Cómo razona Claude Security cuando escanea tu código (y por qué eso cambia todo)

No es un buscador de patrones más rápido. Es un sistema que entiende lo que hace tu código, no solo cómo está escrito.

El problema que nadie quiere admitir sobre los SAST tradicionales

Hay una conversación que se repite en casi todos los equipos de desarrollo que conozco. El equipo de seguridad instala una herramienta SAST, la ejecuta sobre el repositorio y obtiene un informe con 300 alertas. El equipo de desarrollo lo abre, ve que el 80% son falsos positivos y, a partir de ese momento, empieza a ignorar los informes. La herramienta sigue corriendo en el pipeline. Los resultados siguen llegando. Nadie los lee.

Esto no es un problema de actitud ni de cultura de seguridad. Es un problema de diseño. Los SAST tradicionales fueron construidos con una lógica concreta: buscar patrones de código que históricamente han sido peligrosos. Esa lógica tiene valor, pero también tiene un límite muy claro. Y en 2026, ese límite ya es demasiado visible para seguir ignorándolo.

Cómo funciona un SAST tradicional: la lógica del patrón

Para entender qué cambia con Claude Security, primero hay que entender bien qué hace un SAST clásico.

Un analizador estático tradicional opera, en esencia, como un buscador de patrones sofisticado. Lee el árbol sintáctico de tu código —su estructura formal— y lo compara con una base de reglas que define "esto parece peligroso". Si encuentras una llamada a `eval()`, marca alerta. Si ves que una variable que viene de una request HTTP termina directamente en una query SQL sin pasar por ninguna función conocida de sanitización, marca alerta. Si detectas una función que gestiona contraseñas y no incluye ningún hash reconocible, marca alerta.

La clave está en ese "parece". El SAST clásico trabaja con aproximaciones visuales al código, no con comprensión del comportamiento real. No sabe si esa variable que viene del usuario realmente puede llegar a tu base de datos en tiempo de ejecución. No sabe si la función de sanitización que usas, aunque tenga un nombre no convencional, hace exactamente lo que debe. No sabe si esa lógica que parece insegura está detrás de tres capas de autorización que el escáner nunca llegó a conectar.

El resultado es predecible: muchas alertas sobre cosas que no son vulnerabilidades reales, y —lo que es más preocupante— silencio sobre vulnerabilidades reales que no encajan en ningún patrón conocido.

Cómo razona Claude Security: seguir el flujo, no buscar la forma

El cambio conceptual que introduce Claude Security no es de velocidad ni de cobertura de reglas. Es de modelo mental.

Mientras un SAST tradicional pregunta *"¿este fragmento de código se parece a algo peligroso?"*, Claude pregunta *"¿qué puede hacer realmente este dato a lo largo de toda su vida en el sistema?"*. La diferencia parece sutil en el enunciado, pero es radical en la práctica.

Claude Security construye un modelo del flujo de datos a través del código. No solo ve la línea donde una variable entra en el sistema. La sigue: observa cómo se transforma, a qué funciones se pasa, qué validaciones atraviesa —o no—, cómo se combina con otros datos, y finalmente dónde termina. Si ese dato termina en un lugar donde podría causar daño sin haber pasado por una validación apropiada, lo detecta. Aunque el patrón local no sea reconocible. Aunque la vulnerabilidad esté distribuida en varios archivos. Aunque el camino entre la entrada y el problema tenga tres saltos y cruce dos módulos distintos.

Esto también cambia cómo se evalúa el contexto de autorización. Un SAST tradicional puede ver que un endpoint no tiene una comprobación explícita de permisos en su función principal y marcarlo como vulnerable. Claude puede ver que ese endpoint está montado detrás de un middleware de autenticación que opera a nivel de router, y que ese middleware ya hace la validación que se necesita. No es una regla. Es comprensión de la arquitectura real del sistema.

El caso concreto: la API que expone datos de más

Vamos a un ejemplo real, porque esto se entiende mucho mejor en código que en abstracto.

Imagina una API de gestión de usuarios con esta estructura:

```
/api/v1/users/{id}/profile /api/v1/users/{id}/documents
/api/v1/users/{id}/financials
```

El endpoint `/profile` devuelve información pública del usuario. El endpoint `/financials` devuelve datos económicos sensibles y, por diseño, solo debería ser accesible por el propio usuario o por administradores.

Ahora el problema. En el controlador de `/financials` hay una validación que comprueba si el usuario autenticado tiene permisos. Aparentemente correcta. Pero esa validación llama a una función auxiliar en `utils/permissions.js` que, a su vez, obtiene el rol del usuario de un objeto de sesión que fue construido en `middleware/auth.js`. Hasta aquí, bien.

El problema está tres archivos más abajo. En `services/userData.js`, la función que construye la respuesta final del endpoint tiene un método de serialización heredado de una versión anterior de la API. Ese método fue diseñado para el endpoint `/profile`, que es público. Nadie actualizó el serializador cuando `/financials` empezó a usarlo. El resultado es que incluso cuando la validación de permisos funciona correctamente, la respuesta que sale del servidor incluye campos del objeto de usuario que no deberían estar ahí: número de cuenta, límite de crédito, historial de transacciones resumido.

Un SAST tradicional no ve esto. La función de permisos existe y está siendo llamada. Los patrones de SQL no muestran inyección. No hay `eval()`. No hay concatenación directa de inputs. Todo "parece" correcto en cada archivo por separado.

Claude Security lo detecta porque no analiza archivos. Analiza el camino que recorre el dato desde que entra como request HTTP hasta que sale como response JSON. En ese camino, identifica que el serializador tiene acceso a campos del objeto completo de usuario y que no hay ningún filtrado explícito antes de devolver la respuesta. El contexto de seguridad de la validación de permisos no se propaga al nivel de serialización. Eso es una vulnerabilidad de exposición de datos, y es exactamente el tipo de cosa que se explota en el mundo real sin que nadie haya escrito una línea de código maliciosa.

Lo que esto significa para el equipo de desarrollo

El impacto práctico de este cambio en el razonamiento va más allá de encontrar más bugs. Cambia la forma en que los equipos interactúan con los resultados del análisis.

Cuando una alerta de seguridad viene acompañada de una explicación del flujo de datos —"el input entra aquí, pasa por esta función sin ser filtrado, se combina con este otro parámetro en este archivo, y termina en esta query"— el desarrollador puede entender exactamente qué ocurre y por qué es un problema. No necesita interpretar un código de regla ni buscar en la documentación de la herramienta qué significa "CWE-89". El análisis es legible como razonamiento, no como catálogo.

Esto también reduce drásticamente los falsos positivos, porque el sistema no alerta sobre patrones que "parecen" peligrosos sino sobre flujos que realmente lo son. Y reduce los falsos negativos, porque las vulnerabilidades que no encajan en ningún patrón conocido —como la del ejemplo anterior— son visibles a través del análisis de flujo aunque nunca hayan aparecido en ninguna base de reglas.

Reflexión estratégica: el cambio no es técnico, es conceptual

El salto de los SAST tradicionales a un análisis contextual como el de Claude Security no es simplemente un upgrade de herramienta. Es un cambio en la pregunta fundamental que le hacemos a la seguridad del código.

Hemos pasado décadas preguntando "¿se parece esto a algo que salió mal antes?" Esa pregunta tiene utilidad. Pero las vulnerabilidades más interesantes —y las más explotadas en producción— no se parecen a nada que haya salido mal antes. Son combinaciones nuevas de decisiones razonables tomadas en momentos distintos por personas distintas que nunca tuvieron la posibilidad de ver el sistema completo.

La seguridad del código en 2026 necesita sistemas capaces de ver esa complejidad emergente. No porque los patrones ya no sirvan, sino porque los patrones solos ya no son suficientes.

Tu próximo paso

Si gestionas un equipo de desarrollo o eres responsable de la seguridad de una aplicación, la pregunta que merece la pena hacer no es "¿tenemos un SAST?". La pregunta es "¿nuestras herramientas de análisis entienden el comportamiento de nuestro código o solo su forma?".

La diferencia entre esas dos preguntas puede ser la misma que hay entre una auditoría que encuentra el problema y una auditoría que lo pasa por alto.

Si este artículo te ha resultado útil, compártelo con tu equipo técnico o con quien tome decisiones de seguridad en tu organización. Visita el blog para acceder a más recursos gratuitos sobre ciberseguridad aplicada.