kovra

WHITEPAPER · V1.0 · JUNIO 2026

Secretos en la era de los agentes de IA

el whitepaper de kovra

Un producto de Kaeus Inc

kovra.sh

> **Qué es esto.** Este es el **whitepaper de kovra** — un artículo de fundamentación de diseño, no un paper de investigación. No hace afirmaciones empíricas ni reporta experimentos; argumenta desde un problema real y observable y desde principios de seguridad establecidos hacia las decisiones específicas que toma kovra — y es franco respecto de lo que esas decisiones **no** garantizan. Cada afirmación de diseño que sigue se corresponde con un comportamiento que la herramienta efectivamente impone.

01 El problema

Un secreto solo sirve cuando algo lo usa. Por eso los secretos pasan su vida en movimiento: tecleados en terminales, exportados a shells, escritos en archivos `.env`, pegados entre un gestor de contraseñas y una configuración, copiados en una docena de líneas `export`. Cada paso deja un residuo — en el historial del shell, en los listados de procesos, en los logs, en un archivo que sobrevive a su propósito. La respuesta habitual de la industria es una checklist: no pegar eso ahí, rotar esto, limpiar esos logs. Las checklists pierden ante la conveniencia, de forma confiable, porque el camino inseguro es el fácil y el camino seguro es fricción.

Entonces aparece un nuevo actor en escena: el **agente de IA para programar**. Se lo apunta al repositorio para avanzar más rápido y, al hacerlo, se le otorga el mismo alcance que se tiene. Puede abrir cada `.env`, leer cada configuración, recorrer el historial del shell. Los secretos que estaban meramente dispersos ahora son legibles para un lector automatizado que actúa según lo que lee.

Este es el problema que kovra existe para abordar: una persona desarrolladora necesita que sus herramientas — y ahora sus agentes — usen secretos, mientras lo menos posible los vea en texto claro, y mientras el camino fácil sea también el seguro.

02 Las tensiones

El problema es difícil porque es un nudo de tensiones genuinas, no una sola funcionalidad faltante. Nombrarlas con honestidad es la única forma de razonar sobre una solución.

Uso versus exposición. Un secreto debe estar en texto claro en algún lugar en el momento de usarse — un driver de base de datos necesita la contraseña real. No se puede a la vez usar un valor y garantizar que nada lo vea nunca. La meta realista es reducir el conjunto de cosas que lo ven, y la duración — no llegar a cero.

Conveniencia versus control. Cada control que se agrega (un prompt, una allowlist, una confirmación) es fricción, y la fricción es precisamente lo que empuja a la gente de vuelta al

`.env` en texto plano. Un control demasiado pesado no se usa; una seguridad que no se usa no es seguridad.

Utilidad del agente versus contención del agente. Un agente es valioso porque puede correr los comandos y tocar los sistemas. La misma capacidad es el riesgo. Bloquearlo de todo y es inútil; dejarlo leer todo y es peligroso.

Un principal de confianza que puede ser manipulado. Los modelos de amenaza clásicos asumen un principal en quien se confía por defecto y que ocasionalmente traiciona. Un agente LLM es distinto en naturaleza: es **manipulable por el contenido que lee**. Un README envenenado, un mensaje de error fabricado, el docstring de una dependencia maliciosa pueden redirigirlo. El límite teórico de lo que podría filtrar es el mismo que para un humano; la frecuencia esperada de un intento es mayor, y el disparador puede ser datos, no intención.

03 Las implicaciones

Si se toman esas tensiones en serio, varias conclusiones se siguen antes de escribir una línea de código.

1. **Contención, no prevención.** Dado que un valor debe estar en texto claro en el punto de uso, el objetivo de diseño es reducir superficie y mantener los valores más peligrosos lejos de los lectores menos confiables — no prometer lo imposible.
2. **Por defecto seguro, y volver conveniente lo seguro.** Si el camino seguro es más difícil que pegar un secreto, el camino seguro pierde. La herramienta tiene que hacer que usar un secreto correctamente sea al menos tan fácil como usarlo con descuido — de lo contrario sus propios controles seleccionan por ser evadidos.
3. **Los metadatos no son texto plano.** Un agente puede ser enormemente útil sabiendo solo que un secreto existe, cómo se llama y cuán sensible es — sin nunca ver su valor. La unidad correcta para darle a un agente es metadatos más la capacidad de correr cosas, no el valor.
4. **El límite pertenece a un solo lugar.** Si cada interfaz reimplementa "qué está permitido", divergirán, y la implementación más débil se vuelve la política de facto. La regla tiene que vivir en un solo core que toda interfaz consume.
5. **Algún riesgo es del humano para aceptarlo, deliberadamente.** Hay momentos en que una persona genuinamente necesita un valor en pantalla. La respuesta no es prohibirlo sino hacerlo un acto deliberado, atendido, auditado — nunca un default, nunca algo que un agente pueda disparar por su cuenta.

04 La solución

El modelo de kovra es una respuesta directa a esas implicaciones. Su forma es: dejar que las cosas usen secretos sin verlos, y poner cada excepción detrás de un acto humano deliberado.

Usar, no ver

Las herramientas y los agentes obtienen valores mediante **inyección**: kovra resuelve un secreto y lo coloca directamente en el entorno de un proceso hijo, nunca en disco, en `argv` ni en el historial del shell. El proceso usa el valor; nada en el flujo de trabajo lo muestra. Un archivo `.env.refs` commiteable mapea nombres de variables a **coordenadas** — direcciones, no valores — de modo que el cableado se puede compartir mientras los secretos permanecen en el vault.

Metadatos para los agentes, texto plano retenido

Un agente se conecta sobre un servidor MCP bajo un **scope** — una capacidad que dice qué puede direccionar y hacer. Lee metadatos libremente e inyecta secretos en los comandos que corre, de modo que esos comandos funcionan — pero el texto plano de los secretos sensibles nunca aterriza en la ventana de contexto del modelo, que es el único lugar donde un ataque de prompt-injection podría exfiltrarlo.

La sensibilidad decide la entrega; el entorno agrega un piso

Cada secreto lleva un nivel de sensibilidad. `low` y `medium` fluyen directamente; `high` requiere una **confirmación biométrica atendida** antes de cualquier entrega; `inject-only` nunca se revela en absoluto. El entorno `prod` agrega un piso estructural por encima — un secreto `prod` nace `high`, y su texto plano puede llegar al contexto de un agente solo mediante un reveal iniciado por un humano y confirmado.

Mantener el ejecutor fuera del control del agente

Para el caso más peligroso — inyectar un secreto `high`/`prod` — kovra agrega una **allowlist de ejecutores**: el valor solo puede inyectarse en un ejecutable revisado y allowlisteado, no en un script ad-hoc que el agente acaba de escribir. Este es el quid. Un proceso que el agente escribió puede imprimir su propio entorno; la inyección por sí sola no contiene nada frente a un ejecutor que el agente controla. La contención viene de que el ejecutable esté fuera de ese control.

Un solo core, prompts autoritativos

La política vive en el **core**; la CLI, el wrapper, la web UI y el servidor MCP consumen sus decisiones y nunca las rederivan. Cuando se requiere una confirmación, el texto del prompt lo construye el core a partir de hechos observados — el comando resuelto, la coordenada, la sensibilidad — y nunca lo provee quien llama, de modo que un atacante no puede falsificar un prompt tranquilizador.

05 La criptografía

kovra usa deliberadamente un **conjunto pequeño de primitivas modernas y bien revisadas** del ecosistema de criptografía de Rust, de maneras estándar. Aquí no hay criptografía casera — el trabajo interesante y específico de kovra vive en la política, no en inventar cifrados. Cada elección se corresponde con un trabajo.

Primitiva	Dónde se usa	Por qué esta
ChaCha20-Poly1305	Cifrado en reposo (cada entrada del vault)	Autenticada, de tiempo constante en software
Argon2id	Derivar una clave desde una passphrase	Memory-hard contra fuerza bruta
BLAKE3	Fingerprints de secretos	Rápida, moderna; almacenada truncada (I12)
ed25519 (RSA por compatibilidad)	Credenciales keypair, firma, sellado	Pequeña, rápida, difícil de usar mal
age (X25519 + ChaCha20-Poly1305)	Cifrado de keypairs, backup de la clave maestra	Basada en recipientes, auditada, sin perillas
secrecy / zeroize	Manejo en memoria	Reduce la ventana de texto plano

Cifrado en reposo — ChaCha20-Poly1305

Cada entrada del vault se sella con el AEAD **ChaCha20-Poly1305**. Un AEAD da confidencialidad e integridad en un solo paso: un ciphertext manipulado falla al autenticarse en vez de descifrarse a basura plausible. Lo elegimos por sobre AES-GCM porque es **de tiempo constante en software puro** — no depende de aceleración por hardware de AES para evitar canales laterales de cache-timing — de modo que se comporta de forma idéntica y segura en cualquier máquina donde corra kovra.

Derivación de clave desde una passphrase — Argon2id

Cuando un vault se protege con una passphrase en vez del keychain del SO, la clave de cifrado se deriva con **Argon2id** — el estándar actual de hashing de contraseñas. Es **memory-hard**, lo que vuelve costosa la fuerza bruta por GPU y ASIC, y la variante `id` resiste tanto ataques de canal lateral como de time-memory-tradeoff. Una passphrase humana es de baja entropía; un KDF memory-hard es lo que la vuelve segura para usar como clave en absoluto.

Identidad y fingerprints — BLAKE3

Los secretos se fingerprintean con **BLAKE3**, dando una identidad estable y resistente a colisiones para un valor sin revelarlo. kovra solo almacena y muestra un fingerprint **truncado** — nunca uno lo bastante largo como para permitir que alguien confirme un valor adivinado emparejando su hash (I12). El truncado es una medida deliberada anti-fuerza-bruta, no un atajo.

Claves asimétricas — ed25519 (RSA por compatibilidad)

Las credenciales keypair usan por defecto **ed25519** (EdDSA): claves pequeñas, firmas deterministas rápidas, y ningún parámetro que equivocar. **RSA** se soporta pero acotado a firma/verificación y compatibilidad SSH — nunca cifrado asimétrico, porque el cifrado RSA invita a footguns de padding-oracle. Las claves se generan y almacenan en el **formato OpenSSH** (vía `ssh-key`), de modo que interoperan limpiamente con el ssh-agent y el tooling estándar. El cifrado asimétrico es solo ed25519.

Backup de claves y cifrado de keypairs — age

El cifrado de keypairs y el backup cifrado de la clave maestra usan ambos **age** (ChaCha20-Poly1305, ASCII-armored). age es un formato pequeño, auditado y opinado con **ninguna perilla de configuración que usar mal**. El cifrado de keypairs es **basado en recipientes** — sellado hacia quién puede abrirlo (su clave pública), que es exactamente la propiedad que kovra quiere: autorización anclada a la identidad, no a la posesión de un archivo (I8). El export de la clave maestra usa el modo passphrase (scrypt) de age, de modo que un backup puede recuperarse con cualquier implementación de age en un desastre.

Higiene de memoria — secrecy y zeroize

No son algoritmos, sino parte de la misma disciplina: los valores que portan secretos se envuelven para que nunca aterricen en logs ni en salida de debug, y su memoria se **zeroiza** al liberarse — reduciendo la ventana en la que un texto plano vive en la memoria del proceso. No cambia el límite de la última milla, pero lo estrecha.

06 Los riesgos

Una herramienta de seguridad introduce sus propios riesgos; pretender lo contrario sería lo opuesto a la intención de este artículo.

La clave maestra es una única raíz de confianza. Una clave por vault cifra todo. Si se pierde, el vault es irrecuperable; si se filtra, el cifrado en reposo queda en nada. kovra la custodia en el keychain del SO y ofrece un backup cifrado y protegido por passphrase — pero la concentración de confianza es real, y la higiene de claves es ahora el hábito propio más importante.

La herramienta es parte de la cadena de suministro. kovra corre en la máquina con acceso a los secretos. Un compromiso del binario, de sus dependencias o de su build es un compromiso de todo lo que protege. Esto es inherente a cualquier gestor de secretos y es la razón de una superficie de dependencias pequeña y una postura conservadora — no un riesgo que desaparezca.

Fatiga de confirmación. Los prompts son un control solo mientras se leen. Preguntar demasiado seguido y la gente aprueba por reflejo, razón por la cual kovra hace gate sobre sensibilidad en vez de preguntar por todo — pero un vault mal escalonado todavía puede entrenar a hacer clic en "approve" sin mirar.

Un prompt convincente sigue siendo una decisión humana. Un texto de prompt autoritativo eleva la vara contra prompts falsificados, pero el humano todavía puede aprobar una acción de apariencia legítima y genuinamente mala. La herramienta informa la decisión; no la toma.

07 Las limitaciones

Estas no son brechas a cerrar en una versión posterior. Son propiedades del problema, y nombrarlas es lo que mantiene honesto al resto del artículo.

La última milla es inevitable. En el instante de uso, el texto plano vive en la memoria de un proceso, y quien controla ese proceso puede leerlo. Ninguna herramienta puede entregar un valor a la aplicación mientras impide que la aplicación lo lea. Como todo gestor de secretos serio, kovra **no** intenta impedir que el principal autorizado lea el secreto. Invierte en cifrado, control de acceso, auditoría y reducción de superficie: mitigaciones de "asumir brecha", todas probabilísticas.

Para un secreto verdaderamente crítico, la contención vive en cómo se usa la herramienta. La protección robusta para un valor `prod` crítico es que el agente no controle el ejecutable que lo recibe — artefactos de despliegue revisados, no scripts ad-hoc del agente. El vault habilita esa disciplina; no puede imponerla por uno.

kovra gobierna el evento de autenticación, no la sesión que abre. Cuando kovra firma un desafío SSH o inyecta una contraseña de base de datos, gobierna ese momento. La sesión que se abre después queda fuera de su alcance; kovra no es un proxy de red ni un sandbox de runtime.

Un host comprometido está fuera de alcance. kovra defiende contra la dispersión de secretos y contra un agente que lee lo que no debería. No es una defensa contra malware con los privilegios propios, un keylogger a nivel kernel, o un atacante que ya es dueño de la máquina.

La amenaza del agente se reduce, no se elimina. Mantener el texto plano fuera del contexto del modelo cierra el camino de exfiltración por prompt-injection para los secretos sensibles. No

vuelve confiable a un agente, y no impide que un agente use mal un valor que se le permitió legítimamente usar.

08 En resumen

kovra no pretende resolver la gestión de secretos; ese problema tiene un piso probado y este artículo lo ha nombrado. Lo que hace es alinear el camino fácil y el camino seguro, reducir qué ve un secreto en texto claro, y poner al agente de IA del lado correcto de una línea metadatos-versus-texto-plano — con cada excepción convertida en un acto humano deliberado, atendido y auditado. Esa es una mejora significativa y honesta en un escenario donde el lector de los secretos es ahora automatizado y manipulable. No es, ni pretende ser, la abolición de la última milla.