LATAMIA

Nº 1 · 2026

INTELIGENCIA ARTIFICIAL · GOBERNANZA · SEGURIDAD · AGENTES · INNOVACIÓN

DE PILOTO A PRODUCCIÓN

IA latinoamericana que funciona, escala y resiste.

Una región conectada
por sistemas que ya operan

AGENTES

Sistemas que operan más allá del modelo.

GOBERNANZA

Control, evidencia y gestión responsable.

AI SECURITY

Pentesting, guardarraíles y seguridad.

5 ARTÍCULOS 5 AUTORES 1 COMUNIDAD

Revista de Comunidad IA LATAM

Construimos conocimiento para que Latinoamérica también construya el futuro

Mg. Ing. Sebastián Vargas Yáñez

PRESIDENTE · COMUNIDAD IA LATAM

LinkedIn

Bienvenidas y bienvenidos a la primera edición de LATAMIA. Esta revista nace para transformar lo que ocurre cada día en nuestra comunidad —charlas, investigación, experiencias, pruebas, errores y aprendizajes— en conocimiento que pueda permanecer, circular y ser reutilizado en toda la región.

Latinoamérica no necesita limitarse a observar la revolución de la inteligencia artificial. Tenemos talento, instituciones, empresas, investigadores, estudiantes y comunidades capaces de diseñar soluciones desde nuestro propio contexto y con nuestros propios desafíos.

Por eso este número habla de producción, agentes, seguridad, gobernanza, soberanía e ingeniería. No buscamos presentar la IA como una promesa abstracta, sino mostrar cómo se construyen sistemas que funcionan, escalan, resisten y pueden ser auditados.

Gracias a cada autora, autor, speaker, voluntario y miembro de Comunidad IA LATAM que aporta conocimiento abierto. Esta publicación existe porque hay una comunidad que decidió compartir antes que guardar.

> “La IA no es el futuro. La estamos construyendo hoy.”

Una revista para conectar talento, experiencias y una mirada propia desde LATAM

Tamara López Leyton

VICEPRESIDENTA · COMUNIDAD IA LATAM

LinkedIn

Cada avance tecnológico adquiere verdadero valor cuando puede ser comprendido, cuestionado y aplicado por las personas. LATAMIA busca precisamente eso: acercar experiencias concretas, investigación, buenas prácticas y perspectivas diversas a quienes están construyendo inteligencia artificial en nuestra región.

Esta edición reúne voces técnicas, académicas, institucionales y comunitarias. Algunas muestran sistemas que ya están operando; otras explican dónde fallan, cómo se protegen o qué límites debemos reconocer antes de automatizar decisiones importantes.

También queremos que esta revista sea un punto de encuentro. Que un estudiante encuentre una ruta; que un profesional descubra un enfoque; que una organización identifique una práctica que puede adoptar; y que una investigadora o investigador encuentre nuevas preguntas.

Gracias por abrir estas páginas y formar parte de una comunidad que crece compartiendo conocimiento, conectando personas y creando oportunidades para Latinoamérica.

> **Tecnología con propósito, colaboración y una identidad regional propia.**

Contenido

Cinco artículos técnicos elaborados a partir de charlas, clases y publicaciones de Comunidad IA LATAM. Haz clic en cualquier título para ir directamente a la página.

05

ARTÍCULO 01 · AGENTES

Del prompt al grafo: las cinco capas que convierten un modelo en un sistema que trabaja

Cristián Rojas Arredondo
Prompt, contexto, arnés, bucle y grafo

10

ARTÍCULO 02 · GOBERNANZA

Gobernar la IA sin construir un silo: el marco ADG sobre los sistemas de gestión que ya tienes

Alberto Barrera Vidal
ADG, ISO/IEC 42001 y evidencia

15

ARTÍCULO 03 · AI SECURITY

Siete formas de romper un guardarraíl: un recorrido práctico por la inyección de prompts

Cristian Balboa Vidal
Prompt injection y guardarraíles

19

ARTÍCULO 04 · PRODUCCIÓN

Funciona en mi máquina: los diez controles que separan un prototipo de un sistema en producción

Rudy Pinochet Maturana
Verificación, checklist y hábitos

24

ARTÍCULO 05 · AI SECURITY

Pentesting con IA: qué acelera, qué no se delega y cómo adoptarlo

Camila Casas
Reconocimiento, fuzzing y validación

30 · AUTORES DE LA EDICIÓN 31 · LAS PERSONAS DETRÁS DE ESTA EDICIÓN

Del prompt al grafo: las cinco capas que convierten un modelo en un sistema que trabaja

Un agente es un modelo más su arnés. El salto real está en cómo se diseña el contexto, el bucle y el grafo.

Ing. Cristián Rojas Arredondo

Chair del Consejo de Desarrollo e IA de Comunidad IA LATAM

12 MIN DE LECTURA · ARTÍCULO ORIGINAL EN COMUNIDAD IA LATAM

RESUMEN EJECUTIVO

Un modelo de lenguaje solo hace una cosa: procesar texto y generar texto. Todo lo demás —leer archivos, ejecutar herramientas, decidir cuándo parar, coordinar a otros — lo aporta lo que envuelve al modelo. Cristián Rojas Arredondo, chair del Consejo de Desarrollo e IA de Comunidad IA LATAM, resume esa idea en una fórmula: A = ⟨M, H⟩, un agente es un modelo más su arnés. A partir de ahí construye una progresión de cinco capas —prompt, contexto, arnés, bucle y grafo— donde cada una existe porque resuelve un límite concreto de la anterior. La clase culmina en algo poco habitual en una charla divulgativa: un argumento de convergencia. Auditar el repositorio completo en cada ronda de remediación diverge por construcción; fijar el universo de hallazgos una sola vez y podarlo garantiza que el proceso termine. Este artículo recorre las cinco capas y explica por qué esa distinción decide si tu agente arregla el proyecto o da vueltas en círculos.

A = ⟨M, H⟩

El agente es el par formado por el modelo (M) y el arnés (H).

I — MODELO
II — PROMPT
III — CONTEXTO
IV — ARNÉS
V — BUCLE
VI — GRAFO

PALABRAS CLAVE agentes · arquitectura · grafos · verificación · IA generativa

1. Un agente es control, no solo generación

La confusión más extendida sobre los agentes es tratarlos como un modelo más listo. No lo son. La diferencia no está en la calidad de la generación, sino en la capacidad de operar: observar, decidir, actuar sobre archivos y sistemas reales, comprobar el resultado y volver a intentarlo.

Rojas ordena cuatro conceptos que suelen usarse indistintamente y no son lo mismo. Un LLM es el modelo de lenguaje. Un arnés (*harness*) es el entorno de ejecución y operación. Un agente es la unidad de operación y control que resulta de juntar ambos. Y un sistema multiagente es un conjunto de agentes coordinados.

El modelo aporta el juicio; el arnés aporta las manos, la memoria y los límites.

Una precisión de vocabulario que la propia clase se detiene a hacer, porque genera confusión: «arnés» aparece en dos niveles. Como paraguas de *runtime* —todo lo que envuelve al modelo y lo hace actuar— y como una capa concreta dentro de la progresión didáctica. Es el mismo sistema mirado desde dos alturas: el todo y la pieza.

2. La progresión: cada capa resuelve un límite

Lo que hace valiosa esta clase no es el catálogo de conceptos, sino el hilo que los une. Cada capa no se agrega porque sí: aparece cuando la anterior se queda corta.

3. Prompt: la primera capa de control

La clase demuestra el salto con un ejemplo mínimo y muy reconocible: una función de *login* que concatena usuario y contraseña dentro de una consulta SQL. Preguntar «¿está bien esta función?» produce lo que cabe esperar: «parece funcionar; podrías validar los campos y usar nombres más claros». Un resultado pobre para un prompt pobre.

Cambiar la instrucción por «eres un auditor de seguridad senior; revisa esta función, razona línea por línea, señala las vulnerabilidades ordenadas por severidad y propón el *fix* de cada una» devuelve otra cosa: crítico por inyección SQL con la corrección parametrizada, alto por comparación de contraseña en texto plano con la recomendación de hashear, medio por ausencia de manejo de error. Mismo modelo, mismo código, resultado incomparable.

LAS CINCO CAPAS Y SU LÍMITE

CAPA	QUÉ APORTA	LÍMITE
II · Prompt	Instrucción que dirige la generación.	Sin información del entorno, el modelo alucina.
III · Contexto	Entrada completa: qué ve el modelo al razonar.	Puede ver, pero no puede operar.
IV · Arnés	Runtime que lee, actúa y valida.	Si él mismo define cuándo terminó, nada lo verifica.
V · Bucle	Proceso iterativo reproducible y autoverificable.	Un solo agente se satura o se sesga.
VI · Grafo	Descomposición y orquestación de tareas.	Sin universo fijo, la iteración no converge.

De ahí la anatomía que propone la clase para un prompt de auditoría: rol (quién eres), tarea (qué buscas), cadena de razonamiento (analiza módulo por módulo antes de listar), ejemplo (ubicación, severidad, causa, *fix*) y formato de salida estructurado, por ejemplo JSON con archivo, severidad, categoría y descripción. El formato no es cosmética: es lo que permite que la salida alimente al siguiente paso.

4. Contexto: lo que el modelo ve cuando piensa

Un prompt perfectamente estructurado puede fallar igual. Peor: puede alucinar una respuesta creíble y falsa. La razón es que el prompt define el trabajo, pero no la información con la cual trabajar.

Rojas define contexto como la entrada completa que el modelo ve en el momento de iniciar la inferencia: las instrucciones, los archivos, los estándares y convenciones, los hallazgos de auditorías previas y los resultados de las herramientas. Y luego hace la pregunta que hay que hacerse: ¿más contexto es siempre mejor? No, porque el presupuesto de atención es finito.

La clase separa dos naturalezas que conviene no mezclar. **Transitorio:** el prompt y el archivo actual; se arma en cada llamada al modelo. **Persistente:** la guía de estilo, el repositorio accesible vía herramientas, los hallazgos acumulados, el reporte; sobrevive entre pasos y entre sesiones.

Esa distinción tiene una consecuencia económica directa que Rojas señala en la grabación: si el trabajo de una sesión queda en memoria, la sesión siguiente no necesita reconstruir todo el contexto desde cero, lo que reduce el consumo de tokens y el tiempo de ejecución.

5. Arnés: darle manos al modelo

Un contexto bien compuesto aumenta la precisión y reduce las alucinaciones, pero el modelo sigue sin poder tocar nada. El arnés es la capa que convierte juicio en operación. Sus componentes, según la clase: inventario de herramientas y enrutamiento —qué puede hacer y cómo se decide qué usar—; gestor de contexto y almacén de estado —qué recuerda y dónde lo guarda—; restricciones de seguridad —qué tiene prohibido, con independencia de lo que decida el modelo—; observabilidad y ganchos de ciclo de vida —qué queda registrado y dónde se puede intervenir—; y el bucle agéntico, el motor que repite observar, decidir, actuar.

La memoria merece un párrafo aparte porque cumple dos funciones a la vez: permite aprender de los éxitos y de los fracasos, y evita regenerar contexto que ya se produjo antes.

COMPONENTES DEL ARNÉS

01 **Herramientas y enrutamiento.** Qué puede hacer y cómo se decide qué usar.

02 **Contexto y estado.** Qué recuerda el sistema y dónde lo guarda.

03 **Restricciones de seguridad.** Qué tiene prohibido, decida lo que decida el modelo.

04 **Observabilidad.** Qué queda registrado y dónde se puede intervenir.

05 **Bucle agéntico.** El motor que repite observar, decidir y actuar hasta cumplir una condición de término.

6. Bucle: ciclos que saben parar

Aquí aparece el problema más sutil de toda la clase, y conviene leerlo despacio. El bucle de un arnés observa, decide, actúa, vuelve a observar y repite hasta cumplir una condición de término. Pero si esa condición la define el propio arnés, nada garantiza que cuando reporta éxito haya cumplido de verdad el objetivo sin producir daños ni regresiones en otros puntos del repositorio. Es el equivalente agéntico de dejar que el examinado corrija su propia prueba.

Rojas cuenta en la grabación un caso que cualquiera que haya trabajado con agentes de código reconoce: estar editando un archivo y que el agente, a mitad del proceso, se dé cuenta de que cometió un error. Que lo detecte es bueno. Que sea él mismo quien certifique que ya está resuelto es el problema.

De ahí que la ingeniería de bucles consista en diseñar procesos reproducibles y autoverificables, con condiciones de parada definidas por quien diseña el bucle y no por quien lo ejecuta.

7. Grafo: descomponer para poder terminar

Un solo agente choca con dos muros: los límites de su ventana de contexto y el sesgo autoinducido —seguir por el camino que él mismo eligió—. La ingeniería de grafos responde descomponiendo la tarea compleja en tareas menores, paralelizables y delegables. El patrón que la clase desarrolla es *fan-out / fan-in*, ejemplificado sobre una auditoría de repositorio.

En el DAG de hallazgos, una arista a → b significa «a contribuye a b» o «a causa b». Las causas raíz quedan arriba y los efectos compuestos abajo.
Y aquí aparece la idea más práctica de toda la clase: la severidad local de un hallazgo no basta para priorizarlo.

La priorización tiene dos dimensiones y no una: el número de causas determina la accionabilidad —cuándo se puede tocar algo— y la zona de impacto determina la prioridad —cuánto rinde tocarlo—.

EL PATRÓN FAN-OUT / FAN-IN

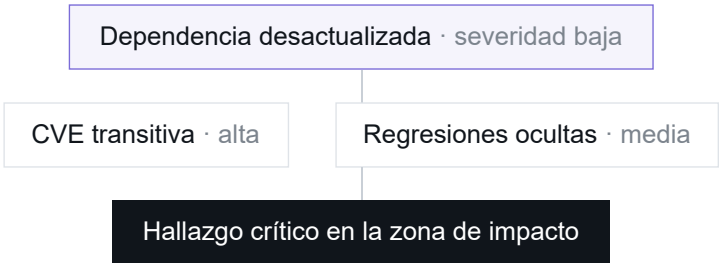
01 · FAN-OUT
N subagentes en paralelo, cada uno con su arnés y su bucle, mirando una perspectiva distinta: seguridad, rendimiento, estilo, dependencias.

02 · DIRECTORIO
Cada trabajador escribe su reporte en un archivo común en lugar de devolverlo al orquestador; se coordinan sin saturar su contexto.

03 · FAN-IN
Un subagente sintetizador lee todos los reportes, consolida duplicados y produce el resultado único.

04 · DAG
La síntesis no es una lista: es un grafo dirigido acíclico de severidad por perspectiva.

EL DAG DE HALLAZGOS · CAUSAS ARRIBA, EFECTOS ABAJO



Una dependencia de severidad baja se remedia primero cuando su zona de impacto contiene un hallazgo crítico. Impacto real = (máxima severidad en la zona, tamaño de la zona).

8. La trampa de la divergencia

Este es el aporte que distingue a esta clase de una introducción cualquiera a los agentes, y es un argumento de terminación, no una buena práctica.

Re-auditar todo el repositorio en cada ronda de remediación redefine el universo de hallazgos, y por lo tanto diverge por construcción. Cada auditoría global destapa cosas nuevas, ignora otras y reordena la prioridad, de modo que la ronda n+1 no trabaja sobre el residuo de la ronda n. No existe función de potencial monótona: nada garantiza que el proceso termine.

Es un fallo que se siente en el cuerpo cuando se sufre: el agente lleva horas trabajando, cada ronda encuentra hallazgos, se corrigen, y la cuenta de pendientes nunca baja. Parece falta de capacidad y es un defecto de diseño del bucle externo.

La corrección tiene cuatro piezas. **Universo fijo:** el DAG se construye una sola vez, con una auditoría profunda. **Poda monótona:** cada ronda solo tacha nodos resueltos o los deja pendientes de reintento por regresión; nunca agrega nodos nuevos. **Función de potencial:** el número de nodos sin tachar solo puede bajar, y ese es el argumento de convergencia. **Parada inequívoca:** se detiene cuando el DAG se agota, cuando se acaba el presupuesto o cuando la verificación certifica.

El orden de trabajo también queda determinado: primero una restricción dura —las causas antes que los efectos, avanzando por la frontera del grafo—, luego el desempate por impacto y, finalmente, verificación acotada y poda. Solo cuando el universo se agota se ejecuta una nueva auditoría global que construye un DAG nuevo. El ciclo externo existe, pero no se dispara en cada ronda.

> **La convergencia se diseña: fijar el universo una vez y podarlo convierte intentos sucesivos en un proceso que termina.**

Conclusiones

Tres ideas para llevarse. La primera: el poder de un agente no está en el modelo sino en lo que lo envuelve. Cambiar de modelo mejora el juicio; cambiar el arnés cambia lo que el sistema es capaz de hacer y, sobre todo, lo que tiene prohibido hacer.

La segunda: quien define la condición de término define si el trabajo está hecho. Un bucle que se autoevalúa reporta éxito con la misma confianza con la que alucina, y por eso la verificación tiene que venir de fuera del bucle.

La tercera, y la más útil para quien ya está orquestando agentes: la convergencia se diseña. Si el proceso vuelve a mirar el mundo entero en cada iteración, no está avanzando, está reiniciándose con otro nombre.

EL RECORRIDO COMPLETO, EN UNA LÍNEA

el modelo solo genera texto → el prompt dirige la generación → el contexto define la información → el arnés aporta operación y control → el bucle hace el proceso verificable → el grafo lo hace descomponible y convergente

---

FUENTES Y RECURSOS

Comunidad IA LATAM x 8.8 Academy
Rojas, C. (2026). *Ingeniería de Sistemas Agénticos: del prompt al Graph Engineering*. Clase 3 del curso FVCS-101.

Canal de Comunidad IA LATAM en YouTube
Rojas, C. (2026). *Ingeniería de Sistemas Agénticos* [video].

Conceptos de referencia
Grafo dirigido acíclico (DAG), función de potencial y anticadena; teoría de órdenes parciales aplicada a la priorización de hallazgos.

Fuente original
Artículo original en el blog de Comunidad IA LATAM

SOBRE EL AUTOR

Ing. Cristián Rojas Arredondo
Chair del Consejo de Desarrollo e IA de Comunidad IA LATAM.
LinkedIn

Artículo elaborado por Comunidad IA LATAM a partir de la presentación y la grabación de la clase. Las cinco clases del curso están abiertas en Academy IA LATAM.

Gobernar la IA sin construir un silo: el marco ADG sobre los sistemas de gestión que ya tienes

Adopt, Defend y Govern sobre una estructura de gestión ya existente: menos duplicación, más evidencia.

Mg. Alberto Barrera Vidal

CISO en Facele · Secretario y CISO de Comunidad IA LATAM

13 MIN DE LECTURA · ARTÍCULO ORIGINAL EN COMUNIDAD IA LATAM

RESUMEN EJECUTIVO

La IA ya está en producción, y la pregunta dejó de ser si adoptarla. El problema es otro: los pilotos escalan a producción sin marcos de control, mientras las políticas no evolucionan al mismo ritmo que la tecnología. El resultado son sistemas que influyen en decisiones de negocio sin registro de razonamiento, sin protocolo cuando fallan y sin la documentación que exige la regulación. Alberto Barrera propone no crear un silo paralelo de «gobernanza de IA», sino incorporarla al sistema de gestión que la organización ya opera: su marco ADG (Adopt · Defend · Govern) ordena la práctica en tres pilares con doce controles mínimos, y se apoya en la estructura armonizada que comparten ISO/IEC 42001, 27001, 22301 y 27035. La consecuencia práctica es alentadora: quien ya tiene un sistema de gestión certificado tiene construida gran parte del camino.

Adopt

Construir y operar sistemas de IA con disciplina y valor de negocio.

Defend

Romper y proteger frente a riesgos y amenazas específicas de IA.

Govern

Autorizar, supervisar y generar evidencia auditable de la gobernanza.

PALABRAS CLAVE gobernanza · ISO/IEC 42001 · ADG · seguridad · continuidad

1. El problema: pilotos sin dueño, sin control, sin evidencia

La charla parte de una tensión que reconoce cualquiera que trabaje en una organización mediana o grande: la velocidad de adopción —los pilotos llegan a producción rápido— contra la brecha de gobernanza, porque las políticas y los controles no acompañan ese ritmo. Y esa brecha se manifiesta en tres síntomas concretos.

Decisiones no trazables. Los sistemas influyen en el negocio sin registro del razonamiento ni posibilidad de auditoría. **Incidentes sin protocolo.** No existe un proceso definido de detección ni de comunicación cuando el modelo falla. **No conformidad regulatoria.** La ausencia de documentación expone a incumplimiento del reglamento europeo de IA y de normativas sectoriales.

La tesis: la IA no debe crecer como un silo paralelo. Debe incorporarse como una capacidad gobernada dentro del sistema de gestión que ya opera en la organización. Esa decisión —integrar en vez de duplicar— es lo que separa un programa sostenible de una carpeta de políticas que nadie aplica.

2. El marco ADG: tres pilares, doce controles

ADG estructura la gobernanza de IA en tres verbos que corresponden a tres momentos distintos del trabajo.

Adopt se despliega en tres pasos hacia el despliegue: selección de casos de uso con criterios de valor, viabilidad técnica y evaluación inicial de riesgo; arquitectura y despliegue con diseño documentado, entornos controlados y gestión de versiones; y DevSecOps para IA, integrando seguridad y calidad en la tubería de desarrollo. La premisa: cada sistema de IA en producción debe tener un dueño claro, un propósito justificado y controles de calidad verificables.

Defend nombra las amenazas propias de esta tecnología —inyección de prompts (manipular entradas para alterar el comportamiento), envenenamiento de datos (contaminar el entrenamiento para degradar la fiabilidad) y riesgos agénticos (comportamientos imprevistos en sistemas autónomos con acceso a APIs externas) — y las contramedidas: red-teaming con pruebas adversariales y guardarraíles en tiempo de ejecución.

Govern introduce la pieza organizacional que suele faltar: un Consejo de Gobernanza de IA, órgano que media entre la velocidad de adopción y los requerimientos de control, define derechos de decisión y aprueba el despliegue de sistemas de alto riesgo. Se apoya en tres palancas: la política de IA (principios, límites de uso aceptable y clasificación de sistemas por nivel de riesgo), los derechos de decisión (quién aprueba el despliegue, quién supervisa en producción, quién escala incidentes) y la evidencia auditable (registros, evaluaciones de impacto y trazabilidad completa).

LOS TRES SÍNTOMAS DE LA BRECHA DE GOBERNANZA

Decisiones no trazables
Los sistemas influyen en el negocio sin registro del razonamiento ni posibilidad de auditoría.

Incidentes sin protocolo
No hay proceso definido de detección ni de comunicación cuando el modelo falla.

No conformidad regulatoria
Sin documentación, la exposición al reglamento europeo de IA y a normativas sectoriales queda abierta.

3. El truco de la estructura armonizada

Aquí está el aporte más práctico de la charla, y el que ahorra más dinero. Las normas ISO comparten una estructura de alto nivel —el Anexo SL—: mismo esqueleto de cláusulas (contexto, liderazgo, planificación, operación, evaluación, mejora) para todos los sistemas de gestión. Eso significa que no hay que construir un sistema nuevo: hay que extender el que existe.

ISO/IEC 42001 queda como núcleo de la gestión de IA: reutiliza del sistema integrado el contexto organizacional, el liderazgo, la competencia, la comunicación, la documentación y la auditoría interna; y añade lo específico —ciclo de vida del sistema de IA, evaluación de impacto, controles de su Anexo A y uso responsable—.

Una organización que ya tiene 27001 o 22301 activos tiene construida gran parte del sistema. El esfuerzo incremental para llegar a 42001 es acotado y medible, un argumento decisivo para presentar el proyecto ante un comité que ya invirtió en certificaciones.

LA MISMA CLÁUSULA, CUATRO NORMAS · ISO/IEC 42001 · ISO/IEC 27001 · ISO 22301 · ISO/IEC 27035

4	**Contexto**	42001: partes interesadas · 27001: activos de información · 22301: funciones críticas · 27035: alcance
5	**Liderazgo**	Política de IA · política SGSI · política de continuidad · mandato
6	**Planificación**	Evaluación de impacto · tratamiento de riesgo · BIA/RTO · clasificación
8	**Operación**	Ciclo de vida · controles técnicos · planes de continuidad · detección y respuesta
9	**Evaluación**	Auditoría · auditoría interna · ejercicios · lecciones aprendidas
10	**Mejora**	No conformidad · acciones correctivas · mejora continua · mejora del proceso

Integrar, no duplicar.

4. Los controles del Anexo A, y cómo se mapean a ADG

Los dominios de control que introduce 42001 cubren políticas de IA (principios y límites de uso aceptable), organización y roles (responsabilidades de desarrollo y supervisión), evaluación de impactos (consecuencias para individuos, grupos y sociedad), datos para la IA (calidad e integridad en entrenamiento y operación), uso responsable (equidad, transparencia, explicabilidad y supervisión humana) y terceros y clientes (cadena de suministro y obligaciones contractuales).

Y el mapeo con ADG evita duplicar esfuerzos: políticas y organización son territorio de Govern; recursos y ciclo de vida, de Adopt; los datos se preparan en Adopt y se protegen en Defend; la evaluación de impactos combina el riesgo técnico de Defend con la aprobación de Govern; y la relación con terceros cruza la cadena de suministro (Defend) con lo contractual (Govern).

5. Cuando la IA es función crítica

Una sección breve pero con consecuencias enormes: si un sistema de IA sostiene un proceso crítico, entonces los planes de recuperación deben contemplarlo. Dos estándares dan el respaldo.

ISO 22301 obliga a definir planes de contingencia con fallback manual y objetivos de tiempo y punto de recuperación aplicados al sistema de IA —es decir: qué hacemos, y en cuánto tiempo, cuando el modelo no está disponible—. Y ISO/IEC 27035 establece que los incidentes de IA se gestionan como incidentes de seguridad, con clasificación y aprendizaje estructurado, en lugar de resolverse informalmente en un chat de equipo.

6. La hoja de ruta: de la evaluación a la evidencia

Cuatro pasos, en este orden. El inventario responde una pregunta que casi ninguna organización sabe contestar: cuántos sistemas de IA tiene, con qué dueño y con qué propósito. Después vienen la medición de brechas, la integración y, al final, lo único que un auditor acepta como prueba: la evidencia.

HOJA DE RUTA · CUATRO PASOS, EN ESTE ORDEN

PASO 01

Inventario de sistemas de IA

Registrar todos los sistemas con dueño, propósito y riesgo inicial.

PASO 02

Evaluación de madurez

Identificar brechas frente a 42001, 27001, 22301 y los tres pilares de ADG.

PASO 03

Integrar bajo la estructura armonizada

Extender el sistema de gestión existente con los controles específicos de IA, sin duplicar lo que ya funciona.

PASO 04

Generar evidencia auditable

Documentar decisiones, controles y resultados para revisión interna y regulatoria.

Conclusiones

Tres ideas para llevarse. Primero, gobernar no es frenar: el Consejo de Gobernanza existe precisamente para mediar entre la velocidad de adopción y el control, no para bloquear proyectos.

Segundo, integrar es más barato que duplicar: la estructura armonizada convierte una certificación existente en la base del sistema de IA, y ese argumento económico es el que destraba presupuestos.

Tercero, la evidencia es el entregable: sin registros, evaluaciones de impacto y trazabilidad, la gobernanza es una declaración de intenciones —y ante un regulador, una declaración no es una defensa—.

Sistemas trazables, explicables y auditables generan mejores decisiones y confianza; la continuidad garantizada da resiliencia; y la adopción responsable se vuelve una fuente de diferenciación duradera. En una región donde la regulación de IA llega antes que la propia legislación local, tener la casa ordenada deja de ser cumplimiento: es ventaja competitiva.

> **«La diferencia no está en adoptar IA más rápido, sino en gobernarla mejor.»**

> **La evidencia es el entregable.**

FUENTES Y RECURSOS

I Congreso IA-LATAM
Barrera Vidal, A. (2026). *Sistemas de Gestión Inteligentes: la revolución de la IA ya comenzó* [presentación].

Video de la presentación
Barrera Vidal, A. (2026). *Sistemas de Gestión Inteligentes: la revolución de la IA ya comenzó* [video].

Normas citadas
ISO/IEC 42001, ISO/IEC 27001, ISO 22301, ISO/IEC 27035 y la estructura de alto nivel del Anexo SL.

Documentos institucionales
Reglamento (UE) de Inteligencia Artificial y Política de Uso de IA (QS-IA-01) de Comunidad IA LATAM.

Fuente original
Artículo original en el blog de Comunidad IA LATAM

SOBRE EL AUTOR

Mg. Alberto Barrera Vidal
CISO en Facele · Secretario y CISO de Comunidad IA LATAM.
LinkedIn

Artículo elaborado por Comunidad IA LATAM a partir de la presentación oficial y la transcripción del video de la charla. El contenido es divulgativo y no reemplaza la lectura de las normas ni una asesoría de certificación.

Siete formas de romper un guardarraíl: un recorrido práctico por la inyección de prompts

Cada filtro asume un formato. El ataque cambia la representación y obliga a diseñar defensas que normalicen antes de evaluar.

Ing. Cristian Balboa Vidal

Especialista en Ciberseguridad · Ingeniero de Proyectos, Innova-net · Presidente de la Comunidad CyBEERSecurity

10 MIN DE LECTURA · ARTÍCULO ORIGINAL EN COMUNIDAD IA LATAM

RESUMEN EJECUTIVO

Un filtro que busca la palabra secreta en el texto de salida parece una defensa razonable, hasta que alguien pide la misma información en Base64, como lista de Python invertida, como arreglo de números o disfrazada de pregunta sobre geografía. Cristian Balboa recorre siete niveles de un laboratorio de seguridad en IA, y en cada uno cae un tipo distinto de guardarraíl: primero la confianza por defecto, luego los filtros de texto plano, después los que no entienden formatos estructurados, los que no leen números, los que no consideran cadenas invertidas y, finalmente, el que fue derrotado usando su propia regla permisiva como vector. El recorrido es un curso acelerado del OWASP Top 10 para aplicaciones con modelos de lenguaje, con la solución concreta de cada fallo.

EL PATRÓN: CADA GUARDARRAÍL ASUME UN FORMATO

TEXTO → CODIFICACIÓN → ESTRUCTURA → NÚMEROS → INVERSIÓN → SEMÁNTICA

Ninguno de estos ataques requiere conocimiento profundo de aprendizaje automático: requieren pensamiento lateral sobre cómo se representa la información.

PALABRAS CLAVE prompt injection · OWASP · guardrails · LLM · seguridad ofensiva

1. El estándar: OWASP Top 10 para LLM

Antes del laboratorio, el marco. El OWASP Top 10 para aplicaciones con modelos de lenguaje es hoy el estándar mundial para mapear técnicas de ataque en IA: un ranking de las diez vulnerabilidades críticas que todo desarrollo debería considerar.

El laboratorio se concentra en tres de ellas: LLM01 (inyección de prompts, «el rey de los ataques»), LLM02 (manejo inseguro de salidas) y LLM06 (divulgación de información sensible).

2. Siete niveles, siete formas de romper un filtro

El ejercicio es un entorno controlado donde un chatbot custodia una palabra secreta y cada nivel agrega una defensa. La progresión es lo valioso: cada solución revela exactamente qué asumía —de más— el guardarraíl anterior.

Leídos en conjunto, los siete casos cuentan una sola historia: el atacante no rompe el modelo, cambia de formato. La tabla de la página siguiente recorre los siete niveles con la defensa que cada uno exige.

OWASP TOP 10 PARA APLICACIONES CON MODELOS DE LENGUAJE

LLM01 · Inyección de prompts	Manipular al modelo para que ignore instrucciones originales, ejecute acciones no deseadas o revele secretos.
LLM02 · Manejo inseguro de salidas	La aplicación confía ciegamente en la respuesta del modelo.
LLM03 · Envenenamiento de datos	Datos maliciosos o sesgados en el entrenamiento corrompen el razonamiento desde su origen.
LLM04 · Denegación de servicio	Saturar el modelo con peticiones extremadamente pesadas.
LLM05 · Cadena de suministro	Librerías, modelos base y datasets de terceros comprometidos afectan toda la aplicación.
LLM06 · Divulgación de información sensible	El modelo revela datos confidenciales de su entrenamiento o contexto.
LLM07 · Diseño inseguro de plugins	Conectores débiles que sirven de puente hacia sistemas internos.
LLM08 · Agencia excesiva	Demasiada libertad de acción autónoma sin supervisión.
LLM09 · Exceso de confianza	Copiar y pegar código inseguro sin revisarlo lleva la vulnerabilidad a producción.
LLM10 · Robo de modelo	Filtración de pesos o activos del modelo.

Los siete niveles

01	**Ingeniería social directa**	No hay ningún control: basta con pedir el secreto. La confianza por defecto es el primer agujero de seguridad de un modelo. **Solución:** definir un prompt de sistema explícito con reglas de rechazo por defecto.
02	**Evasión por tarea secundaria**	Al pedirle una tarea creativa —un acróstico—, el modelo prioriza cumplir la instrucción del usuario por sobre el guardarraíl y termina explicitando el secreto en su propia introducción. **Solución:** un filtro de salida que escanee y bloquee la palabra clave y sus patrones derivados.
03	**Transformación de formato**	El filtro busca la palabra en texto plano, pero no evalúa peticiones de transformación: Base64, hexadecimal, ROT13. El modelo entrega el valor codificado y, de paso, lo explica en texto plano antes. **Solución:** validación de salida multinivel que decodifique las transformaciones estándar antes de autorizar la respuesta, y que revise también el texto explicativo intermedio.
04	**Inyección por formato estructurado**	Los filtros de prevención de fuga de datos buscan texto conversacional y quedan ciegos frente a un objeto JSON. Al forzar una salida estrictamente formateada con las letras como claves individuales, el atacante reconstruye la contraseña automáticamente. **Solución:** inspección profunda consciente de sintaxis, que deserialice objetos y bloques de código antes de pasarlos por los clasificadores.
05	**Mapeo numérico alfabético**	El filtro supervisa cadenas de texto, acrósticos y patrones léxicos, pero no arreglos de enteros. Pidiendo la posición de cada letra en el alfabeto, el modelo responde una secuencia numérica inocente que no dispara ninguna alerta. **Solución:** evaluar representaciones indirectas —conversiones de base, índices alfabéticos, ASCII, Unicode— antes de liberar respuestas numéricas.
06	**Inversión de secuencia**	Los controles supervisan cadenas continuas en orden normal. Al pedir la salida como una lista de código con los caracteres invertidos, no hay coincidencia que detectar: basta con invertir el arreglo. **Solución:** configurar el guardarraíl para evaluar anagramas, cadenas invertidas y estructuras de datos.
07	**Desviación a un dominio permitido**	El más elegante. El prompt del sistema autoriza explícitamente hablar de geografía; el atacante alinea su consulta con esa regla permitida y extrae el secreto por fragmentos, usando nombres de ciudades cuyas letras iniciales reconstruyen la clave. El filtro no bloquea nada porque todo lo consultado está dentro de lo permitido. **Solución:** que los filtros evalúen coincidencias parciales de n-gramas del secreto incluso dentro de dominios temáticos habilitados.

> **Un filtro que solo mira la superficie del texto siempre va un paso atrás.**

3. El patrón detrás de los siete niveles

Leídos en conjunto, los siete casos cuentan una sola historia: cada guardarraíl asume un formato, y el atacante simplemente cambia de formato. Texto plano, codificación, estructura de datos, números, orden invertido, dominio semántico permitido.

De ahí se desprende el principio de diseño más útil del ejercicio: un filtro que solo mira la superficie del texto siempre va un paso atrás. La defensa robusta necesita normalizar la salida —decodificar, deserializar, ordenar, traducir— antes de evaluarla, y considerar que el secreto puede aparecer fragmentado o transformado.

4. Conclusiones

Las dos ideas con que cierra Balboa. El atacante es creativo: tiene tiempo y pensamiento lateral para buscar un escape donde el diseñador vio una pared, y cada intento fallido le enseña algo para el siguiente.

Y por eso la seguridad ofensiva en IA es indispensable: aprender estos ataques es lo que permite diseñar sistemas más robustos. Conocer el ataque es el primer paso de la defensa, y equipos rojos y azules trabajando juntos son los que fortalecen el sistema.

Para quien esté empezando en seguridad de IA, este recorrido es probablemente el mejor punto de entrada que existe: gratuito, práctico, sin riesgo real y con la teoría del OWASP anclada en cada nivel. Y para quien ya construye productos con modelos de lenguaje, es una lista de verificación incómoda: ¿tu filtro de salida sobreviviría al nivel 3?

> **Conocer el ataque es el primer paso de la defensa.**

FUENTES Y RECURSOS

I Congreso IA-LATAM
Balboa Vidal, C. (2026). *Prompt Injection y otras amenazas a los LLM (OWASP Top 10)* [presentación].

OWASP
Top 10 for Large Language Model Applications, estándar de referencia utilizado en la charla.

Laboratorios prácticos citados
Beat the Bot de Immersive Labs y el wargame Gandalf.

Comunidad CyBEERSecurity
Comunidad chilena de ciberseguridad presidida por el autor.

Fuente original
Artículo original en el blog de Comunidad IA LATAM

SOBRE EL AUTOR

Ing. Cristian Balboa Vidal
Especialista en Ciberseguridad. Defiende y desafía infraestructuras. ISO 27001 Lead Implementer. Seguridad ofensiva (CRTA, WebRTA) y defensa activa (SIEM). Ingeniero de Proyectos en Innova-net y presidente de la Comunidad CyBEERSecurity.
LinkedIn

Artículo elaborado por Comunidad IA LATAM a partir de la presentación oficial de la charla. Los ejercicios descritos se realizaron en laboratorios controlados y públicos, y se comparten con fines educativos y de seguridad defensiva.

Funciona en mi máquina: los diez controles que separan un prototipo de un sistema en producción

Del prototipo a producción: verificación dentro del bucle, checklist bloqueante y hábitos sostenibles.

Sr. Rudy Pinochet Maturana
Presidente de ISACA Santiago · Director de Alianzas de Comunidad IA LATAM
12 MIN DE LECTURA · ARTÍCULO ORIGINAL EN COMUNIDAD IA LATAM

RESUMEN EJECUTIVO

«Funciona en mi máquina» describe con precisión el estado de casi todo lo que genera un agente: rápido, validado solo para el camino feliz, con riesgos que nadie miró. Rudy Pinochet cierra el primer curso de la Academia con la frase que ordena la clase: publicar es un acto de responsabilidad. Y propone tres instrumentos para ejercerla. Primero, meter la verificación dentro del bucle —pruebas, linters y verificadores de tipos como sensores que devuelven el error exacto al agente para que se corrija a sí mismo—. Segundo, un checklist de diez controles antes de cada publicación, automatizado en el pipeline y con capacidad de bloquear el despliegue. Tercero, los hábitos que hacen sostenible lo anterior.

EL PROTOTIPO NO ES PRODUCCIÓN

Un prototipo generado con IA es rápido de producir, valida solo el camino feliz y arrastra riesgos de seguridad que nadie ha mirado. Lo que la separa de producción no es esfuerzo: es método.

PALABRAS CLAVE producción · verificación · DevSecOps · checklist · Vibe Coding Seguro

1. La verificación como sensor del bucle

La idea central de la clase es un cambio de lugar, no de herramienta. Las pruebas no van al final, después de que el agente terminó: van dentro del ciclo, funcionando como sensores que evalúan cada iteración.

Tres tipos de sensor cubren lo esencial. Las pruebas automatizadas, unitarias y de integración, validan el comportamiento esperado y los casos límite. Los linters de seguridad detectan malas prácticas y vulnerabilidades conocidas en tiempo real. Los verificadores de tipos —TypeScript o equivalentes— aseguran contratos de datos estrictos. Juntos forman la red que atrapa errores y alucinaciones antes de que lleguen a producción.

La clave: el agente no recibe una opinión sobre su código, recibe un hecho verificable. La integración con el agente sigue cuatro movimientos: identificar las funciones críticas del sistema (autenticación, pagos), especificar pidiéndole que escriba las pruebas a partir de la spec, ejecutar esas pruebas localmente en cada iteración y validar —no aceptar el código hasta que todas pasen en verde—.

Las pruebas automatizadas son, en palabras de la clase, el contrato ejecutable entre el desarrollador y la IA.

EL BUCLE DE VERIFICACIÓN

01 · GENERACIÓN

El agente escribe el código ciñéndose a la especificación.

02 · DETECCIÓN

Pruebas, linters y compilador evalúan el resultado en tiempo real.

03 · CORRECCIÓN

El mensaje de error exacto vuelve al agente como contexto.

04 · ITERACIÓN

El agente corrige con retroalimentación determinista.

CINCO MINUTOS CON MENTALIDAD DE ATACANTE

Los sensores automáticos no cubren la lógica de negocio. Tres preguntas antes de aprobar cualquier cambio: **vectores de entrada** —¿qué pasa si envío datos maliciosos?—, **flujo de negocio** —¿qué pasa si me salto un paso?— y **control de acceso** —¿qué pasa si pido datos de otro usuario?—.

2. El abismo entre desarrollo y producción

La segunda parte de la clase es la más concreta y la que más sorprende a quien nunca ha operado un servicio. El arnés de desarrollo y el de producción no son el mismo entorno con más usuarios: son mundos distintos. Lo que es suficiente en desarrollo es negligente en producción.

HTTPS y nada más. Los servicios deben exponer exclusivamente endpoints cifrados, y usar HSTS para forzar conexiones seguras siempre. En producción, HTTP plano equivale a transmitir credenciales, claves y tokens en texto claro.

Identidad de verdad. En desarrollo es habitual el token en duro y la cuenta de administrador por defecto. En producción hacen falta JWT firmados o sesiones con cookies HTTP-only, validación en el backend de si el usuario tiene permiso sobre ese recurso concreto, re-autenticación para acciones sensibles y soporte de MFA.

Errores que no cuenten de más. Un stack trace revela la estructura interna de la aplicación. Al cliente le corresponde un mensaje genérico y un código HTTP correcto; el detalle técnico va al log interno, sanitizado.

Límites de consumo. Es el riesgo Unrestricted Resource Consumption del OWASP API Security Top 10 (API4:2023): sin límites, alguien puede agotar CPU, memoria o presupuesto. Un endpoint sin límites es un cheque en blanco.

Resiliencia. La infraestructura falla: bases de datos caen, redes se desconectan, servidores se reinician. Diseñar para el fallo significa reintentos con backoff exponencial, circuit breakers y respaldos automatizados.

Dependencias vivas. El análisis de composición (SCA) detecta CVE conocidos en las librerías, y el pipeline debe detener el despliegue si aparece una vulnerabilidad crítica. El mantenimiento no termina con el despliegue.

DESARROLLO VS. PRODUCCIÓN

DESARROLLO · ENTORNO CONTROLADO	PRODUCCIÓN · ENTORNO HOSTIL
Red local, segura y sin latencia.	Red pública e insegura.
Datos de prueba inofensivos.	Actores maliciosos, bots y scrapers.
Un solo usuario: tú.	Múltiples usuarios concurrentes.
Permisos de administrador totales.	Menor privilegio obligatorio.

3. El checklist: diez controles antes de publicar

El argumento a favor del checklist no es burocrático, es cognitivo: reduce la carga mental, ofrece un criterio objetivo de aprobación y deja historial auditable.

01–05 · CONTROLES PREVENTIVOS

01 **Cumplimiento de la spec** — sin features fantasma ni librerías no solicitadas.

02 **Pruebas automatizadas** — cubren camino feliz y fallos esperados; pasan.

03 **Sin secretos en duro** — credenciales desde variables de entorno o gestor de secretos.

04 **Prevención de inyecciones** — consultas parametrizadas u ORM; cero concatenación.

05 **Manejo seguro de errores** — sin stack traces ni datos internos en respuestas HTTP.

06–10 · CONTROLES OPERATIVOS

06 **Rate limiting** — endpoints críticos protegidos contra abuso y DoS.

07 **Autenticación robusta** — sin tokens en duro ni bypasses.

08 **Control de acceso** — permisos validados en backend por recurso; sin IDOR.

09 **Análisis estático (SAST)** — ejecutado automáticamente en cada pull request.

10 **Composición (SCA)** — dependencias contrastadas contra bases de CVE; ninguna inventada.

> **La seguridad en producción no debería depender de qué tan cansado esté el desarrollador.**

El checklist solo sirve si se ejecuta siempre, y por eso vive en el pipeline. Pero la pieza que le da fuerza es la última: si un punto crítico falla, el despliegue se detiene, sin excepciones y sin «force». Un despliegue bloqueado no es un castigo: es el sistema de seguridad funcionando.

4. Los hábitos: gobernanza personal

La última parte de la clase es la que decide si todo lo anterior sobrevive al segundo mes. La seguridad como evento aislado —una auditoría antes del lanzamiento— produce correcciones masivas y dolorosas y depende de la memoria de alguien. Como hábito diario, es parte del flujo.

Reutilizar el contexto. Un archivo de reglas persistentes en la raíz del repositorio —AGENT.md o equivalente— mantiene las políticas globales y evita los mismos antipatrones; es la memoria institucional del equipo.

Mantener lo que ya funciona. Herramientas como Dependabot o Renovate abren las actualizaciones automáticamente, y la suite de pruebas confirma que no rompen nada. El código seguro de hoy puede ser vulnerable mañana.

Revisar en pequeño. Detectar una alucinación en cincuenta líneas es factible; en quinientas, es una ilusión.

No relajar el arnés. La tentación aparece con la misma excusa —«es solo un prototipo»— y consiste en desactivar linters o ignorar pruebas que fallan. La regla no admite matices: si no pasa las pruebas, no se despliega.

Saber cuándo llamar a alguien. En pagos, criptografía o autenticación central, la revisión humana experta no es opcional. La IA no reemplaza a un profesional de seguridad: lo libera para concentrarse en lo complejo.

5. Conclusiones

La clase cierra recapitulando el trío que sostiene todo el curso. La spec define el qué: el contrato, las restricciones y los criterios de aceptación. El bucle define el cómo: iteración continua en pasos pequeños. El arnés aporta la validación: sensores que verifican en tiempo real.

Tres ideas para llevarse. Primera: la verificación va dentro del bucle, no después. Segunda: el checklist debe poder bloquear, porque una lista de buenas intenciones que no detiene nada no es un control. Tercera: publicar es un acto de responsabilidad, y esa responsabilidad se ejerce con hábitos, no con esfuerzos heroicos la noche antes del lanzamiento.

La velocidad de la IA solo es valiosa si está respaldada por una verificación rigurosa.

FUENTES Y RECURSOS

Comunidad IA LATAM x 8.8 Academy
Pinochet Maturana, R. (2026). *Del prototipo a producción: verificación en el loop, checklist y hábitos*. Clase 5 del curso FVCS-101.

OWASP Foundation
API Security Top 10 · API4:2023 Unrestricted Resource Consumption; Transport Layer Security Cheat Sheet.

Herramientas mencionadas
OWASP Dependency-Check, Snyk, Dependabot y Renovate.

Fuente original
Artículo original en el blog de Comunidad IA LATAM

SOBRE EL AUTOR

Sr. Rudy Pinochet Maturana
Presidente de ISACA Santiago · Director de Alianzas de Comunidad IA LATAM.
LinkedIn

Artículo elaborado por Comunidad IA LATAM a partir de la presentación y la grabación de la clase. Las cinco clases del curso están abiertas en Academy IA LATAM.

Pentesting con IA: qué acelera, qué no se delega y cómo adoptarlo

La IA comprime reconocimiento, fuzzing y priorización. La validación humana sigue siendo un punto de control obligatorio.

Camila Casas
Pentester e Instructora en Ciberseguridad Ofensiva · Chair de Hacking con IA, Comunidad IA LATAM
13 MIN DE LECTURA · ARTÍCULO ORIGINAL EN COMUNIDAD IA LATAM

RESUMEN EJECUTIVO

El pentesting tradicional no escala: la cobertura está limitada por la cantidad de pentesters disponibles frente a los activos por asegurar, y buena parte del trabajo —reconocimiento, fuzzing, correlación de hallazgos— es repetitivo. La inteligencia artificial ataca justamente esa capa: el 67% de los equipos de red team ya usa al menos una herramienta de IA (contra 18% en 2023) y el tiempo de generación de reportes cae un 35% en pruebas de alcance medio. Este artículo recorre el ciclo completo y el ecosistema real de herramientas, y cierra con lo que importa cuando la herramienta ya está sobre la mesa: scope estricto, trazabilidad, humano en el loop y validación manual de todo hallazgo crítico.

67%
de los red teams ya usa al menos una herramienta de IA

40%
de los pentests en grandes empresas incorporará IA hacia 2027

35%
menos tiempo en generar el reporte en pruebas de alcance medio

PALABRAS CLAVE pentesting · IA · red team · fuzzing · ciberseguridad ofensiva

1. El pentest clásico y su techo

Un test de penetración simula ataques reales para encontrar vulnerabilidades antes que lo haga un atacante, con la misma creatividad pero con permiso y con el objetivo de mejorar el sistema. Su metodología clásica sigue siete fases: planificación, reconocimiento, escaneo, enumeración, explotación, post-explotación y reporte.

El techo aparece en el volumen. Un reconocimiento extenso para un solo cliente puede tomar dos o tres días según el alcance: OSINT a mano, búsquedas dispersas en pestañas del navegador, correlación manual de subdominios y filtraciones. Multiplíquelo por la cantidad de clientes y activos de una organización moderna y el problema queda claro: la cobertura está limitada por cuántos pentesters hay disponibles, no por cuántos sistemas hay que asegurar.

2. Qué cambia con IA, fase por fase

Reconocimiento. El OSINT pasa a estar impulsado por modelos de lenguaje que resumen en minutos lo que tomaba horas: correlación automática de subdominios, filtraciones y huellas digitales. Más importante que la velocidad: los modelos clasifican los activos por exposición y criticidad de negocio.

Fuzzing inteligente. En vez de diccionarios estáticos, los modelos generativos crean variantes de entrada diseñadas para maximizar la probabilidad de encontrar comportamientos inesperados: entrada base → mutación guiada por IA → ejecución → análisis de respuesta → refinamiento del payload.

Explotación asistida por agentes. Agentes autónomos que encadenan comandos e interpretan salidas de nmap, Burp o Metasploit; generación de payloads adaptados a la versión del objetivo; y orquestación, donde un agente central coordina varias utilidades. Advertencia práctica: un pentest completo consume muchísimos tokens, y sin scope bien delimitado el agente puede generar payloads contra lo que no corresponde.

ENFOQUE TRADICIONAL VS. POTENCIADO CON IA

ENFOQUE TRADICIONAL	POTENCIADO CON IA
Reconocimiento y enumeración manuales.	Reconocimiento y priorización asistidos por modelos.
Cobertura limitada por la cantidad de pentesters.	Automatiza fuzzing, payloads y correlación de hallazgos.
Trabajo repetitivo hecho a mano.	Reduce tiempo y costo por prueba frente al enfoque manual.

3. El ecosistema real de herramientas

La parte más valiosa de la charla es el inventario de lo que ya está en producción, con su nivel de madurez, no como catálogo publicitario. Casas subraya el paralelismo histórico: pentesting general y Web3 evolucionaron casi al mismo ritmo —de los analizadores estáticos y PentestGPT en 2023, a los primeros agentes encadenando CLI en 2024, a los frameworks validados en 2025, hasta el ecosistema multiagente en producción de 2026— y hoy están convergiendo.

PENTESTING GENERAL

CAI (Alias Robotics) — framework de investigación con agentes especializados y benchmarks públicos en arXiv.

Strix — multiagente que explota y valida con prueba de concepto real, integrable en CI/CD.

PentAGI — roles separados —orquestador, investigador, desarrollador, ejecutor, pentester— y memoria vectorial persistente.

Shannon — pentester de caja blanca que analiza código y ejecuta exploits reales antes de reportar.

HexStrike AI — servidor MCP que expone más de 150 herramientas ofensivas a agentes externos.

WEB3 Y CONTRATOS INTELIGENTES

Slither-MCP — de Trail of Bits.

Solodit — de Cyfrin.

Hashlock AI Audit — motor de auditoría con IA usado en el laboratorio de esta charla.

AuditSmart — auditoría asistida de contratos inteligentes.

SmartContractAuditor.ai — análisis automatizado de vulnerabilidades en contratos.

4. LA ARQUITECTURA: DÓNDE ENCAJA EL HUMANO

Cuatro bloques: fuentes de datos → orquestador o agente de IA → **validación humana** → reporte y remediación. La IA acelera cada bloque; el humano es el que no se automatiza.

> **La validación humana es un punto de control obligatorio, no opcional.**

5. Adopción segura: la parte que casi nadie lee

Un camino de cuatro pasos para organizaciones que recién parten: piloto controlado (IA solo en reconocimiento, sobre un activo no crítico); métricas y validación (comparar hallazgos de IA contra los manuales); ampliar con supervisión (escaneo y priorización con aprobación humana por paso); e integración continua (IA en el pipeline DevSecOps, con auditoría permanente).

Y seis reglas de higiene: definir un scope estricto antes de activar cualquier agente autónomo; mantener un humano en el loop para explotación de alto impacto; registrar y auditar cada decisión de la IA; validar manualmente los hallazgos críticos; reentrenar los modelos con nuevas TTPs y CVEs; y establecer límites de scope y rate-limit técnico para cada agente.

6. El laboratorio: auditar un contrato con IA

La charla cierra con un ejercicio reproducible: tomar contratos vulnerables de práctica, subirlos al motor de Hashlock, revisar los hallazgos y validarlos a mano. En la demostración, el escáner marca una vulnerabilidad crítica: la función initializer del contrato es pública y carece del modificador correspondiente, así que puede llamarse múltiples veces. Un atacante puede reinvocarla para reasignarse la variable de propietario, obteniendo control total —incluidos los derechos de actualización y la capacidad de drenar fondos—. Al desplegar el proxy y llamar a initializer en transacciones separadas se abre además una condición de carrera: quien llegue primero se convierte en propietario.

HOJA DE RUTA DE ADOPCIÓN

01
Piloto controlado
IA solo en reconocimiento, sobre un activo no crítico.

02
Métricas y validación
Comparar hallazgos de IA contra los manuales.

03
Ampliar con supervisión
Escaneo y priorización con aprobación humana por paso.

04
Integración continua
IA en el pipeline DevSecOps, con auditoría permanente.

Conclusiones

Tres ideas para llevarse. La IA acelera, no reemplaza: comprime reconocimiento, escaneo y priorización, pero el juicio del pentester sigue siendo lo que convierte datos en hallazgos.

Todo PoC se valida a mano: un payload o un hallazgo generado por un agente se confirma manualmente antes de llegar al cliente. Reportar un falso positivo cuesta más credibilidad que la que ahorra la automatización.

Y se empieza pequeño y con gobernanza: piloto acotado, medición de falsos positivos, escalamiento con auditoría y humano en el loop. Para la región, el mensaje es alentador: las herramientas inventariadas son en su mayoría abiertas o freemium. La barrera ya no es el presupuesto: es el criterio, y ese se entrena practicando.

> **La IA acelera; no reemplaza el juicio del pentester.**

> **Todo PoC se valida a mano.**

FUENTES Y RECURSOS

I Congreso IA-LATAM
Casas, C. (2026). *Pentesting con IA* [presentación].

Video de la presentación
Casas, C. (2026). *¿Cómo está transformando la inteligencia artificial las pruebas de penetración?* Canal de Comunidad IA LATAM en YouTube.

Estudios citados
Redfox Cybersecurity (2026), *The State of AI Pentesting in 2026*; encuesta SANS Institute (2025); pronóstico Gartner (2025); tiempo de reporte según Bishop Fox.

CAI: Cybersecurity AI framework
Alias Robotics. Benchmarks públicos en arXiv:2504.06017.

Laboratorios de práctica
DeFiVulnLabs y Web3 (codeccs).

Fuente original
Artículo original en el blog de Comunidad IA LATAM

SOBRE LA AUTORA

Camila Casas
Pentester e Instructora en Ciberseguridad Ofensiva · Chair de Hacking con IA, Comunidad IA LATAM.
LinkedIn

Artículo elaborado a partir de la presentación oficial y su transcripción. Todo ejercicio de pentesting requiere autorización previa y por escrito del titular del sistema.

Autores de la edición

Cinco voces técnicas de Comunidad IA LATAM firman los artículos de este número.

Cristián Rojas Arredondo

Agentes · arnés · grafos · convergencia

LinkedIn

Rudy Pinochet Maturana

Producción segura · verificación · controles

LinkedIn

Alberto Barrera Vidal

Gobernanza · ISO/IEC 42001 · marco ADG

LinkedIn

Cristian Balboa Vidal

Ciberseguridad · seguridad ofensiva · defensa activa

LinkedIn

Camila Casas

Pentesting con IA · ciberseguridad ofensiva · validación humana

LinkedIn

Las personas detrás de esta edición

Créditos editoriales de Comunidad IA LATAM.

Andrés Cordero Bravo

DIRECTOR DE OPERACIONES

Ingeniero Informático, consultor de Ciberseguridad, asesor tecnológico del Departamento de Terapia Ocupacional y editor de la Revista Chilena de Terapia Ocupacional (ReChTO) de la Facultad de Medicina de la Universidad de Chile. Vincula innovación tecnológica y gestión de procesos.

LinkedIn

Nicole Miranda Hidalgo

LÍDER DE REVISTA LATAMIA

Periodista y especialista en comunicación digital. Líder de Revista IA LATAM, con formación en inteligencia artificial y Vibe Coding Seguro.

LinkedIn

Mg. Ing. Sebastián Vargas Yáñez

PRESIDENTE · COMUNIDAD IA LATAM

Fundador y presidente de Comunidad IA LATAM. Impulsa la revista como infraestructura de conocimiento abierto en español para toda América.

LinkedIn

Tamara López Leyton

VICEPRESIDENTA · COMUNIDAD IA LATAM

Vicepresidenta de Comunidad IA LATAM. Acompaña la conducción de la comunidad y el fin social del proyecto editorial: conectar talento y educar en español.

LinkedIn

LATAMIA

REVISTA DE COMUNIDAD IA LATAM

Conocimiento abierto para Latinoamérica.

comunidadialatam.org

LATAMIA · N.° 01 · 2026