHECK

Checks adherence to corporate policies, LGPD, HIPAA and regulatory standards.

TEST COVERAGE

Ensures generated code has minimum test coverage before accepting.

```
hooks:
  validation:
    - name: security-scan   handler: command   script: "semgrep --config auto ."          on_failure: block
    - name: lint-check      handler: command   script: "eslint --fix ."                   on_failure: warn
    - name: coverage-gate   handler: command   script: "jest --coverageThreshold=80"     on_failure: block
```

TYPES · CUSTOM

Custom hooks: build them to order for specific needs.

NOTIFICATION HOOK

Sends alerts to Slack, Teams or email when agents complete critical tasks.

METRICS HOOK

Collects performance, cost and quality metrics for observability dashboards.

TRANSFORM HOOK

Transforms the agent's output before delivering to the user: sanitizes, formats, enriches.

Extensibility: custom hooks let organizations adapt the framework to their unique needs without modifying the system core.

PART

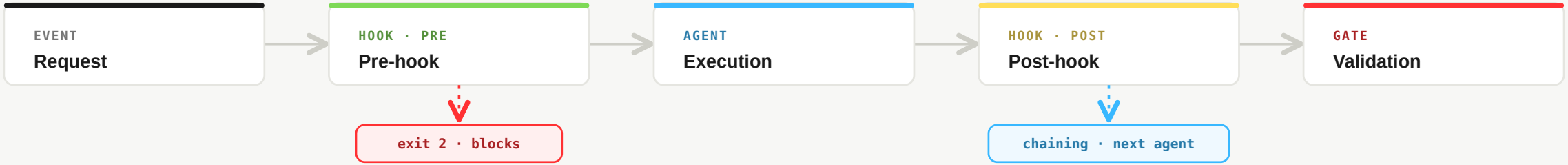
III

The lifecycle.

From request to output, every interception point. Nothing passes without verification.

LIFECYCLE · THE COMPLETE FLOW

From request to output: five points, five interception opportunities.



1 · Request

The event reaches the system: push, PR, user command or cron schedule.

2 · Pre-hook

Validates permissions, loads context, checks preconditions. Can block execution (exit code 2).

3 · Execution

The agent executes its main task using skills and tools provided by the framework.

4 · Post-hook

Validates results, records metrics, notifies systems. Can trigger the next agent via chaining.

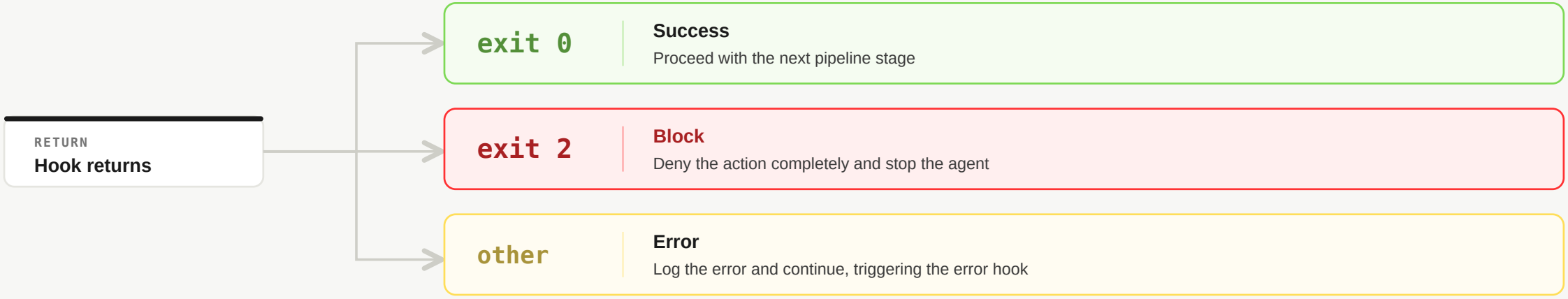
5 · Validation

Security and compliance quality gates before delivery. The output leaves audited and logged.

Every point in the lifecycle is an interception opportunity. Hooks ensure nothing passes without verification.

LIFECYCLE · EXIT CODES

Three numbers control everything: 0 proceeds, 2 blocks, other logs.



The timeline: trigger, execution, validation. Success continues, block stops, error triggers the error hook. It is the most important semantics in the system.

LIFECYCLE · THE 3 HANDLERS

Three handler types, from direct script to full subagent.

COMMAND

Direct shell scripts that execute quick actions. Ideal for simple, lightweight validations.

```
BRANCH=$(git branch --show-current)
if [[ "$BRANCH" == "main" ]]; then
  exit 2  # Block
fi
exit 0    # Continue
```

PROMPT

LLMs that analyze and make intelligent decisions. They use reasoning to evaluate complex context.

```
"Analyze the diff for:
 1. Security vulnerabilities
 2. Breaking changes
 If critical: exit 2 (block)
 Otherwise: exit 0 (approve)"
```

AGENT

Complete subagents with tools and skills. They execute complex tasks like code review.

```
Agent: SecurityReviewAgent
 Skills: [code-review, vuln-scan]
 Tools:  [semgrep, codeql, gitleaks]
 Output: structured_report.json
```

Start with command for lightweight validations. Graduate to prompt when you need reasoning, and to agent when verification is a full task.

LIFECYCLE · CHAINING

Hook chaining: the multi-agent pipeline of a pull request.

on_pull_request · chain

```
[event] pull request opened by the dev or by the Coding Agent

[lint-check] eslint + prettier validate format, block on failure
[security-scan] semgrep and CodeQL scan, exit 2 blocks the merge
[test-generation] agent generates and runs tests, coverage above 80%
[code-review] review agent analyzes logic and comments on the PR

all hooks passed: PR approved and merged
total pipeline: ~28s, in sequence, immediate stop if a blocking hook fails
```

EACH STEP ADDS VALUE

Post-hooks trigger the next agent. The pipeline becomes a chain where each link adds value.

BLOCKING FIRST

Lint and security are blocking and come first: fail fast, before spending on the expensive agents.

HUMAN OUT OF THE LOOP

28 seconds of automated pipeline against 45 minutes of manual review per PR.

PART

IV

The implementation.

The real configuration, the security scanning hook line by line, and where GitHub Copilot hooks run today.

IMPLEMENTATION · THE CONFIGURATION

A single JSON configuration defines every hook in the repository.

.github/hooks/hooks.json

```
{
  "hooks": {
    "pre-commit": [{
      "name": "validate-syntax",
      "handler": "command",
      "script": "npx eslint .",
      "on_failure": "block",
      "timeout": 10000
    }, {
      "name": "check-secrets",
      "handler": "command",
      "script": "gitleaks detect --source .",
      "on_failure": "block"
    }],
    "post-commit": [{
      "name": "notify-team",
      "handler": "command",
      "script": "curl -X POST $SLACK_WEBHOOK",
      "on_failure": "warn"
    }]
  }
}
```

ANATOMY OF A HOOK

Each hook has a name, handler, script, failure behavior and timeout. Five fields, a complete contract.

BLOCK VS WARN

on_failure block for the critical (syntax, secrets). warn for the informational (notification). Always separate them.

VERSIONED IN GIT

In .github/hooks/*.json on the default branch. Applies to every agent in the repository, inherited by the team.

IMPLEMENTATION · SECURITY SCANNING

The security hook, line by line: three scans, one verdict.

security-scan.sh

```
#!/bin/bash
# Security scanning para código gerado por agente

# 1. Secrets expostos
gitleaks detect --source . --no-git
[ $? -ne 0 ] && exit 2  # BLOCK

# 2. Análise estática OWASP
semgrep --config "p/owasp-top-ten" --error .
[ $? -ne 0 ] && exit 2  # BLOCK

# 3. Auditoria de dependências
npm audit --audit-level=critical
[ $? -ne 0 ] && exit 2  # BLOCK

exit 0  # CONTINUE: todas as varreduras passaram
```

GITLEAKS

Detects exposed secrets and credentials before they reach the commit.

SEMGREP

Scans generated code against the OWASP top ten. A critical vulnerability blocks.

NPM AUDIT

Audits dependencies and stops critical-level vulnerabilities in the chain.

The script can run multiple checks in sequence. Exit code 2 blocks, exit code 0 continues. Simple to audit, impossible to skip.

IMPLEMENTATION · EXECUTION MECHANICS

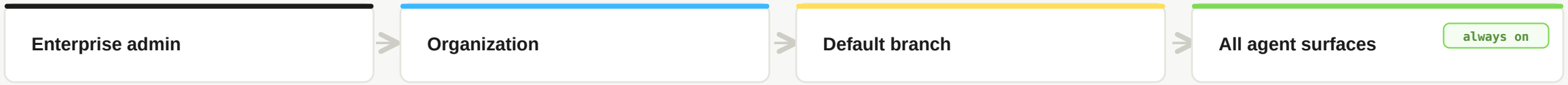
The fine mechanics: sequential, with timeouts, and fail-secure where it matters.

Sequential	Hooks for the same event run in array order. The first deny skips the rest.
Under 5s	The official execution recommendation. The default timeout is 30 seconds per hook.
Timeout does not crash	If it expires: the hook is terminated, the event is logged and the agent continues, for most events.
preToolUse is fail-secure	The exception: errors, crashes or timeouts DENY the tool instead of letting it through. A broken hook never becomes an open door.
Never log secrets	Prompts and tool arguments can contain sensitive data. Redact before persisting or forwarding.

In the CLI, disableAllHooks pauses the configuration without deleting anything: useful for debugging, sensitive releases, or letting a contributor opt out locally.

IMPLEMENTATION · WHERE THEY RUN

GitHub Copilot hooks run on every surface of the agent.



SURFACE	STATUS	NOTE
GitHub GitHub Copilot coding agent	GA	Cloud agent via Issues and PRs, hooks on the default branch
GitHub Copilot CLI	GA	In the terminal, with extra personal hooks in ~/.copilot/hooks
VS Code	Preview	In preview since March 2026, with 8 events
JetBrains IDEs	Preview	Agent hooks in preview since March 2026

Since June 2026, admins distribute managed hooks across the whole enterprise: the same configuration, always on, in every GitHub Copilot client in the organization.

PART

V

Production.

Three real scenarios, the measured impact with versus without hooks, and the practices that separate operators from firefighters.

PRODUCTION · THREE SCENARIOS

Three teams, three problems, the same mechanism.

MARIA · DEVOPS, FINTECH

CI/CD with hooks

Branch protection in the pre-hook, semgrep and gitleaks as blocking, post-hooks triggering the full CI pipeline.

Zero vulnerabilities in production in 6 months

ANA · TECH LEAD, E-COMMERCE

Code quality with hooks

ESLint and Prettier as blocking, an agent hook generates tests and validates coverage above 80%, mandatory JSDoc.

Quality score from 72% to 96%, onboarding 60% shorter

SOFIA · SECURITY, HEALTHCARE

HIPAA compliance

An LLM hook analyzes every diff for HIPAA patterns, a chain of SAST + SCA + secrets + DAST, audit with hash and timestamp.

100% compliance across 3 audits, review from 2h to 30s

The pattern is the same: pre-hook validates, blocking scans, post-hook chains, custom hook measures. Only the domain changes.

PRODUCTION · THE IMPACT

With versus without hooks: the numbers that close the case.

WITHOUT HOOKS

- Manual code review: 45 minutes per PR.
- Vulnerabilities detected late, already in production.
- Style inconsistency across teams and agents.

→

WITH HOOKS

- ✓ Automated pipeline: 28 seconds per PR.
- ✓ Vulnerabilities blocked before the merge.
- ✓ 100% style consistency, continuous compliance.

0x

FASTER: 45MIN TO 28S PER PR

0

VULNERABILITIES IN PRODUCTION IN THE LAST 6 MONTHS

100%

AUDIT COVERAGE: EVERY ACTION LOGGED

PRODUCTION · TOKEN ECONOMY

Since June 1st, tokens are money. Hooks are the cost control.

RULE OUTSIDE THE CONTEXT

A rule in instructions occupies the context and is billed as input on every request. The same rule in a hook runs as shell: zero token cost.

BLOCKING EARLY IS CHEAP

The preToolUse deny cuts the execute, fail, analyze, retry cycle. Every avoided loop is input and output that never reaches the bill.

RESPONSE WITHOUT LLM

In the CLI, a userPromptSubmitted hook can return the response directly on stdout and skip the model entirely. Zero tokens for deterministic answers.

LEAN OUTPUT

The hook's stdout enters the agent's context. A verbose hook adds billed tokens. Quiet on success, specific on failure.

Since June 1st, 2026, GitHub Copilot bills via AI Credits metered in input, output and cached tokens, with 1 credit worth one US cent. Every wasted agent loop now shows up on the bill.

PRODUCTION · BEST PRACTICES

Five practices and one anti-pattern from people running hooks in production.

Fail fast	Put blocking hooks first in the chain.	`Lint → Security → Tests`
Explicit timeout	Always set a timeout to avoid stuck hooks.	`"timeout": 10000`
Idempotency	Hooks must produce the same result if executed multiple times.	`re-run = same output`
Log everything	Every hook execution must produce an auditable log.	`JSON structured logs`
Blocking vs warning	on_failure block for critical, warn for informational.	`block | warn`
Anti-pattern	Never put slow hooks (over 30s) as blocking in interactive flows. Use async hooks or move them to post-commit.	

SUMMARY

Six key concepts to take home.

HOOKS

Passive interceptors in the agent lifecycle. Centralized governance.

PRE-HOOKS

Validate before the action. Can block execution with exit 2.

POST-HOOKS

Validate results. Trigger chaining to the next agent.

ERROR HOOKS

Capture failures, attempt fallback, notify the team.

HANDLERS

Command (shell), Prompt (LLM), Agent (full subagent).

CHAINING

Hooks create multi-agent pipelines. Each step adds value.

CLOSING

The agent cannot bypass it. That is the point.

CONTACT

Paula Silva
Software Global Black Belt
linkedin.com/in/paulanunes

NEXT STEP

One blocking hook, today

a check-secrets with gitleaks, in your repo, this week

.GITHUB/HOOKS/HOOKS.JSON · START HERE

```
"pre-commit": [{
  "name": "check-secrets",
  "handler": "command",
  "script": "gitleaks detect --source .",
  "on_failure": "block"
}]
# and no secret gets through again
```


Published 2026-06-11 · v2.1