

GITHUB COPILOT AGENTIC FRAMEWORK

Hooks: interceptando o ciclo de vida do agente.

Intercepte, valide e controle o comportamento de agentes em cada ponto do ciclo de vida. A camada de governança que o agente não pode contornar: do pre-hook que bloqueia ao post-hook que encadeia o próximo agente.

AUTORA

Paula Silva

FUNÇÃO

Software Global Black Belt

DURAÇÃO

45 a 60 minutos

DATA

2026-06-11

AGENDA

Cinco partes, um modelo mental.

PARTE I Conceito, o que são hooks, e as duas analogias que destravam tudo: middleware e linha de montagem

PARTE II Tipos, pre-generation, post-generation, validação e hooks customizados

PARTE III Ciclo de vida, o fluxo completo, exit codes, os 3 handlers e o chaining multi-agente

PARTE IV Implementação, a configuração real, o security scanning, a mecânica de execução e onde os hooks rodam

PARTE V Produção, três cenários reais, o impacto, a economia de tokens com AI Credits e as best practices



PARTE

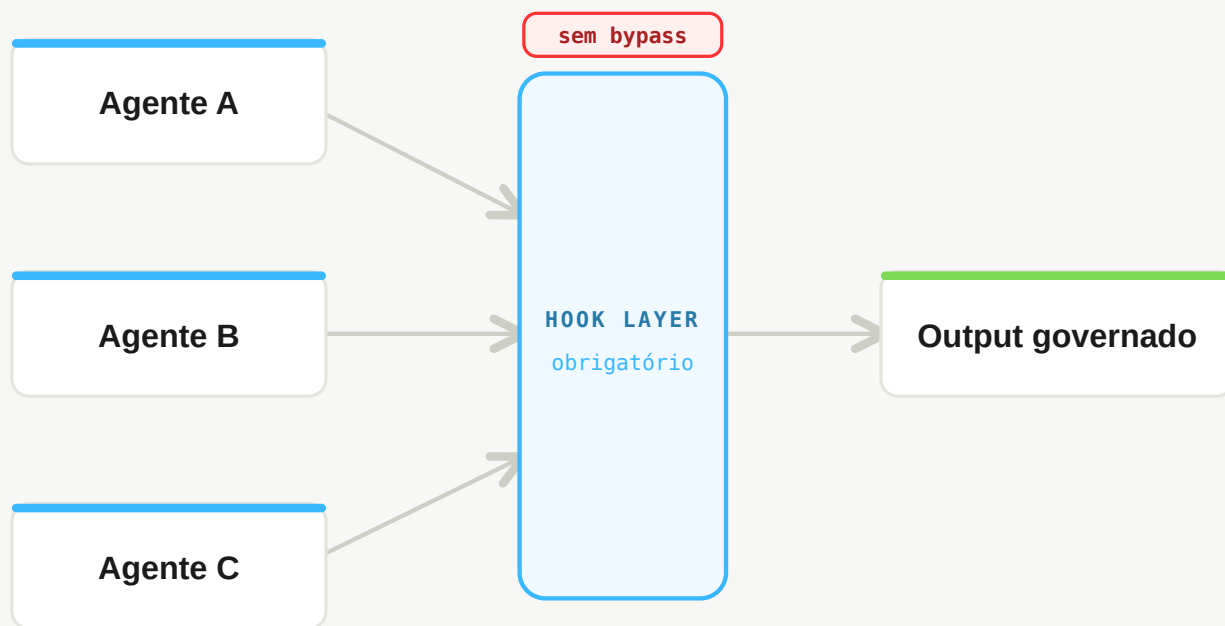


O conceito.

Os sensores silenciosos da automação agêntica. Dormem até serem ativados, e o agente não pode contorná-los.

CONCEITO • O QUE SÃO HOOKS

Mecanismos passivos que respondem a eventos do ciclo de vida do agente.



INTERCEPTADORES DE EVENTO

Respondem a eventos específicos no ciclo de vida de um agente. Dormem até serem ativados automaticamente.

BARREIRAS OBRIGATÓRIAS

Agentes NÃO podem contorná-los. Hooks são impostos pelo sistema, garantindo que toda ação siga padrões de governança.

ZERO OVERHEAD

Hooks só custam ciclos quando realmente precisam executar. Quando dormem, o overhead é virtualmente zero.

Sem hooks, cada agente gerenciaria sua própria validação e tratamento de erros. Hooks centralizam essas responsabilidades, garantindo consistência em toda a frota.

Se você conhece Express.js, já conhece hooks.

MIDDLEWARE HTTP

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

Intercepta requests HTTP antes e depois do handler principal. Valida auth, loga, transforma dados.

HOOKS DE AGENTE

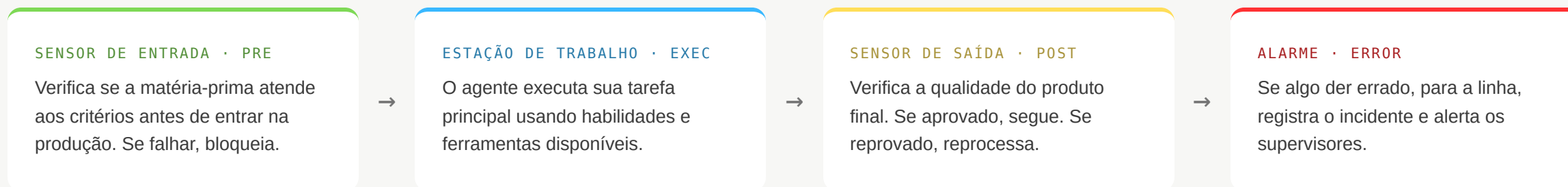
```
pre_hook:  validate_permissions  // PRE
execution: agent_task            // EXECUTION
post_hook: quality_check        // POST
err_hook:  fallback_handler     // ON ERROR
```

Intercepta ações de agentes antes e depois da execução. Valida permissões, escaneia segurança, garante qualidade.

A mesma intuição de middleware se aplica: interceptar, validar, transformar. Mas para agentes de IA em vez de HTTP requests.

CONCEITO · ANALOGIA 02

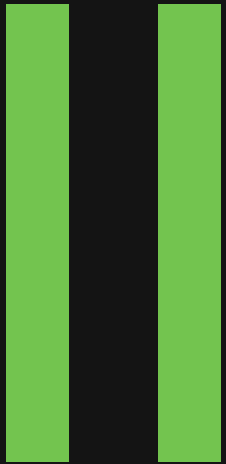
Hooks como linha de montagem: checkpoints de qualidade em cada etapa.



Assim como sensores tornam uma fábrica eficiente e confiável, hooks tornam agentes governados, auditáveis e seguros.



PARTE



Os tipos.

Pre-generation, post-generation, validação e hooks customizados. Cada tipo guarda um momento diferente do ciclo.

TIPOS • PRE-GENERATION

Pre-generation hooks validam contexto e permissões ANTES do agente agir.

hooks.json • pre-generation

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

Valida permissões do usuário antes de qualquer geração.

Carrega diretrizes do repo, como os arquivos .instructions.md.

Verifica recursos e pré-condições do ambiente.

Pode BLOQUEAR a ação se falhar. É o único momento de dizer não.

TIPOS • POST-GENERATION

Post-generation hooks validam resultados APÓS o agente completar a tarefa.

```
hooks.json • post-generation

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

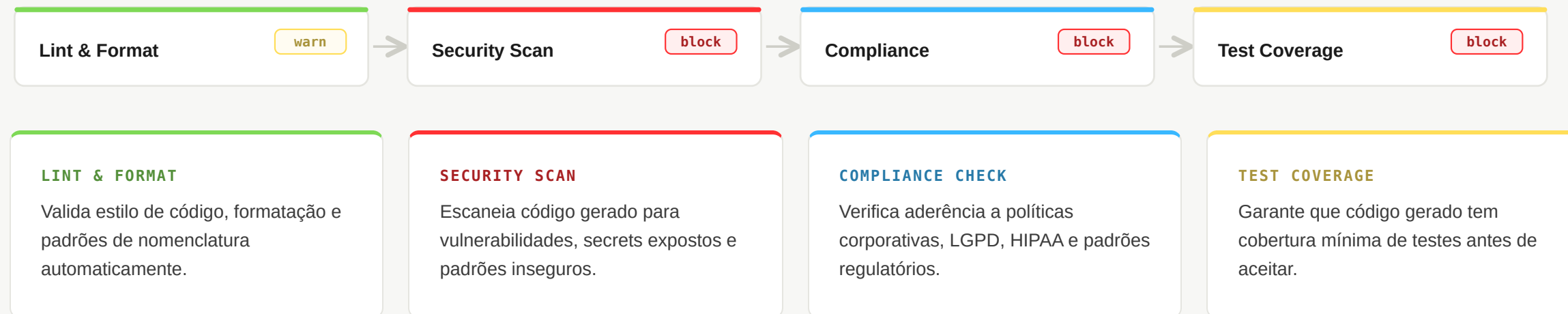
Valida a qualidade dos resultados gerados.

Registra métricas e logs de cada execução.

Notifica sistemas interessados ao concluir.

Dispara o próximo agente via chaining, criando pipelines multi-agente.

Validation hooks: qualidade, segurança e compliance como portões.



```
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
```

TIPOS • CUSTOMIZADOS

Custom hooks: crie sob medida para necessidades específicas.

NOTIFICATION HOOK

Envia alertas para Slack, Teams ou email quando agentes completam tarefas críticas.

METRICS HOOK

Coleta métricas de performance, custo e qualidade para dashboards de observabilidade.

TRANSFORM HOOK

Transforma o output do agente antes de entregar ao usuário: sanitiza, formata, enriquece.

Extensibilidade: custom hooks permitem que organizações adaptem o framework às suas necessidades únicas sem modificar o core do sistema.



PARTE

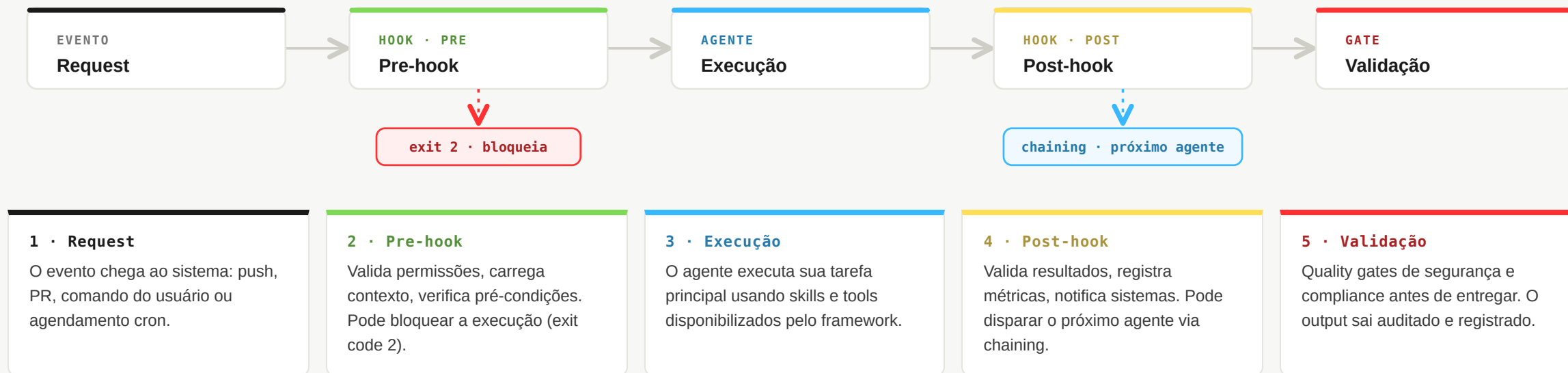


O ciclo de vida.

Da requisição ao output, todos os pontos de interceptação. Nada passa sem verificação.

CICLO DE VIDA · O FLUXO COMPLETO

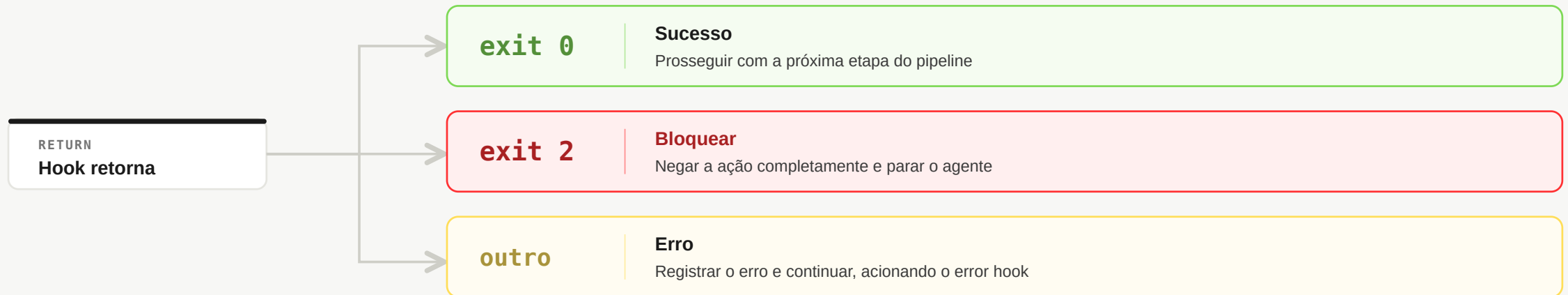
Da requisição ao output: cinco pontos, cinco oportunidades de interceptação.



Cada ponto no ciclo de vida é uma oportunidade de interceptação. Hooks garantem que nada passe sem verificação.

CICLO DE VIDA · EXIT CODES

Três números controlam tudo: 0 prossegue, 2 bloqueia, outro registra.



A timeline: disparo, execução, validação. Sucesso continua, bloqueio para, erro aciona o error hook. É a semântica mais importante do sistema.

Três tipos de manipuladores, do script direto ao subagente completo.

COMMAND

Scripts de shell diretos que executam ações rápidas.
Ideais para validações simples e leves.

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

PROMPT

LLMs que analisam e tomam decisões inteligentes.
Usam raciocínio para avaliar contexto complexo.

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

AGENT

Subagentes completos com ferramentas e habilidades. Executam tarefas complexas como revisão de código.

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

Comece com command para validações leves. Gradue para prompt quando precisar de raciocínio, e para agent quando a verificação for uma tarefa completa.

Hook chaining: o pipeline multi-agente de um pull request.

on_pull_request • chain

[event] pull request aberto pelo dev ou pelo Coding Agent

[lint-check] eslint + prettier validam formato, bloqueiam se falhar

[security-scan] semgrep e CodeQL escaneiam, exit 2 bloqueia o merge

[test-generation] agente gera e executa testes, cobertura acima de 80%

[code-review] agente de revisão analisa lógica e comenta no PR

todos os hooks passaram: PR aprovado e merged

pipeline total: ~28s, em sequência, parada imediata se um blocking falhar

CADA ETAPA AGREGA

Post-hooks disparam o próximo agente. O pipeline vira uma cadeia onde cada elo adiciona valor.

BLOCKING PRIMEIRO

Lint e security são blocking e vêm primeiro: falhe rápido, antes de gastar com os agentes caros.

HUMANO FORA DO LOOP

28 segundos de pipeline automatizado contra 45 minutos de revisão manual por PR.

PARTE

IV

A implementação.

A configuração real, o hook de security scanning linha a linha, e onde os hooks do GitHub Copilot rodam hoje.

Uma única configuração JSON define todos os hooks do repositório.

```
.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"
    }]
  }
}
```

ANATOMIA DE UM HOOK

Cada hook tem nome, handler, script, comportamento em falha e timeout. Cinco campos, contrato completo.

BLOCK VS WARN

on_failure block para o crítico (sintaxe, secrets). warn para o informativo (notificação). Separe sempre.

VERSIONADO NO GIT

Em .github/hooks/*.json no branch padrão. Vale para todo agente do repositório, herdado pelo time.

O hook de segurança, linha a linha: três varreduras, um veredito.

```
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

Detecta secrets e credenciais expostas antes de chegarem ao commit.

SEMGREP

Varre o código gerado contra o OWASP top ten. Vulnerabilidade crítica bloqueia.

NPM AUDIT

Audita dependências e barra vulnerabilidades de nível crítico na cadeia.

O script pode executar múltiplas verificações em sequência. Exit code 2 bloqueia, exit code 0 continua. Simples de auditar, impossível de pular.

A mecânica fina: sequencial, com timeout, e fail-secure onde importa.

Sequencial

Hooks do mesmo evento rodam na ordem do array. O primeiro deny pula os demais.

Abaixo de 5s

Recomendação oficial de execução. O timeout padrão é 30 segundos por hook.

Timeout não derruba

Estourou: o hook é encerrado, o evento é registrado e o agente continua, na maioria dos eventos.

preToolUse é fail-secure

A exceção: erro, crash ou timeout NEGAM a tool em vez de deixar passar. Um hook quebrado não vira porta aberta.

Nunca logue secrets

Prompts e argumentos de tool podem conter dados sensíveis. Redija antes de persistir ou encaminhar.

No CLI, `disableAllHooks` pausa a configuração sem apagar nada: útil em debug, em releases sensíveis, ou para um contribuidor fazer opt-out local.

Hooks do GitHub Copilot rodam em todas as superfícies do agente.



SUPERFÍCIE	STATUS	OBSERVAÇÃO
GitHub GitHub Copilot coding agent	GA	Agente em nuvem via Issues e PRs, hooks no branch padrão
GitHub Copilot CLI	GA	No terminal, com hooks pessoais extras em ~/.copilot/hooks
VS Code	Preview	Em preview desde março de 2026, com 8 eventos
JetBrains IDEs	Preview	Agent hooks em preview desde março de 2026

Desde junho de 2026, admins distribuem hooks gerenciados por todo o enterprise: a mesma configuração, sempre ativa, em cada cliente do GitHub Copilot da organização.

PARTE



A produção.

Três cenários reais, o impacto medido com versus sem hooks, e as práticas que separam quem opera de quem apaga incêndio.

PRODUÇÃO · TRÊS CENÁRIOS

Três times, três problemas, o mesmo mecanismo.

MARIA · DEVOPS, FINTECH

CI/CD com hooks

Branch protection no pre-hook, semgrep e gitleaks como blocking, post-hooks disparando o pipeline completo de CI.

Zero vulnerabilidades em produção em 6 meses

ANA · TECH LEAD, E-COMMERCE

Code quality com hooks

ESLint e Prettier como blocking, hook agent gera testes e valida cobertura acima de 80%, JSDoc obrigatório.

Quality score de 72% para 96%, onboarding 60% menor

SOFIA · SECURITY, HEALTHCARE

Compliance HIPAA

Hook com LLM analisa cada diff por padrões HIPAA, cadeia SAST + SCA + secrets + DAST, auditoria com hash e timestamp.

100% compliance em 3 auditorias, review de 2h para 30s

O padrão é o mesmo: pre-hook valida, blocking escaneia, post-hook encadeia, custom hook mede. O que muda é só o domínio.

Com versus sem hooks: os números que fecham o caso.

SEM HOOKS

- Revisão manual de código: 45 minutos por PR.
- Vulnerabilidades detectadas tarde, já em produção.
- Inconsistência de estilo entre times e agentes.



COM HOOKS

- ✓ Pipeline automatizado: 28 segundos por PR.
- ✓ Vulnerabilidades bloqueadas antes do merge.
- ✓ 100% de consistência de estilo, compliance contínuo.

0x

MAIS RÁPIDO: 45MIN PARA 28S POR PR

0

VULNERABILIDADES EM PRODUÇÃO NOS ÚLTIMOS 6 MESES

100%

COBERTURA DE AUDITORIA: CADA AÇÃO REGISTRADA

Desde 1º de junho, tokens são dinheiro. Hooks são o controle de custo.

REGRA FORA DO CONTEXTO

Uma regra em instructions ocupa o contexto e é cobrada como input em cada request. A mesma regra num hook roda como shell: custo zero de token.

BLOQUEAR CEDO É BARATO

O deny do preToolUse corta o ciclo executar, falhar, analisar, tentar de novo. Cada loop evitado é input e output que não vai para a fatura.

RESPOSTA SEM LLM

No CLI, um hook de userPromptSubmitted pode devolver a resposta direto no stdout e pular o modelo inteiro. Zero tokens para respostas determinísticas.

OUTPUT ENXUTO

O stdout do hook entra no contexto do agente. Hook verboso adiciona tokens cobrados. Silencioso no sucesso, específico na falha.

Desde 1º de junho de 2026, o GitHub Copilot cobra por AI Credits medidos em tokens de input, output e cache, com 1 crédito valendo um centavo de dólar. Cada loop desperdiçado do agente agora aparece na fatura.

Cinco práticas e um anti-pattern de quem roda hooks em produção.

Falhe rápido

Coloque hooks blocking primeiro na cadeia.

Lint → Security → Tests

Timeout explícito

Sempre defina timeout para evitar hooks travados.

```
"timeout": 10000
```

Idempotência

Hooks devem produzir o mesmo resultado se executados múltiplas vezes.

re-executar = mesmo output

Log tudo

Cada execução de hook deve gerar log auditável.

JSON structured logs

Blocking vs warning

on_failure block para críticos, warn para informativos.

block | warn

Anti-pattern

Nunca coloque hooks lentos (mais de 30s) como blocking em fluxos interativos. Use async hooks ou mova para post-commit.

RESUMO

Seis conceitos-chave para levar para casa.

HOOKS

Interceptadores passivos no lifecycle do agente.
Governança centralizada.

PRE-HOOKS

Validam antes da ação. Podem bloquear a execução com exit 2.

POST-HOOKS

Validam resultados. Disparam chaining para o próximo agente.

ERROR HOOKS

Capturam falhas, tentam fallback, notificam a equipe.

HANDLERS

Command (shell), Prompt (LLM), Agent (subagente completo).

CHAINING

Hooks criam pipelines multi-agente. Cada etapa adiciona valor.

ENCERRAMENTO

O agente não pode contornar. Esse é o ponto.

CONTATO

Paula Silva

Software Global Black Belt

linkedin.com/in/paulanunes

PRÓXIMO PASSO

Um hook blocking, hoje

um `check-secrets` com `gitleaks`, no seu repo, esta semana

Publicado em 2026-06-11 · v2.1

`.GITHUB/HOOKS/HOOKS.JSON` · COMECE AQUI

```
"pre-commit": [{  
  "name": "check-secrets",  
  "handler": "command",  
  "script": "gitleaks detect --source .",  
  "on_failure": "block"  
}]  
# e nenhum secret passa de novo
```