PreToolUse Hook en Claude Code

Guía práctica — Bloqueo de acceso a archivos sensibles

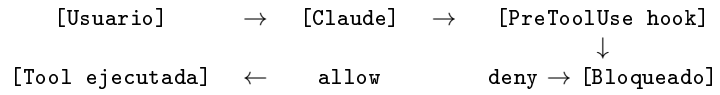
Marco Alice · marco.alice.wrk@gmail.com · Marzo 2026

¿Qué es un PreToolUse hook?

Un hook es un script externo que Claude Code ejecuta **antes** de invocar cualquier herramienta (`Read`, `Grep`, `Bash`, `Write`, etc.). Recibe via `stdin` un JSON con el nombre de la tool y sus parámetros, y devuelve una decisión: **permitir** o **denegar**.

> **Principio fundamental:** el agente no decide si puede acceder a un recurso sensible. El hook lo decide antes de que lo intente. Tus políticas de seguridad como restricciones ejecutables, no como suposiciones.

Flujo de ejecución



Implementación — Bloqueo de archivos sensibles

Paso 1 — Crear el directorio y el script

El script Python es idéntico en los tres sistemas operativos. Solo difieren los comandos de setup inicial.

Linux / macOS (terminal):

```
mkdir -p .claude/hooks
touch .claude/hooks/block_sensitive_files.py
chmod +x .claude/hooks/block_sensitive_files.py
```

Windows (PowerShell):

```
New-Item -ItemType Directory -Force -Path .claude/hooks
New-Item -ItemType File -Path .claude/hooks/block_sensitive_files.py
# chmod no aplica; Claude Code invoca python directamente via settings.json
```

Windows (CMD):

```
mkdir .claude\hooks
type nul > .claude\hooks\block_sensitive_files.py
```

A continuacion, copia el siguiente contenido en `block_sensitive_files.py`:

Listing 1: block_sensitive_files.py — compatible Linux / macOS / Windows

```
#!/usr/bin/env python3
# Nota Windows: el shebang es ignorado. Claude Code invoca el script
# via "python .claude/hooks/block_sensitive_files.py" en settings.json.
"""
PreToolUse hook: bloquea acceso a archivos y busquedas sobre rutas sensibles.
Herramientas interceptadas: Read, Grep, Bash.
"""
import json
import re
import sys

SENSITIVE_FILE_PATTERNS = [
    r"\.env$",                # .env exacto
    r"\.env\.",               # .env.production, .env.local, etc.
    r"\.pem$",                # Certificados PEM
    r"\.key$",                # Claves privadas
    r"\.p12$", r"\.pfx$",    # Keystores
    r"secrets?[/\\]",         # Carpetas secrets/ o secret/
    r"credentials?[/\\]",     # Carpetas credentials/
    r"\.aws[/\\]credentials",
    r"\.ssh[/\\]",            # Claves SSH
    r"id_rsa$", r"id_ed25519$",
    r"\.kubeconfig$",
    r"vault[-_]token",
```

```
]

SENSITIVE_BASH_PATTERNS = [
    r"cat\s+.*\.env",
    r"echo\s+.*\$[A-Z_]*(?:KEY|SECRET|TOKEN|PASSWORD|PASS|PWD)",
    r"printenv.*(?:KEY|SECRET|TOKEN|PASSWORD)",
    r"env\s*\|.*grep.*(?:KEY|SECRET|TOKEN|PASSWORD)",
]

COMPILED_FILE = [re.compile(p, re.IGNORECASE) for p in SENSITIVE_FILE_PATTERNS]
COMPILED_BASH = [re.compile(p, re.IGNORECASE) for p in SENSITIVE_BASH_PATTERNS]


def is_sensitive_path(path: str) -> bool:
    return any(p.search(path) for p in COMPILED_FILE)

def is_sensitive_bash(command: str) -> bool:
    return any(p.search(command) for p in COMPILED_BASH)

def deny(reason: str) -> None:
    output = {
        "hookSpecificOutput": {
            "hookEventName": "PreToolUse",
            "permissionDecision": "deny",
            "permissionDecisionReason": reason,
        }
    }
    print(json.dumps(output))
    sys.exit(0)

def allow() -> None:
    sys.exit(0)

def main() -> None:
    try:
        payload = json.load(sys.stdin)
    except json.JSONDecodeError:
        sys.exit(0)

    tool_name  = payload.get("tool_name", "")
    tool_input = payload.get("tool_input", {})

    if tool_name == "Read":
        path = tool_input.get("file_path", "")
        if is_sensitive_path(path):
            deny(f"Acceso bloqueado: '{path}' contiene informacion sensible.")

    elif tool_name == "Grep":
        path = tool_input.get("path", "")
        if is_sensitive_path(path):
            deny(f"Busqueda bloqueada: la ruta '{path}' puede contener credenciales.")

    elif tool_name == "Bash":
        command = tool_input.get("command", "")
        if is_sensitive_bash(command):
            deny("Comando bloqueado: intenta leer credenciales directamente.")

    allow()

if __name__ == "__main__":
    main()
```

Paso 2 — Registrar el hook en settings.json

El campo command difiere segun el sistema operativo. Usa la variante correspondiente:

Linux / macOS:

Listing 2: .claude/settings.json — Linux / macOS

```
{
  "hooks": {
    "PreToolUse": [
      {
        "matcher": "Read|Grep|Bash",
        "hooks": [
```

```
          {
            "type": "command",
            "command": "python3 .claude/hooks/block_sensitive_files.py",
            "timeout": 10
          }
        ]
      }
    ]
  }
}
```

Windows (usa `python` en lugar de `python3`, y backslash en la ruta):

Listing 3: .claude/settings.json — Windows

```
{
  "hooks": {
    "PreToolUse": [
      {
        "matcher": "Read|Grep|Bash",
        "hooks": [
          {
            "type": "command",
            "command": "python .claude\\hooks\\block_sensitive_files.py",
            "timeout": 10
          }
        ]
      }
    ]
  }
}
```

Nota: si en Windows tienes Python instalado via Microsoft Store o pyenv-win, verifica con `python --version` en PowerShell. Algunos entornos exponen `python3` correctamente; en ese caso usa la variante Linux/macOS.

Paso 3 — Verificar

En Claude Code, solicita que lea el `.env`:

```
> lee el contenido de .env
```

Respuesta esperada del agente: *"Acceso bloqueado: '.env' contiene información sensible..."* — la lectura no se ejecuta.

Referencia rápida — Patrones cubiertos

Patrón	Herramienta	Descripción
`.env, .env.*`	Read, Grep	Variables de entorno
`.pem, .key, .p12`	Read	Certificados y claves privadas
`secrets/, credentials/`	Read, Grep	Directorios de secretos
`.aws/credentials`	Read, Grep	Credenciales AWS
`.ssh/, id_rsa`	Read	Claves SSH
`.kubeconfig`	Read	Configuración Kubernetes
`vault-token`	Read, Grep	Tokens Vault
`cat .env (bash)`	Bash	Lectura directa via shell
`printenv / echo $TOKEN`	Bash	Exposición de variables

Separadores de ruta en Windows: los patrones `secrets?[/\\]` y `.aws[/\\]` ya cubren tanto `/` como `\` gracias al alternador regex. No es necesario modificar el script para Windows.

Buenas prácticas

- **El hook como primera línea, no como única.** Combínalo con `.gitignore` actualizado, gestores de secretos (Vault, AWS Secrets Manager) y variables de entorno inyectadas en runtime.
- **No bloquees en exceso.** Un hook demasiado restrictivo degrada la utilidad del agente. Bloquea rutas concretas, no categorías genéricas.

- **Loggea los bloqueos.** Añade escritura a un archivo de log en la función `deny()` para auditoría.
- **Versiona los hooks.** El directorio `.claude/hooks/` debe estar bajo control de versiones como cualquier otro código de infraestructura.
- **Timeout conservador.** 10 segundos es suficiente para lógica de validación. Si necesitas más, el hook está haciendo demasiado.

Importante: Los hooks ejecutan código con los mismos permisos que el proceso de Claude Code. Revisa y audita cualquier hook de terceros antes de activarlo, especialmente los que interactúan con la red.

Marco Alice · Python Developer & Data Engineer · Córdoba, España
linkedin.com/in/marcoalice · github.com/marcoalc-uco · Basado en el curso *Claude Code in Action* — Anthropic, 2026