BE SOFTWARE ARCHITECTURE WISE · CAPÍTULO 8

Seguridad, privacidad y cumplimiento desde el diseño

La seguridad y la privacidad son cualidades de la estructura de un sistema, determinadas por dónde se sitúan los límites de confianza, quién puede hacer qué y por dónde viajan los datos. Este capítulo trata los principios del diseño seguro, el modelado de amenazas con diagramas de flujo de datos, STRIDE y árboles de ataque, la identidad y el acceso con OAuth 2.0 y OpenID Connect, el cifrado, la gestión de claves y los secretos, zero trust entre servicios, la seguridad de la cadena de suministro de software, la privacidad desde el diseño en el marco del RGPD (GDPR) y la seguridad en operación. Está dirigido a desarrolladores y arquitectos que conocen las arquitecturas distribuidas y de datos de los Capítulos 6 y 7 y que ahora deben hacerlas seguras.

bewiseapps.com/es/software-architecture-wise/security-by-design/ · Versión 2026-11-03. Solo información educativa, no asesoramiento profesional.

DE UN VISTAZO

Aprenderás a

- Aplicar la tríada CIA, los principios de diseño de Saltzer y Schroeder, la defensa en profundidad y los valores predeterminados seguros a las decisiones arquitectónicas, y explicar de qué son responsables los arquitectos en un ciclo de vida de desarrollo seguro.
- Modelar las amenazas de un diseño con un diagrama de flujo de datos con límites de confianza, STRIDE y árboles de ataque, y valorar y registrar los riesgos y las mitigaciones resultantes.
- Distinguir la autenticación de la autorización, elegir entre sesiones y tokens, y explicar OAuth 2.0 con PKCE, las credenciales de cliente (client credentials), OpenID Connect, y RBAC, ABAC y el control de acceso basado en relaciones.
- Decidir qué cifrar, dónde y con qué claves, usando cifrado de sobre (envelope encryption), un KMS, hash de contraseñas, gestión de secretos y tokenización.
- Explicar la arquitectura zero trust y diseñar la seguridad entre servicios con TLS mutuo, identidad de carga de trabajo y puntos de aplicación de políticas.
- Reducir el riesgo de la cadena de suministro de software con SBOM, el marco SLSA, la firma, la fijación de versiones (pinning) y las compilaciones aisladas, y resumir las obligaciones del Reglamento de Ciberresiliencia de la UE (Cyber Resilience Act).
- Diseñar conforme a los principios del RGPD (GDPR), la protección de datos desde el diseño y por defecto, y los derechos de los interesados, incluida la supresión en copias de seguridad, cachés y logs de eventos.
- Planificar el logging de seguridad, la detección, la gestión de vulnerabilidades y la respuesta a incidentes, incluida la notificación de violaciones en 72 horas que exige el RGPD.

Antes de empezar

Capítulos 6 y 7, en especial los microservicios, la arquitectura dirigida por eventos y la propiedad de los datos; la sección del Capítulo 2 sobre la seguridad como atributo de calidad; la sección del Capítulo 0 sobre redes, HTTPS y TLS.

Tiempo

Unas 4 horas de lectura; 45 minutos para la prueba.

Términos clave

límite de confianza · mínimo privilegio · defensa en profundidad · modelado de amenazas · STRIDE · OAuth 2.0 · OpenID Connect · PKCE · cifrado de sobre (envelope encryption) · zero trust (confianza cero) · TLS mutuo · SBOM · seudonimización · crypto-shredding (destrucción criptográfica)

La seguridad es una propiedad de la estructura

La mayoría de las brechas graves no se deben a un ataque criptográfico exótico. Proceden de decisiones estructurales: un servicio interno que confiaba en cualquier llamante de la red, una API de administración que se saltaba una comprobación de autorización, una copia de seguridad de la base de datos copiada a un bucket de almacenamiento que cualquiera podía leer, una clave de acceso subida en un commit a un repositorio, una dependencia que nadie sabía que el sistema usaba. Cada una de ellas fue una decisión arquitectónica, normalmente tomada de forma implícita, sobre dónde reside la confianza, qué componente puede hacer qué y por dónde viajan los datos. Por eso este capítulo trata la seguridad como un atributo de calidad diseñado en la estructura, en el sentido de los escenarios de seguridad del Capítulo 2, y no como una lista de productos que se compran al final.

También es la razón por la que la seguridad es inseparable de los estilos de los Capítulos 6 y 7. Cada límite de servicio, cola, caché, réplica y copia analítica aumenta la superficie de ataque y crea otro lugar donde hay que proteger los datos. La distribución multiplica el número de límites de confianza; la arquitectura de datos decide cuántas copias de datos personales existen y cuánto tiempo viven. El ejemplo recurrente es la plataforma de reparto de comida que se usa a lo largo de todo el curso: clientes, restaurantes y repartidores, un API gateway, los servicios de pedidos, pagos, despacho y notificaciones, un flujo de eventos y un almacén de datos.

Cómo encajan las ocho secciones

Las cuatro primeras secciones construyen el núcleo del diseño seguro. **Principios de seguridad** establece el vocabulario: la tríada CIA, los principios de diseño de 1975 de Saltzer y Schroeder, como el mínimo privilegio y los valores predeterminados a prueba de fallos, la defensa en profundidad, los valores predeterminados seguros, el OWASP Top 10 y un ciclo de vida de desarrollo seguro basado en el SSDF del NIST. **Modelado de amenazas** convierte esos principios en una actividad de diseño repetible: dibujar un diagrama de flujo de datos con sus límites de confianza, preguntarse qué puede salir mal con STRIDE y árboles de ataque, valorar los riesgos y decidir las mitigaciones. **Identidad y acceso** trata de quién llama: las contraseñas tal como recomienda hoy el NIST, las passkeys (llaves de acceso), las sesiones y los tokens, OAuth 2.0 con PKCE, OpenID Connect y los modelos de autorización RBAC, ABAC y el control de acceso basado en relaciones. **Protección de datos** trata el cifrado en tránsito y en reposo, el hash de contraseñas, la gestión de claves con cifrado de sobre (envelope encryption), los secretos, la tokenización y la decisión sobre qué cifrar y dónde.

Las cuatro secciones siguientes amplían la mirada. **Zero trust** sustituye la idea de una red interna de confianza por la autenticación y la autorización de cada solicitud entre servicios, mediante TLS mutuo, certificados y puntos de decisión y de aplicación de políticas. **Seguridad**

de la cadena de suministro trata las dependencias, los sistemas de compilación y las imágenes de contenedor como parte de la superficie de ataque, con listas de materiales de software, el marco SLSA, la firma y el Reglamento de Ciberresiliencia de la UE (Cyber Resilience Act). **Privacidad desde el diseño** explica cómo el RGPD (GDPR) y el RGPD del Reino Unido (UK GDPR) dan forma a la arquitectura: minimización de datos, seudonimización, derechos de los interesados, conservación y supresión en las copias de seguridad y los logs de eventos. **Seguridad en operación** cierra el ciclo con el logging de seguridad, la detección, la gestión de vulnerabilidades, la respuesta a incidentes y la notificación de violaciones de datos.

Qué deciden los arquitectos y qué delegan

Un arquitecto no necesita ser un pentester ni un criptógrafo, y no debe inventar criptografía. Su trabajo es tomar las decisiones que son caras de cambiar y que ningún equipo puede tomar por sí solo: dónde se sitúan los límites de confianza, qué proveedor de identidad y qué modelo de tokens usa toda la plataforma, qué componente es el punto de decisión de políticas para qué recursos, qué datos se clasifican como restringidos y, por tanto, se cifran con qué claves, cómo demuestran los servicios su identidad entre sí y cómo detectará la organización un ataque y responderá a él. Estas decisiones tienen su lugar en los registros de decisiones de arquitectura y en los conceptos transversales de una descripción arc42, como cualquier otra decisión significativa.

Los desarrolladores son responsables de un conjunto de inquietudes distinto e igual de importante: validar la entrada, codificar la salida, usar correctamente las bibliotecas de autenticación y autorización de la plataforma, manejar los errores sin fallar en abierto, mantener las dependencias actualizadas y escribir pruebas del comportamiento de seguridad. Ambos se encuentran en el modelado de amenazas, en las revisiones de diseño seguro y en las comprobaciones automatizadas del pipeline de entrega. Donde los profesionales discrepan, por ejemplo sobre cuánto trabajo de seguridad trasladar a los desarrolladores o sobre lo formal que debe ser el modelado de amenazas, el capítulo presenta ambas posturas.

Leyes, normas y precisión

La seguridad y la privacidad están llenas de normas numeradas y disposiciones legales, y este capítulo las nombra con precisión: las publicaciones especiales del NIST, las RFC del IETF, los proyectos de OWASP, PCI DSS, el RGPD (GDPR) y el RGPD del Reino Unido (UK GDPR), y el Reglamento de Ciberresiliencia de la UE. Las normas cambian, así que cada mención indica la edición vigente en la última revisión que figura en la sección y remite a la fuente oficial. El material sobre la regulación explica cómo afecta la ley a la arquitectura; es información, no asesoramiento jurídico, y los abogados y el delegado de protección de datos de cada organización siguen siendo las personas a quienes consultar sobre una situación concreta.

La idea que debes llevarte

Decide de forma deliberada, pronto y por escrito en quién y en qué confía tu sistema, para qué y por qué, y haz que cada una de esas decisiones de confianza sea verificable. La mayoría de las técnicas de este capítulo, desde los modelos de amenazas y el mínimo privilegio hasta los ámbitos (scopes) de OAuth, el cifrado de sobre, el TLS mutuo, las compilaciones firmadas y los logs de auditoría, son formas de sustituir la confianza implícita por una confianza explícita y comprobable. Un sistema diseñado así limita el daño de la brecha que acabará llegando y puede mostrar a sus clientes, auditores y reguladores cómo los protege.

SECCIÓN 8.1 DE 8.8 · INTERMEDIO · 28 MIN DE LECTURA · ÚLTIMA REVISIÓN: NOVIEMBRE DE 2026

Principios de seguridad para arquitectos

ANTES DE LEER ESTO

Esta sección se basa en:

- 0.4 Comunicación en red
- 0.6 API e integración
- 2.7 Seguridad, usabilidad y más

Términos anteriores que utiliza: Tríada CIA, API gateway (puerta de enlace de API), TLS y HTTPS, Límite de tasa

LA IDEA EN BREVE

Piensa en cómo se protege una sucursal bancaria. La cámara acorazada no está protegida por una sola cerradura, sino por varias: una puerta principal cerrada con llave, un personal que solo tiene llaves de las salas que necesita, una cámara que requiere dos personas para abrirse, cámaras de vigilancia y una alarma que avisa a la policía. El edificio se diseñó con todo esto en mente desde el principio; nadie atornilló una cámara acorazada a una tienda después. La seguridad del software funciona igual. Los arquitectos deciden pronto dónde están las puertas cerradas, quién recibe qué llaves y qué ocurre cuando falla una cerradura, para que un solo error no lo abra todo.

LO QUE NECESITAS SABER

- La seguridad es una cualidad de la estructura: dónde se sitúan los límites de confianza, qué componente puede hacer qué y por dónde viajan los datos determinan la mayoría de las propiedades de seguridad, y la mayoría de las brechas explotan decisiones de ese tipo y no la criptografía.
- La confidencialidad, la integridad y la disponibilidad, junto con la autenticidad, la responsabilidad y el no repudio, deben nombrarse para cada activo porque entran en conflicto: fallar en cerrado protege los datos, pero cuesta disponibilidad.
- Los principios de Saltzer y Schroeder de 1975 (economía del mecanismo, valores predeterminados a prueba de fallos, mediación completa, diseño abierto, separación de privilegios, mínimo privilegio, mínimo mecanismo común, aceptabilidad psicológica) siguen siendo la base del diseño seguro.

- La defensa en profundidad superpone controles independientes para que un solo fallo no deje un activo al descubierto; los valores predeterminados seguros y un camino pavimentado (paved road) hacen que el camino fácil sea el seguro.
- Reduce la superficie de ataque y sitúa la autenticación, la autorización y la validación en cada límite de confianza; todo lo que está en el cliente queda fuera del límite.
- El OWASP Top 10:2025 es una lista de concienciación, no una norma; el SSDF del NIST (SP 800-218) describe un ciclo de vida de desarrollo seguro, y los arquitectos son responsables de las decisiones de seguridad que afectan a varios equipos, mientras que cada equipo es responsable de aplicarlas correctamente.

La seguridad como cualidad de la estructura

El Capítulo 2 presentó la seguridad como atributo de calidad: la capacidad del sistema de proteger datos y funciones frente al uso no autorizado mientras sirve a quienes sí están autorizados, expresada mediante escenarios de seguridad comprobables que nombran un atacante, un ataque, un objetivo y una respuesta medible. Este capítulo da el paso siguiente y se pregunta cómo produce la arquitectura esa cualidad. La respuesta es que la mayoría de las propiedades de seguridad las decide la estructura: qué componentes existen, cuáles confían en cuáles, dónde se almacenan y se copian los datos y por qué pocos puntos debe pasar cada solicitud.

Piensa en cómo se producen las brechas reales. Un atacante cambia el número de pedido en una URL y ve la dirección de otro cliente, porque el servicio comprobó que el usuario había iniciado sesión, pero no que el pedido fuera suyo. Un bucket de almacenamiento con exportaciones de la base de datos se puede leer desde internet porque alguien cambió un valor predeterminado hace años. Un servicio interno de informes acepta cualquier solicitud que venga de dentro de la red, así que una sola computadora portátil comprometida llega a todos los registros de clientes. Ninguno de estos casos necesita criptografía ingeniosa para explotarse; cada uno es una decisión de diseño sobre la confianza que nadie tomó de forma explícita. Seguridad desde el diseño significa tomar esas decisiones de forma deliberada, pronto y por escrito, y después hacer que sea difícil violarlas por accidente.

Por eso la arquitectura de seguridad se sitúa junto a las demás inquietudes transversales de la sección 8 de arc42: un modelo de identidad, un modelo de autorización, un concepto de protección de datos y un concepto de logging que todos los equipos siguen. El resto de esta sección establece el vocabulario y los principios sobre los que se apoyan las demás secciones del capítulo.

Confidencialidad, integridad y disponibilidad en un sistema real

La tríada CIA es el punto de partida. En la plataforma de reparto de comida, la **confidencialidad** abarca los nombres, direcciones, números de teléfono e historiales de pedidos de los clientes, la ubicación en tiempo real y los documentos de identidad de los repartidores, los datos bancarios a los que se pagan las liquidaciones de los restaurantes y los propios secretos de la plataforma. La **integridad** abarca los precios, los totales del carrito, los reembolsos, los pagos a los repartidores y la carta que ve el cliente: un atacante que puede poner a cero el precio de un plato, o desviar las liquidaciones de un restaurante a su propia cuenta bancaria, no ha leído nada secreto, pero ha causado un daño grave. La **disponibilidad** importa sobre todo durante el pico de la noche, cuando un ataque de denegación de servicio o una infección de ransomware cuesta ingresos a cada minuto.

ISO/IEC 25010 también incluye la **autenticidad** (las identidades se pueden demostrar), la **responsabilidad** (las acciones se pueden atribuir a quien las realizó), el **no repudio** (se puede demostrar que las acciones ocurrieron) y, como novedad de la edición de 2023, la **resistencia** (el producto sigue funcionando como se espera bajo ataque). Los arquitectos deben nombrar qué propiedades importan para cada activo, porque tiran en direcciones distintas. Fallar en cerrado protege la confidencialidad y la integridad, pero cuesta disponibilidad: si el servicio de autorización está caído, un servicio de pedidos que falla en cerrado rechaza los pedidos. Fallar en abierto preserva la disponibilidad y se juega todo lo demás. La elección correcta varía según la operación: ver una carta pública puede fallar en abierto hacia una copia en caché, mientras que emitir un reembolso debe fallar en cerrado.

Los principios de diseño de Saltzer y Schroeder

En 1975, Jerome Saltzer y Michael Schroeder, a partir de su experiencia con el sistema operativo Multics en el MIT, publicaron ocho principios de diseño para los mecanismos de protección en *The Protection of Information in Computer Systems*. Medio siglo después siguen siendo la base más citada del diseño seguro, porque tratan de la estructura y no de la tecnología. La tabla reformula cada uno y da un ejemplo actual de la plataforma de reparto de comida.

Los ocho principios de diseño con ejemplos actuales

Principio	Qué pide	Ejemplo actual
Economía del mecanismo	Mantener el diseño de seguridad tan pequeño y sencillo como sea posible, para poder inspeccionarlo y probarlo a fondo.	Una única biblioteca de autorización o un único motor de políticas, compartido y bien revisado, en lugar de comprobaciones de permisos escritas a mano de forma distinta en cada servicio.
Valores predeterminados a prueba de fallos	Basar el acceso en un permiso explícito; ante la duda, o cuando algo falla, la respuesta es no.	Un nuevo endpoint de la API está denegado hasta que una política conceda acceso a él; un motor de políticas inalcanzable significa denegar, no permitir.
Mediación completa	Comprobar cada acceso a cada objeto todas las veces, no solo la primera.	El servicio de pedidos comprueba la propiedad en cada `GET /orders/{id}` en lugar de fiarse de cualquier ID que llegue de un usuario con sesión iniciada; las decisiones en caché caducan pronto.
Diseño abierto	La seguridad no debe depender de que el diseño sea secreto; solo las claves y las contraseñas deben ser secretas.	Algoritmos publicados (AES, TLS 1.3) y protocolos estándar (OAuth 2.0) que siguen siendo seguros aunque los atacantes lean el código fuente.
Separación de privilegios	Siempre que sea posible, exigir dos condiciones independientes (dos llaves) antes de conceder el acceso.	Los reembolsos por encima de un umbral necesitan un segundo aprobador; un lanzamiento a producción necesita tanto una revisión de código como un pipeline superado; inicio de sesión multifactor.
Mínimo privilegio	Dar a cada programa y a cada usuario solo los privilegios necesarios para su tarea, durante el tiempo necesario.	La cuenta de base de datos del servicio de carta no puede leer el esquema de pagos; el rol en la nube de un job de compilación puede subir imágenes, pero no eliminar clústeres.
Mínimo mecanismo común	Minimizar los mecanismos compartidos entre usuarios, porque cada mecanismo compartido es una vía por la que puede filtrarse información.	Cachés, colas o pools de conexiones separados por inquilino o por nivel de confianza; una caché compartida con claves mal diseñadas sirve los datos de un usuario a otro.
Aceptabilidad psicológica	La protección debe ser fácil de usar correctamente, o la gente le buscará atajos.	Passkeys en lugar de reglas complejas de contraseñas; valores predeterminados seguros en la plataforma; advertencias poco frecuentes y que, por eso, se leen.

DE DÓNDE VIENE

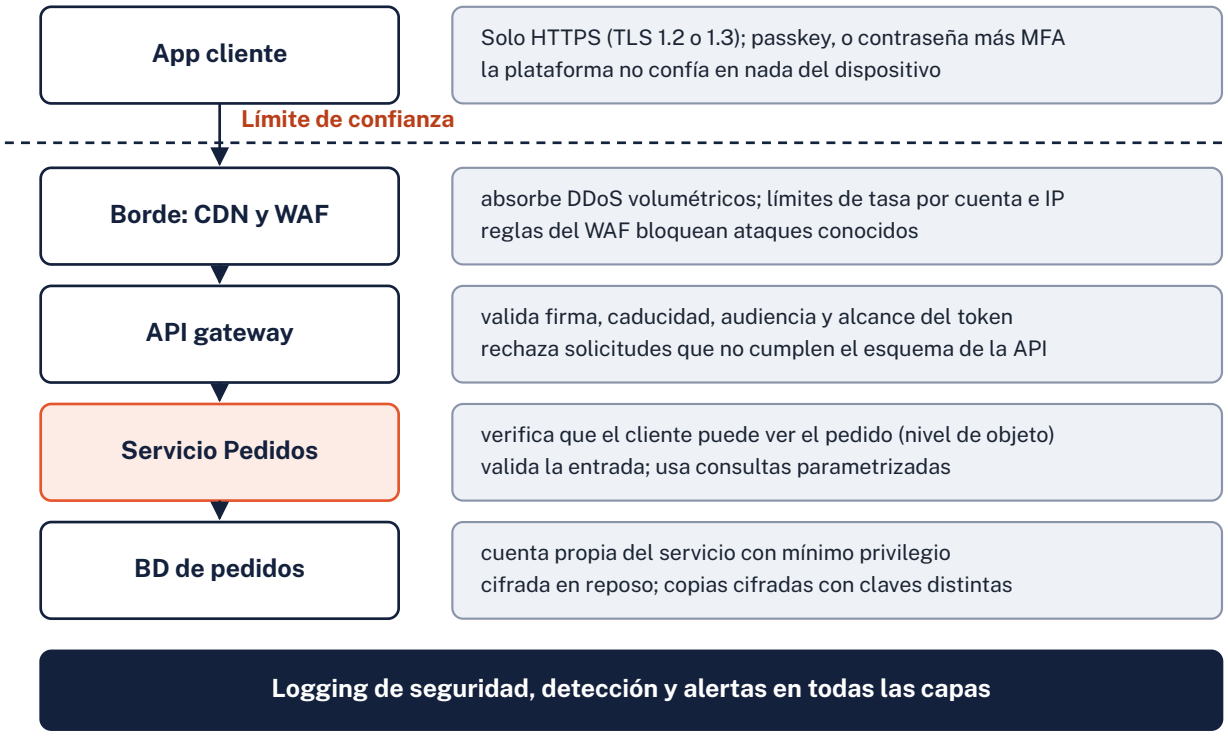
De dónde vienen los principios

El artículo de Saltzer y Schroeder apareció en los *Proceedings of the IEEE* en septiembre de 1975. Además de los ocho principios, describía dos más, el **factor de trabajo** (comparar el costo de eludir un mecanismo con los recursos del atacante) y el **registro del compromiso** (dejar constancia de que se produjo un compromiso aunque no pueda evitarse), que los autores señalaron que solo se aplican de forma imperfecta a los sistemas informáticos. Ambos siguen vigentes hoy: el factor de trabajo en el hash de contraseñas y en la longitud de las claves, el registro del compromiso en los logs de auditoría a prueba de manipulaciones. El diseño abierto reformula el principio de Auguste Kerckhoffs de 1883 para los cifrados militares: un sistema debe seguir siendo seguro aunque todo lo relativo a él, salvo la clave, sea público.

Defensa en profundidad

La defensa en profundidad consiste en superponer varios controles independientes, de modo que el fallo o la elusión de uno no deje el activo al descubierto. La idea procede de la fortificación militar y es práctica habitual tanto en la ingeniería de seguridad física (safety) como en la seguridad. Su valor depende por completo de la palabra *independientes*: dos capas que fallan por la misma razón, como dos servicios que confían en la misma cabecera falsificada, son una sola capa dibujada dos veces.

El diagrama muestra las capas en la ruta de un pedido en la plataforma de reparto de comida. Si la validación de tokens del gateway tiene un error, la comprobación a nivel de objeto del servicio de pedidos sigue impidiendo que un cliente lea el pedido de otro. Si un atacante roba las instantáneas de disco de la base de datos, el cifrado en reposo con claves guardadas en otro lugar las protege. Si todo eso falla, la monitorización debería detectar el patrón inusual de solicitudes y alguien debería responder. La defensa en profundidad es también la respuesta honesta a la pregunta "¿y si falla este control?", que todo modelo de amenazas plantea.



Cada capa aplica sus propios controles; saltarse un único control no deja los datos al descubierto.

SI ESTO ES NUEVO PARA TI

Paso a paso: cuando dos cerraduras son en realidad una

La idea que requiere más atención aquí es que la defensa en profundidad solo funciona si las capas son independientes, de modo que cada una pueda detener un ataque que haya superado a las demás. Sigue una solicitud a lo largo de la ruta del pedido en el diagrama.

- Un cliente con la sesión iniciada pide ver su pedido 1042. La app envía la solicitud con el token de acceso del cliente, que indica quién es.
- El API gateway comprueba que el token es auténtico, no ha caducado y está destinado a esta API, y pasa la solicitud al servicio de pedidos.
- Ahora el cliente modifica la solicitud y cambia 1042 por 1043, el pedido de otra persona. El token sigue siendo válido, así que el gateway la deja pasar: no puede saber quién es el dueño de cada pedido. El servicio de pedidos lo comprueba, ve que el pedido 1043 no es de este cliente y se niega.
- Esa segunda comprobación es una capa separada. Aunque un error dejara pasar un token incorrecto por el gateway, el servicio de pedidos seguiría negándose a mostrar a un cliente el pedido de otro.
- Pero supón que el servicio de pedidos también comprobara el token, con la misma biblioteca compartida que el gateway, y que un defecto de esa biblioteca aceptara tokens sin una firma válida. Entonces un solo token falsificado engañaría a las dos comprobaciones a la vez. Dos comprobaciones que fallan por la misma razón son una sola capa dibujada dos veces.
- Las capas inferiores siguen la misma regla. El cifrado en reposo protege las instantáneas de disco robadas solo porque sus claves se guardan en otro lugar, y la monitorización está ahí para detectar un ataque si todo lo demás falla.
- **La lección:** para cada control, pregúntate '¿y si falla este?', y asegúrate de que lo que lo derrote no derrote también a las capas que hay detrás.

Seguro por defecto y seguro desde el diseño

Un producto es seguro por defecto cuando su configuración de fábrica es la segura, de modo que debilitar la protección requiere un acto deliberado. Muchas brechas grandes se debieron a valores predeterminados inseguros que nadie cambió: bases de datos que escuchaban en internet sin contraseña, consolas de administración con las credenciales de fábrica, buckets de almacenamiento que podían hacerse públicos con un solo ajuste. Los proveedores han reaccionado: desde abril de 2023, por ejemplo, los nuevos buckets de Amazon S3 se crean con Block Public Access activado y las listas de control de acceso desactivadas. En 2023, la Agencia de Ciberseguridad y Seguridad de las Infraestructuras de EE. UU. (CISA) y agencias

asociadas de varios países publicaron una guía conjunta que instaba a los fabricantes a hacer productos seguros desde el diseño y por defecto, y en 2024 CISA lanzó un compromiso voluntario, el Secure by Design pledge, para los fabricantes de software.

Para los arquitectos, el principio de seguro por defecto se aplica a la plataforma tanto como a los productos. Si la plantilla de servicio compartida ya incluye la validación de tokens, el logging de seguridad estructurado, un usuario de contenedor que no es root y una cuenta de base de datos con mínimo privilegio, cada nuevo servicio empieza siendo seguro y los equipos tienen que renunciar a ello de forma explícita, lo que se ve en la revisión. Es lo que algunas organizaciones llaman un **camino pavimentado (paved road)**: el camino más fácil es también el seguro.

Muy relacionado está cómo se comporta un sistema cuando algo sale mal. El OWASP Top 10:2025 añadió una categoría para la mala gestión de condiciones excepcionales, que incluye fallar en abierto: un error en una comprobación de permisos que se captura y se ignora, un timeout que por defecto da "permitido", un fallo de análisis que se salta la validación. Los valores predeterminados a prueba de fallos, aplicados en el código, significan que toda ruta de excepción en una decisión de seguridad termina en denegación.

Superficie de ataque y límites de confianza

La superficie de ataque es todo lo que un atacante puede alcanzar e influir: los endpoints públicos y sus parámetros, los puertos abiertos, las subidas de archivos, las colas de mensajes que aceptan entradas de socios, los webhooks, las consolas de administración, las apps para dispositivos móviles que se pueden descompilar, y el código de terceros y las personas en los que confía el sistema. Michael Howard, de Microsoft, popularizó la idea de medirla a principios de la década de 2000; la regla práctica es hacerla lo más pequeña posible y saber exactamente qué hay en ella. Eliminar un endpoint sin uso, cerrar a internet un puerto de administración o desactivar una funcionalidad que nadie usa reduce el riesgo de forma más barata que cualquier control añadido. Cada elección de estilo arquitectónico del Capítulo 6 tiene aquí un costo de seguridad: cada nuevo servicio accesible por red añade endpoints, credenciales y tráfico que proteger.

Un límite de confianza es el punto en el que los datos o el control pasan entre partes con distintos niveles de confianza o de privilegio. El más evidente está entre internet y la plataforma, pero hay muchos más: entre la app del cliente y la API, entre la tableta de un restaurante y el servicio de carta, entre servicios de equipos distintos, entre la plataforma y un proveedor de pagos, entre el código de la aplicación y la base de datos, y entre los usuarios corrientes y los administradores. Los límites de confianza importan porque es ahí donde deben estar las comprobaciones: la autenticación, la autorización, la validación de la entrada y la

codificación de la salida se hacen cuando los datos cruzan un límite, nunca suponiendo que el otro lado ya las ha hecho. Dibujar los límites de confianza en un diagrama de flujo de datos es el primer paso del modelado de amenazas.

ATENCIÓN

El cliente está fuera del límite

Todo lo que se ejecuta en el dispositivo del cliente, ya sea una app para dispositivos móviles, una aplicación web de una sola página o la tableta de un restaurante, está bajo el control del atacante. Los precios, descuentos, permisos y totales de pedido calculados en el cliente son sugerencias; el servidor debe recalcularlos o verificarlos. Ocultar un botón en la interfaz de usuario no es autorización. Las claves de API incrustadas en una app para dispositivos móviles no son secretas, porque la app se puede descompilar.

El OWASP Top 10 como lista de concienciación

El OWASP Top 10 es la lista clasificada de los riesgos de seguridad más críticos de las aplicaciones web que elabora el Open Worldwide Application Security Project a partir de datos de vulnerabilidades aportados y de una encuesta a la comunidad. La edición actual, el **OWASP Top 10:2025**, es la octava. Frente a la edición de 2021, el control de acceso roto (broken access control) sigue en primer lugar y ahora incluye la falsificación de solicitudes del lado del servidor (SSRF), la configuración de seguridad incorrecta sube al segundo puesto, la antigua categoría de componentes vulnerables y desactualizados se amplía hasta convertirse en **fallos de la cadena de suministro de software**, y una nueva categoría cubre la **mala gestión de condiciones excepcionales**.

OWASP Top 10:2025 y las palancas arquitectónicas

Categoría	Qué sale mal	Palanca arquitectónica
A01 Broken Access Control (control de acceso roto)	Los usuarios actúan fuera de sus permisos, por ejemplo leyendo el pedido de otro cliente al cambiar un ID; incluye la falsificación de solicitudes del lado del servidor.	Un único modelo de autorización, denegar por defecto, comprobaciones a nivel de objeto en cada servicio, acceso de red saliente restringido.
A02 Security Misconfiguration (configuración de seguridad incorrecta)	Valores predeterminados inseguros, almacenamiento abierto, páginas de error demasiado detalladas, funcionalidades innecesarias activadas.	Infraestructura como código, plantillas reforzadas, comprobaciones automatizadas de políticas en el pipeline.
A03 Software Supply Chain Failures (fallos de la cadena de suministro de software)	Dependencias, herramientas de compilación y canales de distribución vulnerables o comprometidos.	Inventario de dependencias (SBOM), fijación de versiones (pinning), procedencia firmada; consulta seguridad de la cadena de suministro.
A04 Cryptographic Failures (fallos criptográficos)	Datos sensibles sin cifrar, algoritmos débiles, mala gestión de las claves.	Clasificación de datos, TLS en todas partes, un KMS y cifrado de sobre.
A05 Injection (inyección)	Datos no confiables interpretados como código: SQL, comandos del sistema operativo, cross-site scripting.	Consultas parametrizadas, codificación de la salida según el contexto, API seguras del framework.
A06 Insecure Design (diseño inseguro)	Controles ausentes o ineficaces que ninguna cantidad de código correcto puede arreglar.	Modelado de amenazas, patrones de diseño seguro, casos de abuso junto a las historias de usuario.
A07 Authentication Failures (fallos de autenticación)	Relleno de credenciales (credential stuffing), flujos de recuperación débiles, defectos en la gestión de sesiones.	Un proveedor de identidad central, MFA o passkeys, límites de tasa en el inicio de sesión.
A08 Software or Data Integrity Failures (fallos de integridad del software o de los datos)	Confiar en código, actualizaciones o datos sin verificar su integridad, como actualizaciones sin firmar o una deserialización insegura.	Verificación de firmas, comprobaciones de integridad de los datos que cruzan límites.
A09 Security Logging and Alerting Failures (fallos de logging y alertas de seguridad)	Los ataques no se registran, o se registran pero nadie los advierte ni actúa ante ellos.	Un estándar de logging de seguridad, alertas, runbooks de incidentes.

Categoría	Qué sale mal	Palanca arquitectónica
A10 Mishandling of Exceptional Conditions (mala gestión de condiciones excepcionales)	Errores gestionados de forma incoherente, sistemas que fallan en abierto, mensajes de error que revelan detalles.	Fallar en cerrado, gestión centralizada de errores, límites de recursos.

Trata la lista como lo que OWASP dice que es: un documento de concienciación, no una norma. Es útil para la formación, para comprobar que un modelo de amenazas no ha pasado por alto toda una clase de problemas y para hablar con las partes interesadas. No es un conjunto completo de requisitos, y "cumplimos el OWASP Top 10" significa poco. Para requisitos verificables, el Application Security Verification Standard (ASVS) de OWASP enumera controles comprobables en tres niveles de rigor, y el OWASP API Security Top 10, independiente del anterior, cubre los riesgos propios de las API, como la autorización a nivel de objeto rota (broken object-level authorisation).

Un ciclo de vida de desarrollo seguro y el shift left

Un ciclo de vida de desarrollo seguro integra actividades de seguridad en cada etapa de la entrega, en lugar de depender de una prueba justo antes del lanzamiento. Microsoft hizo obligatorio su Security Development Lifecycle en 2004, tras el memorando Trustworthy Computing de Bill Gates de 2002, y la idea se extendió por todo el sector. La referencia actual es el Secure Software Development Framework (SSDF) del NIST, SP 800-218, cuya versión 1.1 se publicó en febrero de 2022; el NIST publicó un borrador de revisión (versión 1.2) para comentarios en diciembre de 2025, que no se había aprobado de forma definitiva en el momento de redactar este texto. El SSDF agrupa las prácticas en cuatro familias: **preparar a la organización** (roles, formación, cadenas de herramientas, requisitos de seguridad), **proteger el software** (proteger la integridad del código y de la compilación, conservar la procedencia), **producir software bien protegido** (diseñar para cumplir los requisitos de seguridad, modelar amenazas, revisar y probar, configurar de forma segura por defecto) y **responder a las vulnerabilidades** (identificarlas, corregirlas y analizar las causas raíz para que la misma clase no se repita). Describe resultados en lugar de prescribir herramientas, lo que lo convierte en una buena lista de comprobación para un arquitecto que evalúa un proceso de entrega.

Hacer shift left (desplazar a la izquierda) es adelantar el trabajo de seguridad: requisitos de seguridad y casos de abuso junto con las historias de usuario, modelado de amenazas en el diseño, análisis estático y detección de secretos en cada commit, comprobación de dependencias en el pipeline. El argumento económico es conocido desde el Capítulo 1: una comprobación de autorización ausente que se detecta en una revisión de diseño cuesta una hora; el mismo defecto detectado tras una brecha cuesta una notificación a los reguladores.

Los profesionales discrepan sobre hasta dónde llevarlo. Sus defensores dicen que los desarrolladores que entienden de seguridad escriben menos vulnerabilidades. Sus críticos señalan que limitarse a poner herramientas de seguridad delante de los desarrolladores los inunda de hallazgos de baja calidad y convierte la seguridad en el trabajo a tiempo parcial de todos y en la especialidad de nadie; abogan por valores predeterminados seguros integrados en la plataforma (el camino pavimentado), para que el camino seguro no exija un esfuerzo adicional, y por especialistas en seguridad integrados allí donde el riesgo es mayor. La mayoría de las organizaciones maduras hacen ambas cosas y además siguen probando y monitorizando en producción, por lo que algunas hablan ya de llevar la seguridad a todas partes (shifting security everywhere).

Probar el sistema en ejecución: DAST y fuzzing

El análisis estático lee el código; estas técnicas lo ponen en funcionamiento. Las **pruebas dinámicas de seguridad de aplicaciones** (DAST) sondean una aplicación en ejecución desde fuera con solicitudes hostiles, sin necesitar su código fuente, en busca de vulnerabilidades como la inyección SQL, el cross-site scripting, la autenticación débil y los errores de configuración del servidor. Encajan con una ejecución nocturna contra un entorno de pruebas, pero solo encuentran aquello a lo que pueden llegar y suelen ser débiles en la lógica de negocio, como que un cliente vea el pedido de otro. El **fuzzing** alimenta un programa con entradas mal formadas o inesperadas hasta que falla. Comenzó con el grupo de Barton Miller en la Universidad de Wisconsin-Madison, cuya herramienta `fuzz` hizo que se cayeran o se colgaran entre una cuarta parte y un tercio de las utilidades UNIX que probaron. Es especialmente útil para los analizadores sintácticos, como el código de importación de cartas de la plataforma. OSS-Fuzz de Google lo ejecuta a escala para proyectos de código abierto.

Qué es responsabilidad de los arquitectos y qué de los desarrolladores

El trabajo de seguridad es compartido, y la confusión sobre quién es responsable de qué es en sí misma una vulnerabilidad. El arquitecto es responsable de las decisiones que cruzan los límites entre equipos y son caras de cambiar; los equipos son responsables de aplicarlas correctamente. Un patrón habitual es un pequeño equipo central de seguridad que fija los estándares y gestiona los servicios de seguridad de la plataforma, campeones de seguridad (security champions) en cada equipo de producto que dirigen el modelado de amenazas y las revisiones, y arquitectos que se aseguran de que las decisiones estructurales se tomen y se registren.

Reparto de las responsabilidades de seguridad

Inquietud	Arquitecto (y arquitectura de seguridad)	Equipos de desarrollo
Límites de confianza y flujos de datos	Definirlos, documentarlos y dirigir los modelos de amenazas de los cambios significativos.	Mantener el código coherente con ellos; plantear los cambios que desplazan un límite.
Identidad y acceso	Elegir el proveedor de identidad, el modelo de tokens, el modelo de autorización y dónde se decide la política.	Aplicar la autorización en cada endpoint y cada objeto; probarla.
Protección de datos	Clasificar los datos; decidir qué se cifra, dónde y con qué claves; elegir el KMS y el almacén de secretos.	Usar las bibliotecas aprobadas; no registrar nunca secretos ni datos personales sin necesidad.
Tratamiento de la entrada y la salida	Fijar la estrategia de validación y codificación en los límites; proporcionar bibliotecas compartidas.	Validar, codificar y parametrizar en cada handler.
Dependencias y compilación	Fijar los controles del pipeline, las fuentes aprobadas y la política de SBOM.	Mantener las dependencias al día; actuar ante las alertas en los plazos acordados.
Detección y respuesta	Definir el logging de seguridad, los eventos de auditoría y los roles ante incidentes.	Emitir los eventos acordados; corregir las vulnerabilidades dentro de los niveles de servicio acordados.

EJEMPLO PRÁCTICO

Ejemplo práctico: aplicar los principios a una función de reembolsos para el equipo de soporte

Los agentes de soporte de la plataforma de reparto de comida necesitan reembolsar a los clientes por entregas fallidas desde la herramienta de back office. Un primer diseño permite a cualquier agente llamar al endpoint de reembolsos del servicio de Pagos con cualquier número de pedido y cualquier importe. La tabla muestra cómo lo transforman los principios.

Mínimo privilegio	Los agentes reciben un permiso `refund:create` solo para los pedidos de su propio mercado; la identidad de servicio de la herramienta de back office puede llamar a la API de reembolsos, pero nunca leer datos de tarjetas.
Valores predeterminados a prueba de fallos	Si la comprobación de la política o la consulta del pedido fallan o agotan el tiempo de espera, el reembolso se rechaza con un mensaje de "inténtalo de nuevo"; no existe ninguna ruta alternativa que se salte las comprobaciones.
Mediación completa	El propio servicio de Pagos vuelve a comprobar cada reembolso frente al cargo original (el reembolso no puede superar el importe capturado menos los reembolsos anteriores), en lugar de fiarse de la herramienta de back office.
Separación de privilegios	Los reembolsos de más de £100, o un tercer reembolso al mismo cliente en una semana, necesitan la aprobación de un supervisor en la misma herramienta.
Economía del mecanismo	Usa el motor de políticas y la biblioteca de auditoría que ya tiene la plataforma; no se escribe ningún código de permisos especial para los reembolsos.
Aceptabilidad psicológica	La aprobación se solicita con un clic; los motivos se eligen de una lista breve, para que el control no frene a los agentes honrados.
Responsabilidad	Cada reembolso se escribe en un log de auditoría de solo anexado (append-only) con el agente, el aprobador, el pedido, el importe y el motivo.
Defensa en profundidad	A la herramienta de back office solo se accede mediante inicio de sesión único (SSO) con MFA resistente al phishing, desde dispositivos gestionados.

Ahora, tanto un agente deshonesto como un atacante que se haya hecho con la cuenta de un agente quedan limitados a reembolsos pequeños y registrados en un solo mercado.

El costo es un breve retraso en los reembolsos grandes y algo de trabajo de diseño adicional, lo que resulta mucho más barato que incorporar estos controles a posteriori, una vez descubierto el fraude.

COMPRUEBA LO QUE HAS ENTENDIDO

▼ **¿Por qué la defensa en profundidad depende de que las capas sean independientes?**

Si dos capas fallan por la misma razón, como que ambas confíen en la misma cabecera falsificada, un solo ataque las derrota a las dos, así que no dan más protección que una sola capa.

▼ **¿Qué principio de Saltzer y Schroeder se viola cuando un servicio comprueba que el usuario ha iniciado sesión, pero no que el pedido solicitado le pertenece?**

La mediación completa: hay que comprobar la autorización de cada acceso a cada objeto, no solo el hecho de que el usuario se haya autenticado.

▼ **Un equipo afirma que su aplicación "cumple el OWASP Top 10". ¿Qué tiene de malo esa afirmación?**

El Top 10 es una lista de concienciación de categorías de riesgo, no una norma con requisitos comprobables; la verificación necesita algo como el ASVS de OWASP y un modelo de amenazas del sistema real.

▼ **Da un ejemplo de fallar en abierto y explica por qué es peligroso.**

Una comprobación de permisos que trata un timeout del motor de políticas como "permitido". Un atacante capaz de ralentizar o averiar el motor de políticas obtiene entonces acceso a todo lo que este protege.

▼ **Inténtalo: Cualquier empleado de taquilla puede aplicar un descuento del 100% a cualquier reserva, y la app solo oculta el botón al personal junior. ¿Qué principios de seguridad incumple esto? Rediséñalo brevemente, indicando quién puede aprobar descuentos grandes.**

Ocultar un botón ocurre en el lado del cliente, así que queda fuera del límite de confianza, y falla la mediación completa: el servidor debe comprobar cada descuento. También falla el mínimo privilegio. Da a los empleados un permiso limitado, por ejemplo descuentos de hasta el 10% en su propio recinto, denegado por defecto. Exige una segunda persona para los descuentos mayores, lo que es separación de privilegios, y anota quién hizo qué. Falla en cerrado si la comprobación de permisos no está disponible, aceptando una cola más lenta.

FUENTES

- OWASP Top Ten project (OWASP Top 10:2025): https://owasp.org/www-project-top-ten/
- NIST SP 800-218: Secure Software Development Framework (SSDF) version 1.1: https://csrc.nist.gov/pubs/sp/800/218/final

- Saltzer and Schroeder, The Protection of Information in Computer Systems (1975): https://doi.org/10.1109/PROC.1975.9939
- OWASP SAMM: Software Assurance Maturity Model: https://owasp.org/www-project-samm/
- OWASP DevSecOps Guideline: Dynamic Application Security Testing: https://owasp.github.io/www-project-devsecops-guideline/latest/02b-Dynamic-Application-Security-Testing
- OWASP: Fuzzing: https://community.owasp.org/Fuzzing
- Miller, Fredriksen and So: An Empirical Study of the Reliability of UNIX Utilities: https://ftp1.cs.wisc.edu/par-distr-sys/papers/fuzz.pdf
- OSS-Fuzz documentation: https://google.github.io/oss-fuzz/

SECCIÓN 8.2 DE 8.8 · INTERMEDIO · 23 MIN DE LECTURA · ÚLTIMA REVISIÓN: NOVIEMBRE DE 2026

Modelado de amenazas: STRIDE, árboles de ataque y diagramas de flujo de datos

ANTES DE LEER ESTO

Esta sección se basa en:

- 2.7 Seguridad, usabilidad y más
- 4.3 El modelo C4
- 8.1 Principios de seguridad

Términos anteriores que utiliza: Límite de confianza, Diagrama de contenedores, Webhook, API gateway (puerta de enlace de API), Registro de decisiones de arquitectura (ADR)

LA IDEA EN BREVE

Antes de que una familia se vaya de vacaciones, alguien suele recorrer la casa preguntándose qué podría salir mal: ¿está cerrada la puerta trasera?, ¿tiene un vecino una llave?, ¿está la llave de repuesto debajo de la maceta?, ¿se acumulará el correo y delatará que no hay nadie en casa? Luego deciden qué hacer con cada cosa: cerrar la puerta, pedir al vecino que recoja el correo, cambiar la llave de sitio. El modelado de amenazas es ese mismo recorrido, pero por un diseño de software. El equipo dibuja cómo se mueven los datos por el sistema, se pregunta qué podría hacer un atacante en cada paso y decide qué corregir, qué aceptar y qué eliminar.

LO QUE NECESITAS SABER

- El modelado de amenazas responde a las cuatro preguntas de Shostack: en qué estamos trabajando, qué puede salir mal, qué vamos a hacer al respecto y si lo hemos hecho suficientemente bien.
- Un diagrama de flujo de datos con entidades externas, procesos, almacenes de datos, flujos de datos y límites de confianza es el modelo habitual; los flujos que cruzan límites de confianza merecen la mayor atención.
- STRIDE (Kohnfelder y Garg, Microsoft, 1999) nombra seis categorías de amenaza, cada una de las cuales viola una propiedad: suplantación o spoofing (autenticidad), manipulación o tampering (integridad), repudio (no repudio), divulgación de información (confidencialidad), denegación de servicio (disponibilidad) y elevación de privilegios (autorización).
- Los árboles de ataque (Schneier, 1999) modelan el objetivo de un atacante como nodos AND y OR; el camino más barato a través del árbol muestra dónde importa más una mitigación.

- Valora las amenazas por probabilidad e impacto, con los motivos por escrito; DREAD cayó en desuso porque sus puntuaciones eran subjetivas, y CVSS valora vulnerabilidades conocidas, no amenazas de diseño.
- Toda amenaza termina mitigada, eliminada, transferida o aceptada de forma explícita por alguien con autoridad para ello; modela las amenazas en la fase de diseño y de nuevo cada vez que cambien los límites de confianza, los datos o la autenticación.

Qué es el modelado de amenazas

El modelado de amenazas es la actividad de examinar un diseño para descubrir qué podría fallar en su seguridad y decidir qué hacer al respecto, antes de que lo descubran por ti los atacantes o los incidentes. Es la principal vía por la que los principios de seguridad se convierten en decisiones concretas, y la respuesta más directa a la categoría de OWASP de diseño inseguro: defectos que ninguna cantidad de código correcto puede arreglar porque el control necesario nunca se diseñó.

Adam Shostack, que dirigió el trabajo de modelado de amenazas en Microsoft y escribió el libro de referencia sobre el tema, reduce la actividad a cuatro preguntas. **¿En qué estamos trabajando?** (un modelo del sistema, normalmente un diagrama). **¿Qué puede salir mal?** (amenazas, encontradas de forma sistemática). **¿Qué vamos a hacer al respecto?** (mitigar, eliminar, transferir o aceptar cada amenaza). **¿Lo hemos hecho suficientemente bien?** (revisar el modelo, las amenazas y el seguimiento). Cualquier método que responda a las cuatro preguntas es modelado de amenazas; el resto de esta sección describe las técnicas que la mayoría de los equipos usan para responderlas.

En 2020, un grupo de profesionales, entre ellos Shostack, publicó el Threat Modeling Manifesto, inspirado en el Manifiesto Ágil. Valora una cultura de encontrar y corregir problemas de diseño por encima del cumplimiento de casillas, las personas y la colaboración por encima de los procesos y las herramientas, un camino de comprensión por encima de una instantánea de seguridad puntual, hacer modelado de amenazas por encima de hablar de él, y el refinamiento continuo por encima de una única entrega. Su idea es que el modelado de amenazas es un hábito del equipo que construye el sistema, no un documento que elaboran personas ajenas una vez al año.

Modelar el sistema: diagramas de flujo de datos y límites de confianza

La primera pregunta necesita un modelo del sistema que muestre adónde van los datos y dónde cambia la confianza. La opción tradicional es un diagrama de flujo de datos (DFD) con cinco tipos de elementos: **entidades externas** (personas o sistemas fuera de tu control, dibujados como rectángulos), **procesos** (código que ejecutas tú, dibujado como círculos o

elipses), **almacenes de datos** (bases de datos, archivos, colas y cachés, dibujados entre dos líneas paralelas), **flujos de datos** (flechas que muestran los datos que se mueven entre los demás elementos) y **límites de confianza** (líneas o recuadros discontinuos donde cambia el nivel de confianza).

Los DFD encajan con el modelado de amenazas porque los ataques siguen a los datos: cada flujo que cruza un límite de confianza es un lugar donde un atacante puede inyectar, interceptar o reenviar algo. Un diagrama de contenedores C4 del Capítulo 4 es un buen punto de partida: añade los límites de confianza, convierte cada flecha en un flujo de datos con dirección y etiqueta, e incluye los almacenes y las colas que los diagramas de contenedores a veces dejan implícitos. Mantén el diagrama en el nivel en el que se encuentran las amenazas: un diagrama por cada funcionalidad o flujo significativo es más útil que un único diagrama enorme de todo.

Encontrar amenazas con STRIDE

STRIDE lo crearon en Microsoft en 1999 Loren Kohnfelder y Praerit Garg como regla mnemotécnica para ayudar a los desarrolladores a pensar en las amenazas a sus productos. Cada letra nombra una categoría de amenaza, y cada categoría es la violación de una propiedad de seguridad, que es lo que la hace útil: es un estímulo sistemático, no una lista de ataques conocidos.

STRIDE y la propiedad que viola cada amenaza

Amenaza	Propiedad violada	Ejemplo de reparto de comida	Mitigaciones habituales
Spoofing (suplantación)	Autenticidad	Un atacante inicia sesión como cliente con una contraseña obtenida por relleno de credenciales; un proceso malicioso llama a Pagos haciéndose pasar por Pedidos.	Autenticación fuerte (MFA, passkeys), límites de tasa, TLS mutuo e identidad de carga de trabajo entre servicios.
Tampering (manipulación)	Integridad	Un cliente envía un total del carrito modificado; alguien altera un webhook de pago o una fila de la base de datos.	Recálculo en el servidor, firmas y MAC, TLS, acceso de escritura con mínimo privilegio.
Repudiation (repudio)	No repudio (responsabilidad)	Un cliente niega haber hecho un pedido; un agente de soporte niega haber emitido un reembolso, y nada demuestra lo contrario.	Logs de auditoría vinculados a identidades autenticadas, marcas de tiempo, almacenamiento a prueba de manipulaciones.
Information disclosure (divulgación de información)	Confidencialidad	Una API devuelve las direcciones de los clientes a repartidores que no les están haciendo una entrega; los mensajes de error revelan detalles internos.	Autorización en cada objeto, minimización de datos, cifrado, mensajes de error genéricos.
Denial of service (denegación de servicio)	Disponibilidad	Unos bots inundan el proceso de pago durante el pico de la noche; una consulta de búsqueda costosa agota la base de datos.	Límites de tasa y cuotas, timeouts, bulkheads, autoescalado, límites al costo de las consultas.
Elevation of privilege (elevación de privilegios)	Autorización	Un cliente llama a un endpoint de gerente de restaurante; una inyección permite a un atacante ejecutar código con la identidad del servicio.	Autorización con denegación por defecto, validación de la entrada, sandboxing, mínimo privilegio.

Una forma práctica de aplicar STRIDE es **STRIDE por elemento**: recorrer cada elemento del DFD y hacer solo las preguntas que se aplican a su tipo. Las entidades externas pueden ser suplantadas y pueden repudiar; los procesos están expuestos a las seis categorías; los

almacenes de datos, a la manipulación, la divulgación de información y la denegación de servicio (y al repudio cuando el almacén es un log); los flujos de datos, a la manipulación, la divulgación de información y la denegación de servicio. Dedicar la mayor parte de la atención a los flujos que cruzan límites de confianza mantiene el ejercicio centrado.

STRIDE encuentra amenazas; no las clasifica por prioridad ni elige mitigaciones. Además, en la práctica sus categorías se solapan (una identidad suplantada suele llevar a la divulgación de información o a la elevación de privilegios), lo cual no importa: el objetivo es encontrar la amenaza, no clasificarla a la perfección. Para las amenazas a la privacidad, como vincular o identificar a personas a partir de datos que parecen inofensivos, el método LINDDUN de la KU Leuven cumple el papel que STRIDE cumple para la seguridad; se trata junto con la privacidad desde el diseño.

Árboles de ataque

Un árbol de ataque, popularizado por Bruce Schneier en un artículo de 1999 en *Dr. Dobb's Journal*, mira el sistema desde el lado del atacante. La raíz es un objetivo, como conseguir comida sin pagar; cada hijo es una forma de alcanzar a su padre. Los hijos de un nodo OR son alternativas; todos los hijos de un nodo AND se necesitan a la vez. Las hojas pueden anotarse con propiedades como el costo, la habilidad necesaria o si el paso es siquiera posible, y esos valores se propagan hacia arriba por el árbol: el costo de un nodo OR es el de su hijo más barato, y el de un nodo AND, la suma de sus hijos.

Los árboles de ataque complementan bien a STRIDE. STRIDE recorre el diseño elemento a elemento y es bueno para la cobertura; un árbol de ataque sigue a un atacante motivado y es bueno para mostrar qué camino es el más barato y, por tanto, dónde rinde más una mitigación. En el árbol de abajo, pagar con una tarjeta robada es relativamente costoso porque necesita dos cosas a la vez, mientras que cambiar el precio en la solicitud del cliente es trivial si el servidor se fía de él. El camino más barato fija el riesgo, así que recalcular cada precio en el servidor vale más que cualquier ajuste del sistema antifraude de tarjetas.



Un árbol de ataque para un objetivo del atacante; la hoja más barata muestra dónde importa más una mitigación.

SI ESTO ES NUEVO PARA TI

Paso a paso: leer un árbol de ataque

El punto en el que más gente tropieza aquí es cómo un árbol de ataque muestra dónde importa más una corrección. Un atacante toma la ruta más barata, así que un objetivo solo es tan difícil de alcanzar como su camino más barato. Recorre el árbol del diagrama de abajo arriba.

- Empieza por las hojas, cada una un único paso con un costo aproximado. Obtener cupones con cuentas falsas es barato; conseguir que una tarjeta robada supere 3-D Secure (la comprobación adicional del banco al pagar) y los controles antifraude es costoso; editar el total en la solicitud del cliente es trivial si el servidor se fía de él.
- 'Pago con tarjeta robada' es un nodo AND: el atacante necesita a la vez los datos de la tarjeta y una forma de superar los controles. Su costo es la suma de ambos, así que el único paso costoso hace costosa toda la rama.
- 'Cambiar el precio' es un nodo OR: cualquiera de sus hojas sirve, así que su costo es el de su hijo más barato. La rama sigue siendo trivial mientras el servidor se fíe de los precios que envía el cliente.
- El objetivo de arriba también es un nodo OR, y cuesta lo que cuesta su rama más barata: cambiar el precio. Ese es el riesgo real de la plataforma, por buenos que sean sus controles antifraude.
- Así que el equipo hace que el servidor recalcule cada precio. Ahora un total editado no consigue nada, y esa hoja deja de ser una opción.
- Vuelve a recorrer el árbol hacia arriba. Las hojas más baratas que quedan, como obtener cupones con cuentas falsas, son ahora las que fijan el riesgo, y es ahí donde la siguiente corrección rinde más.
- **La lección:** hacer más difícil un camino caro no cambia nada mientras quede uno barato. Encuentra el camino más barato, ciérralo y vuelve a mirar, porque un atacante motivado simplemente pasa al siguiente más barato.

Valorar el riesgo

Un modelo de amenazas suele encontrar más amenazas de las que un equipo puede abordar a la vez, así que cada una necesita una valoración. El enfoque habitual es juzgar la **probabilidad** (lo fácil, lo atractiva y lo expuesta que es) y el **impacto** (qué se pierde si ocurre: dinero, datos personales, disponibilidad, reputación, exposición legal) en escalas sencillas como baja, media y alta, y combinarlos en una matriz. El valor de la valoración reside menos en la precisión del

número que en un motivo escrito que otros puedan cuestionar. Las organizaciones con una gestión de riesgos madura usan a veces métodos cuantitativos como FAIR (Factor Analysis of Information Risk) para expresar el riesgo como un rango de pérdida esperada.

Conviene conocer dos atajos populares porque a menudo se usan mal. **DREAD** (damage, reproducibility, exploitability, affected users, discoverability: daño, reproducibilidad, explotabilidad, usuarios afectados y descubribilidad), que se usó en su día en Microsoft, puntuaba cada amenaza en cinco factores (a menudo de 1 a 10) y hacía la media. Cayó en desuso, también en Microsoft, porque distintas personas puntuaban la misma amenaza de forma muy diferente y el número promediado daba una falsa sensación de precisión; la descubribilidad, en particular, premia la ocultación, lo que contradice el diseño abierto. CVSS, el Common Vulnerability Scoring System que mantiene FIRST (versión 4.0, 2023), es la herramienta adecuada para otra tarea: valorar la gravedad de una vulnerabilidad conocida en un producto, como un defecto en una biblioteca. Su puntuación base describe la vulnerabilidad, no el riesgo de una amenaza de diseño en tu contexto, así que no debe usarse para priorizar el resultado de un modelo de amenazas.

Toda amenaza termina en una de cuatro decisiones: **mitigar** (añadir un control), **eliminar** (quitar la funcionalidad o los datos, a menudo la mejor opción), **transferir** (dejar que la asuma alguien mejor situado, como un proveedor de pagos que guarda los datos de las tarjetas) o **aceptar** (dejar constancia de que una persona con autoridad para ello ha aceptado el riesgo residual, y de por qué). La aceptación no registrada es el fallo más habitual: se encontró la amenaza, nadie la corrigió y nadie decidió no corregirla.

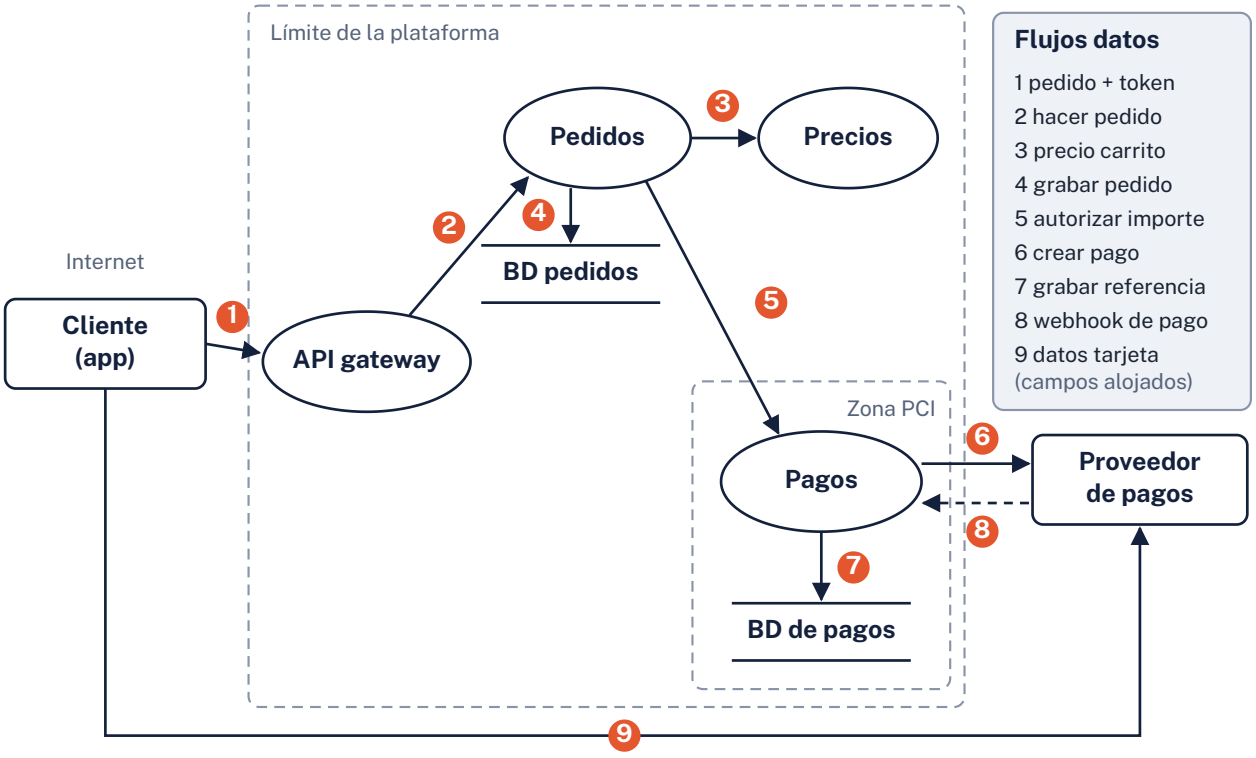
EJEMPLO PRÁCTICO

Ejemplo práctico: modelar las amenazas del proceso de pago

El equipo de reparto de comida está rediseñando el proceso de pago para que los clientes puedan pagar en la app con una tarjeta o un monedero digital. El arquitecto dirige una sesión de 90 minutos con los desarrolladores de pedidos y de pagos, un product owner y un campeón de seguridad. El DFD de abajo responde a la primera pregunta, en qué estamos trabajando. Muestra dos límites de confianza: la plataforma en su conjunto y una zona de pagos más estrecha y muy restringida (etiquetada como zona PCI) en torno al servicio de pagos y su base de datos, que guardan tokens y referencias de pago y se comunican con el proveedor de pagos; aislarlos mantiene pequeño el alcance de PCI DSS de la plataforma. Los datos de las tarjetas en sí (flujo 9) van directamente desde los campos de pago alojados en la app al proveedor de pagos y nunca entran en la plataforma.

1. ¿En qué estamos trabajando?	El DFD del proceso de pago de abajo: la app del cliente, el API gateway, Pedidos, Precios, Pagos, dos bases de datos y el proveedor de pagos, con dos límites de confianza (la plataforma y la zona PCI) y nueve flujos numerados.
2. ¿Qué puede salir mal?	STRIDE por elemento en cada flujo que cruza un límite (1, 5, 6, 8 y 9) y en la API de pedidos, más el árbol de ataque para conseguir comida sin pagar que se mostró antes.
3. ¿Qué vamos a hacer al respecto?	Cada amenaza recibe una valoración, una decisión (mitigar, eliminar, transferir o aceptar) y un responsable; las más importantes están en la tabla de abajo.
4. ¿Lo hemos hecho suficientemente bien?	Una revisión un mes después compara el modelo con lo que se construyó y comprueba que cada riesgo alto se mitigó o se aceptó por escrito.

La forma más barata de proteger los números de tarjeta es no guardarlos: el flujo 9 los mantiene por completo fuera de la plataforma, lo que elimina amenazas en lugar de mitigarlas.



El diagrama de flujo de datos del proceso de pago, con sus dos límites de confianza y sus flujos de datos numerados.

Amenazas seleccionadas del modelo de amenazas del proceso de pago

ID y ubicación	STRIDE	Amenaza	Riesgo	Decisión y mitigación
T1, flujo 1	S	El relleno de credenciales permite hacerse con cuentas que tienen tarjetas guardadas.	Alto	Mitigar: passkeys, o MFA en dispositivos nuevos; límites de tasa por cuenta e IP; comprobación de contraseñas filtradas.
T2, flujos 1 y 2	T	El cliente envía un precio, un descuento o un gasto de envío alterados.	Alto	Mitigar: Pedidos ignora los precios del cliente; Precios los recalcula a partir de las cartas y los cupones; el pago se crea por el total del servidor.
T3, flujo 5	S, E	Otro servicio interno, o un atacante dentro de la red, pide a Pagos que autorice o reembolse importes arbitrarios.	Alto	Mitigar: TLS mutuo con identidad de carga de trabajo; Pagos solo acepta autorizaciones de la identidad de Pedidos, vinculadas a un ID de pedido y a un importe.
T4, flujo 8	S, T	Un webhook falsificado de "pago realizado" marca como pagado un pedido sin pagar.	Alto	Mitigar: verificar la firma y la marca de tiempo del proveedor; confirmar el estado con la API del proveedor antes de liberar el pedido; procesar cada evento una sola vez.
T5, `GET /orders/{id}`	I	Enumerar los ID de pedido revela las direcciones de otros clientes (autorización a nivel de objeto rota).	Alto	Mitigar: comprobación de propiedad en cada solicitud; identificadores aleatorios solo como medida secundaria.
T6, BD de pedidos	I	Los registros de pedidos con direcciones se filtran desde una copia de seguridad o a través de una cuenta de analítica con permisos demasiado amplios.	Medio	Mitigar: cifrado en reposo con claves del KMS, cuentas con mínimo privilegio, copias de seguridad en una cuenta separada.

ID y ubicación	STRIDE	Amenaza	Riesgo	Decisión y mitigación
T7, flujo 1	D	Unos bots agotan la capacidad del proceso de pago en el pico de la noche.	Medio	Mitigar: límites de tasa y detección de bots en el borde, capacidad reservada para el proceso de pago, una sala de espera si hace falta.
T8, Pedidos	R	Un cliente niega haber hecho un pedido y no hay pruebas.	Bajo	Mitigar: evento de auditoría con el ID de usuario, el dispositivo, la hora y los detalles del pedido, conservado según la política.
T9, flujo 9	I	Si los números de tarjeta pasaran por la plataforma, todos los componentes entrarían en el alcance de PCI DSS.	Diseño	Eliminar y transferir: los campos alojados envían los datos de la tarjeta directamente al proveedor; la plataforma solo guarda tokens y referencias.

Las respuestas a las dos últimas preguntas son tan importantes como la tabla. Cada mitigación se convierte en un elemento del backlog con un responsable; T3 y T4 se convierten además en escenarios de seguridad con pruebas automatizadas (un webhook con una firma incorrecta debe rechazarse; una llamada a Pagos desde cualquier identidad distinta de Pedidos debe fallar). Las dos decisiones arquitectónicas, los campos de pago alojados y el TLS mutuo entre Pedidos y Pagos, se registran como ADR. Para la cuarta pregunta, el equipo revisa el modelo un mes después frente a lo que realmente se construyó, y el campeón de seguridad comprueba que cada elemento de riesgo alto se mitigó o se aceptó de forma explícita.

Cuándo y con qué frecuencia modelar las amenazas

El modelado de amenazas rinde más en la fase de diseño, cuando cambiar una decisión cuesta lo que una pasada de borrador por la pizarra. Después debe repetirse cada vez que cambia algo significativo: un nuevo límite de confianza o una nueva integración externa, un nuevo tipo de datos sensibles, un cambio en la autenticación o la autorización, un nuevo entorno de despliegue o el paso de un monolito a servicios. Una regla sensata es convertir "¿cambia esto el modelo de amenazas?" en una pregunta de las revisiones de diseño y de la plantilla de los registros de decisiones de arquitectura, y programar una revisión del modelo de cada sistema al menos una vez al año y después de cada incidente de seguridad.

El esfuerzo debe ser proporcionado, en el espíritu del enfoque guiado por el riesgo del Capítulo 1. Un nuevo microservicio que procesa pagos merece una sesión de media jornada con un DFD completo; añadir un campo a una carta pública necesita cinco minutos y un vistazo con STRIDE

al flujo modificado. Muchos equipos mantienen su DFD como código junto al sistema (OWASP Threat Dragon y la Threat Modeling Tool de Microsoft son herramientas gratuitas de diagramación, y bibliotecas como OWASP pytm generan diagramas y amenazas candidatas a partir del código), para que el modelo se revise y se actualice junto con el sistema que describe.

ATENCIÓN

Errores habituales en el modelado de amenazas

- Intentar modelar toda la empresa a la vez; empieza por una funcionalidad, un flujo o un servicio.
- Hacerlo una vez para una auditoría y archivar el documento; el modelo debe cambiar con el sistema.
- Dejarlo solo en manos del equipo de seguridad; las personas que construyen el sistema saben dónde están sus puntos débiles.
- Especular sobre quién es el atacante en lugar de sobre qué puede salir mal en el diseño.
- Enumerar amenazas sin decisiones, responsables ni seguimiento, o aceptar riesgos sin registrar quién los aceptó.
- Aceptar cientos de amenazas genéricas de una herramienta sin juzgar cuáles se aplican.

PARA PROFUNDIZAR

Otros métodos

PASTA (Process for Attack Simulation and Threat Analysis, descrito por Tony UcedaVélez y Marco Morana en 2015) es un método de siete etapas centrado en el riesgo que parte de los objetivos de negocio y termina en la simulación de ataques; encaja con organizaciones que quieren que el modelado de amenazas esté estrechamente ligado al riesgo de negocio. MITRE ATT&CK es una base de conocimiento de tácticas y técnicas reales de los atacantes, más útil para la ingeniería de detección en las operaciones de seguridad que para el diseño. Los sistemas que incluyen modelos de aprendizaje automático añaden amenazas como la inyección de prompts y el envenenamiento de los datos de entrenamiento, que se tratan en el Capítulo 12.

COMPRUEBA LO QUE HAS ENTENDIDO

▼ **¿Qué categorías de STRIDE se aplican a un flujo de datos en STRIDE por elemento?**

Manipulación, divulgación de información y denegación de servicio: los datos en tránsito se pueden alterar, leer o bloquear.

▼ ¿Por qué CVSS es la herramienta equivocada para priorizar las amenazas encontradas en un modelo de amenazas?

CVSS puntúa la gravedad de una vulnerabilidad conocida concreta en un producto; no tiene en cuenta tu diseño, tus activos ni tu contexto, así que no mide el riesgo de una amenaza a nivel de diseño.

▼ En un árbol de ataque, ¿cómo se calcula el costo de un nodo AND a partir de sus hijos, y por qué importa?

Es la suma de los costos de sus hijos, porque se necesitan todos los pasos. Por eso los nodos AND suelen ser caminos costosos, mientras que el hijo más barato de un nodo OR fija el riesgo.

▼ ¿Cuál es el fallo más habitual en el cuarto paso del modelado de amenazas?

Amenazas que se encontraron pero ni se corrigieron ni se aceptaron de forma explícita: nadie es responsable de ellas y nadie decidió convivir con ellas.

▼ Inténtalo: El proceso de pago tiene un webhook en el que el proveedor de pagos notifica el éxito. Usando STRIDE, nombra la amenaza si cualquiera puede enviarle solicitudes, otra amenaza más y una mitigación para cada una. ¿Cómo termina tu trabajo con cada amenaza?

Suplantación: un mensaje de éxito falsificado marca como pagado un pedido sin pagar. Mitígalo verificando la firma del proveedor y confirmando el pago con el proveedor. Denegación de servicio: inundar el endpoint bloquea las confirmaciones reales; mitígalo con límites de tasa y una cola. La manipulación del importe queda cubierta por la firma. El endpoint cruza un límite de confianza, así que merece atención. Cada amenaza debe terminar mitigada, eliminada, transferida o aceptada explícitamente por alguien con autoridad.

FUENTES

- Threat Modeling Manifesto (2020): https://www.threatmodelingmanifesto.org/
- OWASP Threat Modeling Cheat Sheet: https://cheatsheetseries.owasp.org/cheatsheets/Threat_Modeling_Cheat_Sheet.html
- Microsoft Threat Modeling Tool: STRIDE threat categories: https://learn.microsoft.com/en-us/azure/security/develop/threat-modeling-tool-threats
- Bruce Schneier, Attack Trees (Dr. Dobb's Journal, December 1999): https://www.schneier.com/academic/archives/1999/12/attack_trees.html
- FIRST: Common Vulnerability Scoring System: https://www.first.org/cvss/

SECCIÓN 8.3 DE 8.8 · AVANZADO · 29 MIN DE LECTURA · ÚLTIMA REVISIÓN: NOVIEMBRE DE 2026

Identidad y acceso: autenticación, autorización, OAuth 2.0 y OpenID Connect

ANTES DE LEER ESTO

Esta sección se basa en:

- 0.4 Comunicación en red
- 0.6 API e integración
- 8.1 Principios de seguridad

Términos anteriores que utiliza: Token de acceso, API gateway (puerta de enlace de API), TLS y HTTPS, HTTP

LA IDEA EN BREVE

Piensa en un gran edificio de oficinas. En recepción muestras un documento de identidad con foto y recibes una tarjeta de visitante: eso es la autenticación, demostrar quién eres. Después, la tarjeta abre unas puertas y otras no: eso es la autorización, decidir qué puedes hacer. Si envías a un mensajero a recoger un paquete por ti, no le das tu propia tarjeta; recepción le entrega un pase limitado que solo abre la sala de correo, y solo hoy. OAuth funciona como ese pase limitado, pero para aplicaciones. Y las propias máquinas del edificio, como los ascensores y el sistema de alarma, también necesitan sus propias tarjetas, para poder demostrarse entre sí que son auténticas.

LO QUE NECESITAS SABER

- La autenticación establece quién llama y es mejor centralizarla en un único proveedor de identidad; la autorización decide qué puede hacer con qué objeto y debe aplicarse donde están los datos, en cada servicio.
- NIST SP 800-63B (revisión 4, 2025) prioriza la longitud sobre las reglas de composición, exige comprobar las contraseñas contra listas de bloqueo de contraseñas comunes y filtradas, prohíbe los cambios periódicos forzados y espera opciones resistentes al phishing, como las passkeys.
- Las sesiones del lado del servidor son fáciles de revocar; los JWT autocontenidos escalan, pero no pueden revocarse antes de su expiración, así que mantenlos de corta duración, fija el algoritmo, comprueba el emisor (issuer), la audiencia y la expiración, y no los guardes en localStorage.

- OAuth 2.0 (RFC 6749) delega un acceso limitado mediante tokens; las apps de los usuarios deben usar el flujo de código de autorización con PKCE (RFC 7636), y los servicios, las credenciales de cliente; OAuth 2.1 sigue siendo un Internet-Draft en 2026.
- OpenID Connect añade un token de ID que le dice al cliente quién ha iniciado sesión; el token de ID es para el cliente y el token de acceso, para la API.
- RBAC, ABAC y el control de acceso basado en relaciones (Google Zanzibar, 2019) responden a necesidades distintas; los motores de políticas como OPA separan la decisión de su aplicación, y la identidad de carga de trabajo (SPIFFE, mTLS) sustituye a los secretos compartidos entre servicios.

Autenticación y autorización

Cada solicitud a la plataforma de reparto de comida plantea dos preguntas distintas. La autenticación pregunta *¿quién es?* y se responde con evidencias: una contraseña, una passkey, un certificado, un token firmado. La autorización pregunta *¿puede hacer esto, con esto, ahora?* y se responde con políticas: este cliente puede leer este pedido porque lo hizo él; este repartidor puede ver esta dirección de recogida porque el reparto le está asignado. HTTP también las separa: `401 Unauthorized` significa que la solicitud carece de una autenticación válida, mientras que `403 Forbidden` significa que se sabe quién llama, pero no tiene permiso.

La distinción importa para la arquitectura porque las dos se resuelven en lugares distintos. La autenticación es mejor centralizarla en un único proveedor de identidad (IdP) en el que confíen todas las aplicaciones, de modo que las reglas de contraseñas, la autenticación multifactor, la recuperación de cuentas y el bloqueo se implementen una sola vez. La autorización es distribuida por naturaleza: solo el servicio de pedidos sabe a quién pertenece un pedido, así que la decisión final sobre un pedido debe tomarse allí, aunque antes se hagan comprobaciones gruesas en el API gateway. La mayoría de las brechas de control de acceso del OWASP Top 10 proceden de confundir las dos: un endpoint que comprueba que el token es válido y luego devuelve cualquier objeto cuyo ID reciba.

Contraseñas, autenticación multifactor y passkeys

Las contraseñas siguen siendo el autenticador más común, y las Digital Identity Guidelines del NIST son las reglas más seguidas para ellas. La revisión 4 de SP 800-63B, publicada en julio de 2025, invierte varias costumbres arraigadas. La longitud importa más que la composición: una contraseña usada como único factor debe tener al menos 15 caracteres (al menos 8 cuando solo se usa como parte de una autenticación multifactor), y los verificadores deben aceptar al menos 64 caracteres, incluidos espacios y Unicode. Los verificadores no deben imponer reglas de composición, como exigir un dígito y un símbolo, que hacen las contraseñas más difíciles de recordar sin hacerlas mucho más difíciles de adivinar. Deben comprobar las contraseñas

nuevas contra una lista de bloqueo de contraseñas comunes, previsibles y filtradas. No deben forzar cambios periódicos, solo un cambio cuando haya indicios de que la contraseña está comprometida. Deben permitir los gestores de contraseñas y deberían permitir pegar, no deben ofrecer pistas de contraseña legibles antes de iniciar sesión y no deben pedir preguntas de seguridad. Los intentos fallidos deben tener un límite de tasa.

La autenticación multifactor (MFA) combina factores de distinto tipo: algo que sabes (una contraseña), algo que tienes (un teléfono o una llave de seguridad) y algo que eres (una huella dactilar, que normalmente solo se usa para desbloquear un dispositivo). Frena la mayor parte del relleno de credenciales (credential stuffing), en el que los atacantes prueban pares de usuario y contraseña filtrados de otros sitios. No todos los segundos factores son iguales: los códigos de un solo uso enviados por SMS pueden robarse mediante el duplicado de SIM (SIM swapping), y ningún código de un solo uso, venga de un SMS o de una app de autenticación, resiste el phishing, porque se puede engañar al usuario para que lo escriba en un sitio falso que lo reenvía de inmediato. Por eso las directrices del NIST exigen que los servicios con nivel de garantía de autenticación 2 ofrezcan al menos una opción resistente al phishing, y que los de nivel 3 la usen.

Una passkey (llave de acceso) es la opción resistente al phishing que la mayoría de los usuarios puede usar de verdad. Las passkeys son credenciales FIDO2, basadas en la Web Authentication API (WebAuthn) del W3C y en el Client to Authenticator Protocol (CTAP) de la FIDO Alliance. En el registro, el dispositivo del usuario crea un nuevo par de claves para ese sitio web; el servidor solo guarda la clave pública. Al iniciar sesión, el servidor envía un desafío aleatorio y el dispositivo lo firma con la clave privada después de que el usuario lo desbloquee con una huella, la cara o un PIN. Como el navegador vincula la credencial al origen del sitio web, un sitio de phishing que lo imite no puede obtener una firma válida, y como el servidor no guarda ningún secreto compartido, una brecha en la base de datos no proporciona nada que un atacante pueda reutilizar. Las passkeys pueden estar vinculadas a un dispositivo (en una llave de seguridad de hardware) o sincronizarse entre los dispositivos de un usuario mediante una plataforma o un gestor de contraseñas; la guía del NIST de 2025 acepta las passkeys sincronizadas en el nivel 2, mientras que el nivel 3 exige una clave que no pueda exportarse.

El punto débil arquitectónico de todo esquema de autenticación es la **recuperación de cuentas**. Si una passkey perdida puede sustituirse haciendo clic en un enlace enviado por correo electrónico, la fortaleza de todo el sistema es la de la cuenta de correo. Los flujos de recuperación merecen su propio modelo de amenazas, límites de tasa, avisos a los demás dispositivos del usuario y, en las cuentas de alto valor, esperas o comprobaciones humanas.

Sesiones y tokens

Una vez que el usuario se ha autenticado, el sistema debe reconocer sus solicitudes posteriores sin volver a preguntar. Hay dos diseños básicos. En una **sesión del lado del servidor**, el servidor guarda la sesión (usuario, expiración, quizá roles) y da al navegador un identificador de sesión largo y aleatorio en una cookie. En un diseño de **token autocontenido**, el emisor firma un token que lleva las afirmaciones (claims), normalmente un JSON Web Token (JWT, RFC 7519), y cualquier servicio que tenga la clave pública del emisor puede validarlo sin hacer una consulta.

Sesiones del lado del servidor frente a tokens autocontenidos

Aspecto	Sesión del lado del servidor	Token autocontenido (JWT)
Dónde vive el estado	En un almacén de sesiones en el servidor; el navegador solo guarda un identificador aleatorio.	En el token firmado; el servidor no guarda nada por usuario.
Revocación	Se borra la sesión: inmediata.	No es posible antes de la expiración sin una lista de denegación o una consulta, así que la vida útil debe ser corta.
Escalado entre servicios	Necesita un almacén de sesiones compartido o un enrutamiento persistente (sticky).	Cualquier servicio puede validar la firma localmente.
Tamaño en la red	Una cookie pequeña.	De cientos de bytes a kilobytes en cada solicitud.
Uso típico	Sesiones de navegador para aplicaciones web y backends for frontends.	Tokens de acceso para API, apps móviles y llamadas entre servicios.
Riesgos principales	Fijación de sesión y falsificación de solicitudes entre sitios (cross-site request forgery), que se mitigan regenerando el ID al iniciar sesión y con cookies SameSite.	Robo mediante inyección de scripts si se guarda en el almacenamiento del navegador; errores de validación; claims desactualizados.

Para las aplicaciones de navegador, es difícil superar una sesión del lado del servidor en una cookie marcada como `HttpOnly` (invisible para JavaScript), `Secure` (solo HTTPS) y `SameSite`. Las aplicaciones de una sola página usan cada vez más un **backend for frontend** (BFF): el navegador solo guarda una cookie de sesión para su propio backend pequeño, que guarda los tokens de OAuth y llama a las API en su nombre, un enfoque que recomienda la guía

del grupo de trabajo OAuth del IETF para aplicaciones basadas en navegador. Los JWT brillan entre servicios y en las apps móviles, donde una comprobación sin estado en cada salto evita un cuello de botella central.

ATENCIÓN

Errores comunes con los JWT

- **Sin revocación.** Un JWT robado es válido hasta que expira. Mantén los tokens de acceso de corta duración (minutos), usa tokens de actualización (refresh tokens) que el servidor de autorización pueda revocar y mantén una lista de denegación de ID de token para emergencias, o usa tokens opacos comprobados por introspección (RFC 7662) cuando importe la revocación inmediata.
- **Confusión de algoritmos.** Algunas bibliotecas aceptaban antes el algoritmo indicado en la cabecera del token, incluido `none`, o verificaban un token RS256 como HS256 usando la clave pública como secreto HMAC, lo que permitía a los atacantes falsificar tokens. Fija siempre en el código los algoritmos esperados, como aconseja la RFC 8725 (buenas prácticas actuales para JWT).
- **Guardar tokens en `localStorage`.** Cualquier script que se ejecute en la página puede leerlo, así que un solo fallo de cross-site scripting filtra el token de cada usuario. Es preferible usar cookies HttpOnly a través de un BFF, o mantener los tokens en memoria y vincularlos al cliente con tokens restringidos al remitente (sender-constrained), como DPoP (RFC 9449).
- **Firmado no es cifrado.** La carga útil (payload) de un JWT solo está codificada en Base64url; cualquiera que tenga el token puede leerla. No pongas datos personales en ella.
- **Saltarse la comprobación de los claims.** Valida el emisor, la audiencia y la expiración, no solo la firma, o un token emitido para una API será aceptado por otra.

```python
import jwt  # PyJWT
from jwt import PyJWKClient

ISSUER = "https://id.fooddelivery.example/"
jwks = PyJWKClient(ISSUER + ".well-known/jwks.json")  # issuer's public keys

def verify_access_token(token: str) -> dict:
    signing_key = jwks.get_signing_key_from_jwt(token)
    return jwt.decode(
        token,
        signing_key.key,
        algorithms=["RS256"],      # pinned: never read from the token header
        audience="orders-api",      # the token must be meant for this API
        issuer=ISSUER,
        options={"require": ["exp", "iat", "sub"]},
    )  # raises an exception on any failure: fail closed
```

Validación de un token de acceso en una API de Python con PyJWT: el algoritmo, la audiencia y el emisor los fija la API, nunca se toman del token.

SI ESTO ES NUEVO PARA TI

Paso a paso: un token que nadie puede retirar

La idea más difícil de esta sección es por qué un token autocontenido no puede simplemente cancelarse, y qué hacen los diseñadores al respecto. Sigue un robo con cada uno de los diseños de la tabla.

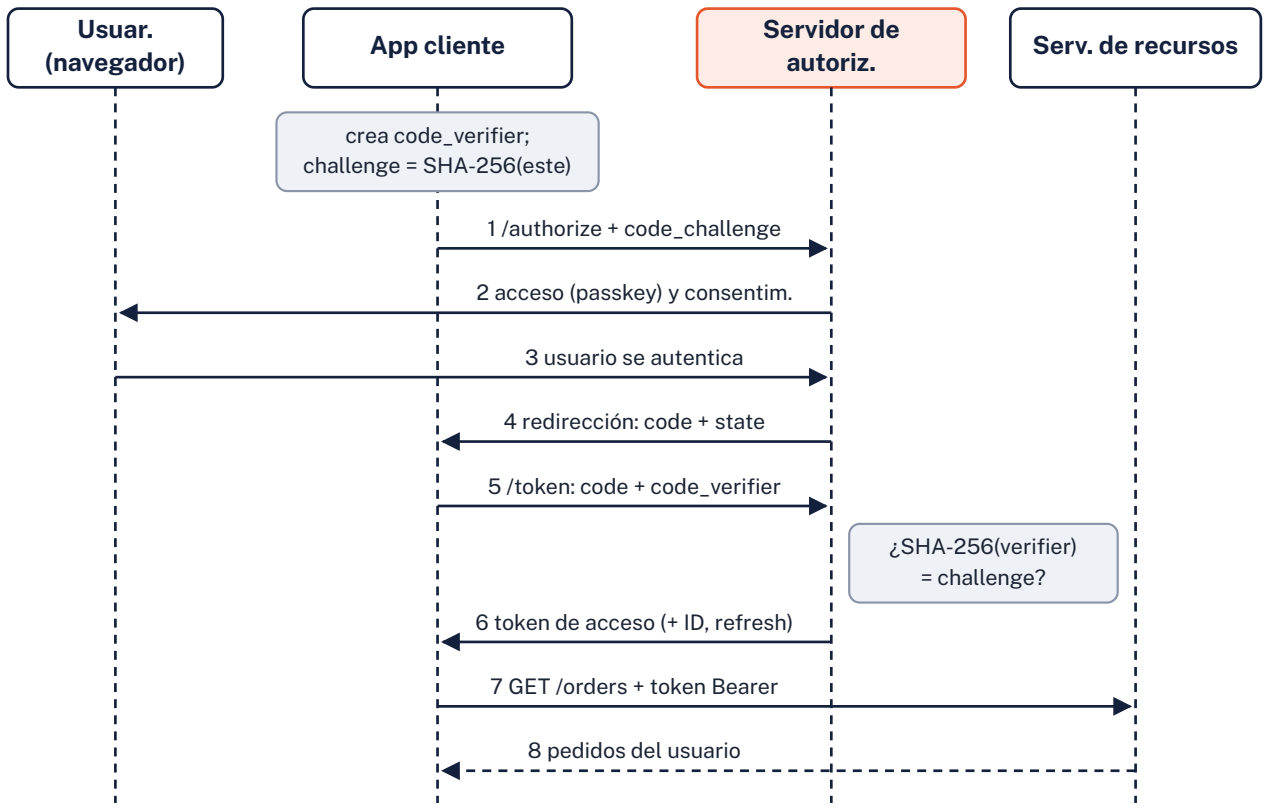
- Con una sesión del lado del servidor, el servidor registra el inicio de sesión del cliente en su almacén de sesiones y solo da al navegador un identificador largo y aleatorio en una cookie, que consulta en cada solicitud.
- Un atacante roba el identificador. En cuanto se detecta el robo, la plataforma borra el registro de la sesión, así que la siguiente solicitud del atacante no encuentra nada y se rechaza: la cancelación es inmediata.
- Con el otro diseño, la app móvil del cliente guarda un JSON Web Token firmado que indica quién es el cliente y cuándo expira el token. El servicio de pedidos comprueba la firma, igual que una tienda comprueba la marca de agua de un billete, y el emisor, la audiencia y la expiración del token, y después confía en él sin preguntar a nadie.
- Ahora roban el token. No hay ningún registro que borrar, así que todos los servicios lo siguen aceptando, porque sigue siendo auténtico, hasta que expira.
- De ahí los remedios de "Errores comunes con los JWT": haz que los tokens de acceso expiren en minutos y renuévalos con un token de actualización que el emisor pueda revocar; cuando importe cancelar de inmediato, mantén una lista de denegación de tokens robados, o usa tokens que cada servicio deba comprobar con el emisor, lo que reintroduce precisamente la consulta que los tokens evitan.
- Así que es un compromiso: los tokens permiten que cada servicio compruebe por sí solo a quien llama, lo que conviene a las apps móviles y a las llamadas entre servicios; las sesiones son fáciles de cancelar y convienen a las aplicaciones de navegador.
- **La lección:** por sí solo, un token autocontenido es una promesa que no puede retirarse antes de que expire. Haz que la promesa sea corta y añade una consulta solo donde importe cancelar de inmediato.

OAuth 2.0: autorización delegada

OAuth 2.0 (RFC 6749, 2012) resuelve un problema concreto: permitir que una aplicación actúe en nombre de un usuario sin conocer su contraseña. Antes de OAuth, un servicio que quería importar tus contactos te pedía la contraseña de tu correo electrónico. OAuth define cuatro roles: el **propietario del recurso** (el usuario), el **cliente** (la aplicación que quiere acceder), el **servidor de autorización** (que autentica al usuario, le pide su consentimiento y emite los tokens) y el **servidor de recursos** (la API que guarda los datos). El cliente recibe un **token de**

acceso limitado por **ámbitos (scopes)** (como `orders:read`), una vida útil y una audiencia, y a menudo un **token de actualización** (refresh token) para obtener nuevos tokens de acceso sin volver a preguntar al usuario.

OAuth define varias formas, llamadas concesiones (grants), de obtener un token. Para cualquier aplicación que actúe en nombre de un usuario, ya sea una app web del lado del servidor, una app de una sola página o una app móvil, la concesión recomendada es el **flujo de código de autorización con PKCE** (Proof Key for Code Exchange, RFC 7636, 2015). Primero, el cliente crea un `code_verifier` aleatorio y envía solo su hash SHA-256, el `code_challenge`, cuando redirige al usuario al servidor de autorización. Después de que el usuario inicie sesión y dé su consentimiento, el servidor de autorización lo redirige de vuelta con un código de autorización de corta duración. A continuación, el cliente canjea el código por tokens en una solicitud directa por el canal trasero (back channel), enviando el verificador original; el servidor comprueba que su hash coincide con el desafío. Un atacante que intercepte el código, por ejemplo una app móvil maliciosa registrada para la misma dirección de redirección, no puede canjearlo sin el verificador, que nunca salió del cliente.



El flujo de código de autorización con PKCE: solo el hash del verificador viaja en las redirecciones del navegador (el canal frontal); el propio verificador se envía directamente al endpoint de tokens.

La **concesión de credenciales de cliente** cubre el caso en el que no hay usuario: un servicio obtiene un token con su propia autoridad, por ejemplo el servicio de despacho cuando llama a la API de un proveedor de mapas asociado, o el sistema de un socio cuando llama a la API de estado de pedidos de la plataforma. El cliente debería autenticarse ante el endpoint de tokens con una clave privada (una aserción JWT firmada) o con TLS mutuo (RFC 8705) en lugar de con un secreto compartido. Hoy se desaconsejan dos concesiones más antiguas: la **concesión implícita**, que devolvía los tokens en la URL del navegador, y la **concesión de credenciales de contraseña del propietario del recurso**, en la que el cliente recogía la contraseña del usuario y, por tanto, anulaba el propósito de OAuth. La RFC 9700, la OAuth 2.0 Security Best Current Practice publicada en enero de 2025, recoge estas lecciones, incluidas la coincidencia exacta de las direcciones de redirección y los tokens restringidos al remitente.

OAuth 2.1 reúne OAuth 2.0 y estas buenas prácticas en un solo documento: PKCE es obligatorio en el flujo de código de autorización, se eliminan las concesiones implícita y de contraseña, las direcciones de redirección deben coincidir exactamente y los tokens Bearer ya no pueden enviarse en las cadenas de consulta de las URL. En la última revisión de esta sección seguía siendo un Internet-Draft del IETF, no una RFC (comprueba su estado actual), pero sus reglas ya describen lo que hacen los servidores de autorización bien gestionados.

GUÍA DE DECISIÓN

Guía de decisión: ¿qué flujo de OAuth 2.0?

Para cada tipo de llamante de las API de reparto de comida.

Q1 **¿El dispositivo no puede mostrar un navegador o es difícil escribir en él, como la pantalla de cocina de un restaurante?**

- **Sí** Usa la concesión de autorización de dispositivo (RFC 8628): el dispositivo muestra un código corto que el usuario aprueba en su teléfono.
- **No** Pasa a la siguiente pregunta.

Q2 **¿Hay una persona presente, y la aplicación actúa en su nombre?**

- **Sí** Usa el flujo de código de autorización con PKCE, tanto para apps web como de una sola página y móviles; nunca las concesiones implícita o de contraseña.
- **No** Pasa a la siguiente pregunta.

Q3 **¿Es uno de tus propios servicios que llama a otro dentro de tu plataforma?**

- **Sí** Es preferible la identidad de carga de trabajo (TLS mutuo con identidades emitidas por la plataforma); si la API espera tokens de OAuth, usa credenciales de cliente con autenticación del cliente basada en claves.
- **No** Pasa a la siguiente pregunta.

Q4 **¿Es el sistema de un socio que llama a tu API con su propia autoridad?**

- **Sí** Usa credenciales de cliente con tokens de ámbito estrecho y corta duración, un cliente por socio y límites de tasa por socio.
- **No** Revisa el requisito: un diseño en el que tu cliente necesita la contraseña de un usuario es una señal de alarma.

OpenID Connect, inicio de sesión único y SAML

OAuth emite tokens de acceso para las API; por sí solo, no le dice al cliente quién es el usuario. Usar un token de acceso como prueba de inicio de sesión es un error bien conocido, porque un token emitido para una aplicación puede reutilizarse en otra que no compruebe su audiencia. OpenID Connect (OIDC, Core 1.0, 2014) añade la capa de identidad que faltaba. Con el ámbito `openid`, el servidor de autorización, que ahora actúa como proveedor de OpenID, devuelve también un **token de ID**: un JWT firmado para el cliente, que indica el emisor, el identificador estable del usuario (`sub`), la audiencia (el ID del cliente), las horas de emisión y de expiración, un `nonce` que lo vincula a este inicio de sesión y, opcionalmente, cómo y cuándo se autenticó el usuario. OIDC también estandariza un endpoint de información del usuario y unos

documentos de descubrimiento que publican los endpoints y las claves de firma del proveedor. La regla práctica: el token de ID es para el cliente, para saber quién ha iniciado sesión; el token de acceso es para la API, para autorizar las llamadas.

El inicio de sesión único (SSO) viene de forma natural: en cuanto un usuario tiene una sesión con el proveedor de identidad, cada aplicación que confía en él puede iniciarle la sesión sin pedirle otra contraseña, y desactivar la cuenta en un solo lugar corta el acceso en todas partes. En las empresas, el SSO a menudo sigue usando **SAML 2.0** (un estándar de OASIS de 2005), que intercambia aserciones XML firmadas a través del navegador. La plataforma de reparto de comida podría usar OIDC para sus propias apps y aceptar SAML de las grandes cadenas de restaurantes que quieren que su personal inicie sesión en el portal de restaurantes con sus cuentas corporativas. El alta y la baja de cuentas en los distintos sistemas suele automatizarse con SCIM (RFC 7643 y 7644).

Modelos de autorización: RBAC, ABAC y relaciones

La autenticación le dice al sistema quién llama; un modelo de autorización decide qué puede hacer. Predominan tres modelos, y los sistemas reales suelen combinarlos.

Tres modelos de autorización

Modelo	Cómo decide	Ventajas	Puntos débiles
Basado en roles (RBAC)	Los permisos se conceden a roles; a los usuarios se les asignan roles. Formalizado por David Ferraiolo y Richard Kuhn en el NIST en 1992.	Fácil de entender y de auditar; se corresponde con funciones laborales como agente de soporte.	No puede expresar "solo los suyos" sin reglas adicionales; acotar por sucursal o por mercado lleva a una explosión de roles.
Basado en atributos (ABAC)	Una política evalúa atributos del sujeto, el recurso, la acción y el entorno; descrito en NIST SP 800-162 (2014).	Muy expresivo: mercado, clasificación de los datos, hora, postura del dispositivo, importe.	Es más difícil responder "¿quién puede acceder a esto?"; las políticas pueden volverse opacas y lentas.
Basado en relaciones (ReBAC)	El acceso sigue relaciones almacenadas como un grafo: un usuario gestiona una sucursal, una sucursal pertenece a un restaurante, un pedido pertenece a una sucursal.	Natural para compartir y para jerarquías; escala a permisos detallados, por objeto.	Necesita un almacén de relaciones dedicado y consistente, y un modelado cuidadoso.

El control de acceso basado en relaciones se conoce sobre todo por el artículo de Google de 2019 sobre **Zanzibar**, el sistema global de autorización que hay detrás de Google Drive, Calendar, YouTube y muchos otros productos. Zanzibar almacena tuplas de relación de la forma *object#relation@user*, evalúa los permisos siguiéndolas y usa tokens de consistencia para que una comprobación de permisos nunca se evalúe con datos más antiguos que el contenido que protege, el llamado problema del nuevo enemigo (new enemy problem): quitar a alguien de un documento y después añadir contenido sensible no debe permitirle verlo. Entre los sistemas de código abierto inspirados en él están SpiceDB, OpenFGA y Ory Keto.

Sea cual sea el modelo, conviene separar el punto de decisión de políticas (PDP), que evalúa la política y responde permitir o denegar, del **punto de aplicación de políticas** (PEP), el código del gateway o del servicio que hace la pregunta y aplica la respuesta. Los motores de políticas de uso general, como Open Policy Agent (OPA, un proyecto graduado de la Cloud Native Computing Foundation, con políticas escritas en su lenguaje Rego) y Cedar (liberado como código abierto por Amazon Web Services en 2023), permiten que la política viva en archivos versionados, se pruebe como el código y se evalúe dentro del proceso o como sidecar, cerca del servicio. Una organización en capas sensata en la plataforma es esta: el API gateway rechaza las solicitudes sin un token válido o sin el ámbito adecuado (grueso), y cada servicio decide las cuestiones a nivel de objeto con datos que solo él tiene (fino). La autorización rota a nivel de objeto es el primer riesgo del OWASP API Security Top 10 (edición de 2023) precisamente porque la comprobación fina es la que los equipos olvidan.

EJEMPLO PRÁCTICO

Ejemplo práctico: quién puede cambiar una carta, marcar un pedido como listo o ver una dirección de recogida

Los restaurantes de la plataforma van desde cafeterías únicas hasta cadenas con decenas de sucursales. Requisitos: los propietarios de las cadenas gestionan todas las sucursales; los gerentes de sucursal editan la carta de su propia sucursal; el personal de cocina marca como listos los pedidos de su sucursal; un repartidor ve los datos de recogida solo de los pedidos que tiene asignados; los gerentes pueden emitir reembolsos de cortesía de hasta £20.

Primer intento: RBAC	Roles `owner`, `manager`, `staff`, `courier`. Entonces cada comprobación necesita saber "¿para qué sucursal?" o "¿qué pedido?", lo que lleva a roles como `manager_branch_1742`: miles de roles, uno por sucursal y puesto.
Tuplas de ReBAC	`restaurant:bella#owner@user:ana`; `branch:bella-soho#parent@restaurant:bella`; `branch:bella-soho#manager@user:ben`; `branch:bella-soho#staff@user:kai`; `order:9812#branch@branch:bella-soho`; `order:9812#courier@user:cara`
Reglas del modelo	`branch.edit_menu` = el gerente, o el propietario del restaurante padre. `order.mark_ready` = el personal o el gerente de la sucursal del pedido. `order.view_pickup` = el repartidor asignado, o el personal de la sucursal del pedido.
Comprobaciones	¿Puede ben editar la carta de bella-soho? Sí (gerente). ¿Y ana? Sí (propietaria del restaurante padre). ¿Puede cara ver la recogida del pedido 9812? Sí, hasta que Despacho borre su tupla de repartidora al completarse el reparto; después, no.
ABAC para el límite de reembolso	La regla de reembolso depende de un atributo de la solicitud, no de una relación: el motor de políticas permite `refund` si el usuario gestiona la sucursal del pedido y el importe es como máximo de £20.
Dónde se aplica	El gateway comprueba la validez y el ámbito del token; los servicios de carta, de pedidos y de despacho piden cada uno al PDP la decisión a nivel de objeto antes de actuar.

Las relaciones responden directamente a las preguntas de auditoría: listar a todos los que pueden editar la carta de una sucursal es una consulta al grafo, no una revisión del código.

Ahora los datos de relaciones también tienen un propietario: Despacho escribe las tuplas de repartidores, el portal de restaurantes escribe las tuplas del personal, y las tuplas obsoletas son un bug de seguridad.

Identidad de carga de trabajo: cuando quien llama es un servicio

Los servicios se autentican entre sí con mucha más frecuencia que las personas inician sesión, y sin embargo las credenciales de servicio suelen ser las más débiles: una clave de API de larga duración en una variable de entorno, compartida por varios servicios y nunca rotada. Si esa clave se filtra a través de un log, un repositorio o una imagen de contenedor, el atacante se convierte en el servicio. La identidad de carga de trabajo sustituye esos secretos por identidades que la plataforma emite para la carga de trabajo en ejecución, después de haber verificado qué carga de trabajo es.

El estándar abierto es **SPIFFE** (Secure Production Identity Framework for Everyone), un proyecto graduado de la Cloud Native Computing Foundation. Nombra cada carga de trabajo con un URI como `spiffe://fooddelivery.example/ns/prod/sa/orders` y emite un documento de corta duración, rotado automáticamente, que lo demuestra, llamado SVID, normalmente un certificado X.509 (o un JWT). Su implementación de referencia, SPIRE, verifica mediante atestación el nodo y la carga de trabajo (por ejemplo, su namespace y su cuenta de servicio de Kubernetes) antes de emitir la identidad. Después, los servicios usan los certificados para el TLS mutuo, de modo que el servicio de pagos sabe que quien llama es de verdad el servicio de pedidos y puede autorizar según la identidad. Las plataformas en la nube ofrecen equivalentes (roles para cargas de trabajo, identidades administradas, federación de identidades de carga de trabajo), y los sistemas de CI como GitHub Actions pueden canjear un token OIDC firmado por credenciales de nube de corta duración, de modo que los pipelines no necesitan claves almacenadas. Cómo se usan estas identidades para construir una red sin confianza implícita es el tema de zero trust.

Acceso administrativo justo a tiempo y de emergencia (break-glass)

Una cuenta de administrador permanente es un objetivo permanente. Con el acceso **justo a tiempo** (just-in-time), un ingeniero no tiene privilegios permanentes, solo elegibilidad: para arreglar una base de datos de producción activa un rol durante un tiempo limitado, da un motivo, supera la autenticación multifactor y, en los roles sensibles, obtiene una aprobación, y cada paso queda registrado. Microsoft Entra Privileged Identity Management es un ejemplo. Las cuentas **de emergencia** (break-glass) cubren el día en que ese proceso falla, por ejemplo porque el proveedor de identidad está caído. La guía de Microsoft recomienda al menos dos cuentas, separadas de las dependencias normales de inicio de sesión, protegidas con autenticación resistente al phishing, con alertas en cada uso y probadas con regularidad. Luego se revisa cada uso, y la vía de emergencia es en sí misma un riesgo que hay que gestionar.

Autorización en los topics de mensajes

La autenticación en el broker no basta: los eventos (Sección 6.4) desencadenan acciones reales, de modo que un servicio que pueda publicar un evento `PaymentCaptured` falsificado puede hacer que se despachen pedidos sin pagar. Concede permisos por topic y por dirección: solo Pagos puede escribir en ese topic y solo los consumidores designados pueden leerlo, lo que también limita quién ve los datos personales que contenga. Apache Kafka lo expresa como listas de control de acceso que indican que un principal puede realizar una operación, como Read o Write, sobre un recurso; con un autorizador configurado, un recurso sin ninguna lista coincidente solo está disponible para los superusuarios, salvo que se establezca `allow.everyone.if.no.acl.found`. Los valores predeterminados varían: en RabbitMQ, los permisos de topic son opcionales. Consulta la documentación de tu broker y mantén estas reglas en el control de versiones.

COMPRUEBA LO QUE HAS ENTENDIDO

▼ **Una API devuelve 403 a quien llama con un token válido. ¿Qué te indica eso?**

Que quien llama se ha autenticado, pero no está autorizado para esta acción u objeto; 401 significaría una autenticación ausente o no válida.

▼ **¿Contra qué protege PKCE en el flujo de código de autorización?**

Contra la interceptación o la inyección del código de autorización: sin el verificador de código, que nunca viaja en las redirecciones del navegador, un código robado no puede canjearse por tokens.

▼ **¿Por qué un servicio no debería aceptar un token de ID de OpenID Connect como token de acceso a una API?**

El token de ID va dirigido al cliente (su audiencia es el ID del cliente) para decirle quién ha iniciado sesión; las API solo deberían aceptar tokens de acceso emitidos para ellas, comprobando la audiencia y el ámbito.

▼ **¿Por qué un rol como manager_branch_1742 sugiere que el modelo de autorización no es el adecuado?**

Codifica una relación (gerente de una sucursal concreta) como un rol, lo que lleva a una explosión de roles; las reglas basadas en relaciones o en atributos la expresan directamente.

▼ **Inténtalo: Elige el flujo de OAuth 2.0 para: la app móvil nativa de un restaurante que usa el personal, y un trabajo backend nocturno que llama a la API de pagos. Además, un desarrollador envía el token de ID a la API de pedidos. Aconseja sobre los tres casos.**

La app móvil actúa para un usuario y no puede guardar un secreto, así que usa el flujo de código de autorización con PKCE. El trabajo nocturno no tiene usuario, así que usa client credentials o, mejor, identidad de carga de trabajo. El desarrollador se equivoca: el token de ID le dice al cliente quién inició sesión, y la API debe recibir el token de acceso, comprobando el emisor, la audiencia y la caducidad. Mantén los tokens de corta duración, y deja que cada servicio siga haciendo sus propias comprobaciones a nivel de objeto.

FUENTES

- RFC 6749: The OAuth 2.0 Authorization Framework: https://www.rfc-editor.org/rfc/rfc6749
- RFC 7636: Proof Key for Code Exchange by OAuth Public Clients: https://www.rfc-editor.org/rfc/rfc7636
- RFC 9700: Best Current Practice for OAuth 2.0 Security: https://www.rfc-editor.org/rfc/rfc9700
- NIST SP 800-63B-4: Authentication and Authenticator Management: https://pages.nist.gov/800-63-4/sp800-63b.html
- Zanzibar: Google's Consistent, Global Authorization System (USENIX ATC 2019): https://research.google/pubs/zanzibar-googles-consistent-global-authorization-system/
- Microsoft Entra Privileged Identity Management: https://learn.microsoft.com/entra/id-governance/privileged-identity-management/pim-configure
- Microsoft: Manage emergency access accounts: https://learn.microsoft.com/en-us/entra/identity/role-based-access-control/security-emergency-access
- Apache Kafka: Authorization and ACLs: https://kafka.apache.org/44/security/authorization-and-acls/
- RabbitMQ: Authentication, authorisation and access control: https://www.rabbitmq.com/docs/access-control

SECCIÓN 8.4 DE 8.8 · AVANZADO · 28 MIN DE LECTURA · ÚLTIMA REVISIÓN: NOVIEMBRE DE 2026

Cifrado, gestión de claves y secretos

ANTES DE LEER ESTO

Esta sección se basa en:

- 0.4 Comunicación en red
- 0.5 Almacenar datos
- 8.3 Identidad y acceso

Términos anteriores que utiliza: TLS y HTTPS, Identidad de carga de trabajo, TLS mutuo (mTLS), Passkey (llave de acceso), JSON Web Token (JWT)

LA IDEA EN BREVE

Piensa en un hotel. Los objetos de valor de los huéspedes van en cajas fuertes en las habitaciones, y las cajas fuertes solo son tan buenas como el control sobre quién tiene los códigos. El hotel no guarda el código de cada huésped en un libro en recepción; la llave maestra está en un armario cerrado con llave que solo puede abrir el director, y cada vez que se abre queda anotado en un registro. Los objetos más valiosos el hotel no los guarda en absoluto: le da al huésped un resguardo numerado y el banco de la esquina custodia el objeto. Proteger los datos funciona igual: cífralos, protege las claves con más cuidado que los datos y, siempre que sea posible, no guardes en absoluto los datos más sensibles.

LO QUE NECESITAS SABER

- Cada capa de cifrado protege frente a actores concretos: TLS frente a la red, el cifrado del almacenamiento frente al robo de soportes y copias de seguridad, el cifrado a nivel de campo frente a los administradores de bases de datos y las copias posteriores; la protección siempre se reduce al control de las claves.
- Los algoritmos simétricos (AES-GCM) cifran grandes volúmenes de datos; la criptografía asimétrica proporciona acuerdo de claves y firmas digitales; los primeros estándares poscuánticos del NIST (FIPS 203 a 205, 2024) convierten la agilidad criptográfica (crypto agility) en una inquietud arquitectónica.
- El hash, los MAC, las firmas, el cifrado y la codificación son herramientas distintas; las contraseñas necesitan funciones de hash de contraseñas lentas y con sal (salt), como Argon2id con los parámetros actuales de OWASP o, mejor aún, passkeys a través de un proveedor de identidad.

- El cifrado de sobre envuelve claves de datos por registro bajo una clave de cifrado de claves guardada en un KMS respaldado por HSM, lo que abarata la rotación, limita el radio de impacto y permite el crypto-shredding, a costa de una dependencia crítica.
- Los secretos deben estar en una bóveda (vault), preferiblemente como credenciales dinámicas de corta duración o sustituidos por la identidad de carga de trabajo; un secreto filtrado debe revocarse primero y limpiarse después.
- La tokenización a través de un proveedor de pagos mantiene los números de tarjeta fuera de la plataforma y reduce el alcance de PCI DSS v4.0.1; un esquema de clasificación de datos decide dónde compensa el costo del cifrado a nivel de campo.

Qué protege el cifrado y qué no

El cifrado se trata a menudo como una casilla que marcar: "los datos están cifrados en tránsito y en reposo". La frase esconde la pregunta que importa, que es **frente a quién**. Cada capa de cifrado protege los datos frente a unos actores y no frente a otros, y en todos los casos la protección es exactamente tan fuerte como el control sobre quién puede usar la clave. El cifrado no elimina el problema del control de acceso; lo traslada de los datos a las claves, donde normalmente es más fácil gestionarlo, auditarlo y revocarlo.

El cifrado **en tránsito** protege los datos que se mueven por una red para que no se lean ni se alteren por el camino. En la web eso significa TLS, presentado en el Capítulo 0; entre servicios significa TLS o TLS mutuo, que se tratan junto con zero trust. El cifrado **en reposo** protege los datos almacenados, pero "en reposo" no es una sola cosa: la tabla muestra las capas habituales en la plataforma de reparto de comida.

Capas de cifrado y las amenazas que aborda cada una

Capa	Protege frente a	No protege frente a
TLS en tránsito	Escuchas y manipulaciones en la red, incluso dentro del centro de datos.	Un servicio o un endpoint comprometido, que ve los datos en texto en claro.
Cifrado de disco o de volumen (por defecto en las principales nubes)	Discos robados o desechados, hardware devuelto al proveedor.	Cualquiera que llegue a los datos a través del sistema operativo o la base de datos en ejecución.
Cifrado de la base de datos o del servicio de almacenamiento (por ejemplo, el cifrado de datos transparente)	Copias de archivos de datos, snapshots y copias de seguridad hechas en la capa de almacenamiento.	Cualquier cuenta de la base de datos que pueda ejecutar consultas, incluida la de la propia aplicación y la de un atacante que use inyección SQL.
Cifrado a nivel de aplicación (a nivel de campo)	Los administradores de bases de datos, las cuentas con permisos demasiado amplios, las copias para análisis y las copias de seguridad filtradas, porque solo contienen texto cifrado.	La propia aplicación si está comprometida, ya que puede usar la clave; la base de datos ya no puede buscar ni ordenar por ese campo.
Cifrado de extremo a extremo	Todos los intermediarios, incluido el operador de la plataforma.	Nada intermedio, pero entonces la plataforma no puede hacer nada útil con los datos.

Al leer la tabla de arriba abajo, cada capa protege frente a más actores y cuesta más funcionalidad. Un buen diseño usa las capas baratas y transparentes en todas partes (hoy TLS y el cifrado del almacenamiento son casi gratuitos) y las caras de forma selectiva, para los datos cuya exposición haría más daño. Decidir qué datos son esos es tarea de la clasificación de datos, que se trata más adelante en esta sección junto con la tokenización y un ejemplo práctico.

Criptografía simétrica y asimétrica

El cifrado simétrico usa una única clave secreta para cifrar y descifrar. Es lo bastante rápido para grandes volúmenes de datos, y el algoritmo estándar es AES (FIPS 197, 2001) con claves de 128 o 256 bits. Los diseños modernos usan **cifrado autenticado**, que detecta la manipulación además de ocultar el contenido: AES en modo Galois/Counter (AES-GCM) o ChaCha20-Poly1305 (RFC 8439). Los modos autenticados necesitan un nonce único para cada mensaje cifrado con la misma clave; reutilizar un nonce con AES-GCM puede revelar el texto en claro y permitir falsificaciones, que es una de las muchas razones para usar bibliotecas bien probadas en lugar de ensamblar las primitivas por tu cuenta.

La criptografía asimétrica usa un par de claves: una clave pública que cualquiera puede tener y una clave privada que se mantiene en secreto. Permite dos cosas que la criptografía simétrica no puede hacer. El **acuerdo de claves** permite que dos partes que nunca se han visto establezcan una clave simétrica compartida a través de una red abierta (Diffie-Hellman de curva elíptica en TLS 1.3). Las **firmas digitales** (RSA, ECDSA, Ed25519) permiten que cualquiera que tenga la clave pública verifique que los datos proceden del titular de la clave privada y no se han modificado, lo que proporciona integridad, autenticidad y una base para el no repudio. Las operaciones asimétricas son lentas, así que los protocolos reales son híbridos: TLS usa la criptografía asimétrica para autenticar al servidor y acordar las claves, y después el cifrado simétrico para los datos.

Una computadora cuántica grande rompería los algoritmos RSA y de curva elíptica actuales, aunque no AES con claves suficientemente largas. En agosto de 2024, el NIST publicó sus primeros estándares poscuánticos: FIPS 203 (ML-KEM, para el establecimiento de claves), FIPS 204 (ML-DSA) y FIPS 205 (SLH-DSA), ambos para firmas. Los datos que deben seguir siendo confidenciales durante muchos años ya están en riesgo hoy por los atacantes que graban ahora el tráfico cifrado para descifrarlo más adelante, así que la respuesta arquitectónica es la **agilidad criptográfica** (crypto agility): mantener un inventario de dónde se usa cada algoritmo, evitar codificarlos de forma fija y preferir bibliotecas, protocolos y proveedores que puedan cambiar de algoritmo, como han empezado a hacer los principales navegadores y bibliotecas TLS con el intercambio de claves poscuántico híbrido.

El hash, los MAC, las firmas y el cifrado son herramientas distintas

Confundir estas primitivas es una fuente habitual de defectos de diseño, como "cifrar" las contraseñas para poder descifrarlas, o confiar en un hash simple para demostrar que un webhook procede de un proveedor de pagos. La tabla las pone una al lado de otra.

Elegir la primitiva adecuada

Herramienta	¿Reversible?	Clave	Úsala para	Ejemplo
Codificación (Base64, hex)	Sí, por cualquiera	Ninguna	Representar bytes como texto. No proporciona ninguna seguridad.	La carga útil de un JWT
Hash criptográfico (SHA-256)	No	Ninguna	Huellas digitales, integridad de los archivos frente a cambios accidentales, direccionamiento por contenido. MD5 y SHA-1 están rotos en cuanto a la resistencia a colisiones.	Suma de comprobación de una versión descargada
Código de autenticación de mensajes (HMAC-SHA-256)	No	Secreto compartido	Demostrar que un mensaje procede de alguien que tiene la clave y que no se ha modificado.	Verificar el webhook de un proveedor de pagos
Firma digital (Ed25519, ECDSA)	No (se verifica)	La clave privada firma y la pública verifica	Prueba de origen que cualquiera puede comprobar sin poder falsificarla.	Tokens de acceso JWT, imágenes de contenedor firmadas
Cifrado (AES-GCM)	Sí, con la clave	Secreta (o un par de claves, en los esquemas híbridos)	Confidencialidad, más integridad en los modos autenticados.	El documento de identidad de un repartidor almacenado
Hash de contraseñas (Argon2id)	No, y deliberadamente lento	Sal por contraseña, pimienta opcional	Guardar contraseñas para que una base de datos robada resista los intentos de adivinarlas.	El almacén de credenciales del proveedor de identidad

Almacenamiento de contraseñas

Un hash de uso general como SHA-256 está diseñado para ser rápido, que es justo lo contrario de lo que necesitan las contraseñas: los procesadores gráficos pueden probar miles de millones de intentos por segundo contra una tabla robada de hashes rápidos. El hash de contraseñas usa funciones deliberadamente lentas y, idealmente, exigentes en memoria

(memory-hard), para que cada intento le cueste al atacante tiempo y memoria reales, con una **sal** (salt) aleatoria y única por contraseña, para que contraseñas idénticas produzcan hashes distintos y las tablas precalculadas no sirvan de nada.

La OWASP Password Storage Cheat Sheet da las recomendaciones actuales por orden de preferencia. **Argon2id** (una variante de Argon2, ganador de la Password Hashing Competition de 2015), con un mínimo de 19 MiB de memoria, dos iteraciones y un grado de paralelismo (o un compromiso equivalente, como 46 MiB con una iteración). **scrypt** donde Argon2id no esté disponible, con un parámetro de costo de 2^{17} (128 MiB), un tamaño de bloque de 8 y un paralelismo de 1. **bcrypt** solo para sistemas heredados, con un factor de trabajo de al menos 10, recordando que la mayoría de las implementaciones ignoran la entrada que supera los 72 bytes. Donde se exija el cumplimiento de FIPS 140, **PBKDF2** con HMAC-SHA-256 y 600 000 iteraciones. Estas cifras suben a medida que mejora el hardware, así que consulta la cheat sheet vigente en lugar de copiar números en un documento de diseño.

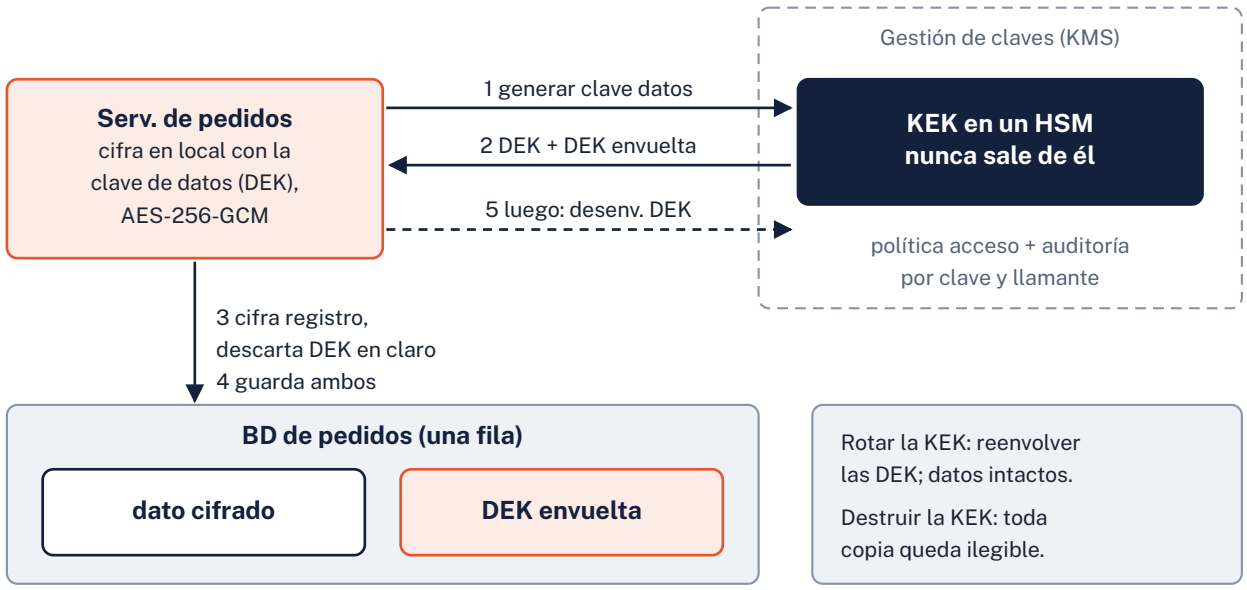
De aquí se derivan tres puntos de diseño. Guarda el algoritmo y los parámetros con cada hash (los formatos codificados estándar lo hacen), para poder aumentar los parámetros más adelante: cuando un usuario inicia sesión correctamente, verifica con el hash antiguo y vuelve a calcular el hash de inmediato con los parámetros nuevos. Considera una **pimienta** (pepper), un secreto que se añade a cada contraseña antes de calcular el hash y que se guarda en una bóveda o un HSM en lugar de en la base de datos, lo que frustra a un atacante que solo robe la base de datos. Y lo más importante: pregúntate si deberías guardar contraseñas en absoluto: delegar la autenticación en un proveedor de identidad y ofrecer passkeys elimina de la mayoría de tus servicios el secreto más atacado.

Gestión de claves y cifrado de sobre

Las claves tienen un ciclo de vida, descrito en NIST SP 800-57 Part 1: generación (a partir de una fuente aleatoria adecuada), almacenamiento, distribución, uso, rotación al final de un criptoperiodo definido, revocación si se comprometen y destrucción. Hacer todo eso de forma segura a mano no es práctico, y por eso las arquitecturas se apoyan en un servicio de gestión de claves (KMS) como AWS KMS, Azure Key Vault o Google Cloud KMS, o en un sistema autogestionado como HashiCorp Vault. Un KMS guarda las claves maestras dentro de módulos de seguridad de hardware (HSM), dispositivos resistentes a la manipulación validados según FIPS 140, y realiza operaciones con esas claves cuando se le solicitan, sujetas a una política de acceso, de modo que las aplicaciones nunca ven las claves maestras y cada uso queda registrado.

Enviar cada registro al KMS para cifrarlo sería lento y caro, y las API de KMS limitan el tamaño de las solicitudes. El cifrado de sobre resuelve esto. La aplicación le pide al KMS una **clave de cifrado de datos** (DEK) nueva; el KMS la devuelve dos veces, en texto en claro y envuelta

(cifrada) con una **clave de cifrado de claves** (KEK) que nunca sale del HSM. La aplicación cifra los datos localmente con la DEK en claro, borra la DEK de la memoria y guarda el texto cifrado junto con la DEK envuelta. Para descifrar, envía solo la pequeña DEK envuelta al KMS, que comprueba el permiso de quien llama, registra la solicitud y devuelve la DEK.



Cifrado de sobre: los datos voluminosos nunca van al KMS, y la clave maestra nunca sale de él.

```
import os
from cryptography.hazmat.primitives.ciphers.aead import AESGCM

def encrypt_field(plaintext: bytes, kms, kek_id: str, context: bytes) → dict:
    dek = AESGCM.generate_key(bit_length=256)           # fresh data key
    nonce = os.urandom(12)                              # never reuse with this key
    ct = AESGCM(dek).encrypt(nonce, plaintext, context)  # context binds it to its row
    wrapped = kms.encrypt(key_id=kek_id, plaintext=dek)  # KEK stays inside the KMS
    return {"ct": ct, "nonce": nonce, "wrapped_dek": wrapped, "kek": kek_id}

def decrypt_field(rec: dict, kms, context: bytes) → bytes:
    dek = kms.decrypt(key_id=rec["kek"], ciphertext=rec["wrapped_dek"])  # checked,
logged
    return AESGCM(dek).decrypt(rec["nonce"], rec["ct"], context)
```

Cifrado de sobre de un campo con la biblioteca cryptography de Python; kms representa cualquier cliente de KMS (las API reales de KMS combinan los dos primeros pasos en una sola llamada generate-data-key).

El cifrado de sobre tiene varias ventajas arquitectónicas. **La rotación es barata**: rotar una KEK crea una nueva versión de la clave para las nuevas operaciones de envoltura, mientras que las versiones antiguas siguen disponibles para descifrar; las DEK existentes pueden volver a envolverse sin tocar los datos, y solo hace falta volver a cifrarlo todo si una DEK puede haber

quedado expuesta. **El radio de impacto es limitado**: con una DEK por registro, archivo o partición, una DEK filtrada expone poco. La **separación de funciones** se vuelve posible: el servicio de pedidos puede usar su KEK para desenvolver, pero no cambiar su política ni borrarla, mientras que los administradores de seguridad pueden gestionar las claves, pero no descifrar los datos. Las **jerarquías de claves** se corresponden con la organización: una KEK por servicio y clase de datos, o por cliente (tenant), de modo que una gran cadena de restaurantes podría incluso aportar y controlar su propia clave. Y destruir una clave es una herramienta poderosa: el crypto-shredding hace ilegible cada copia de los datos cifrados con ella, incluidas las copias de seguridad y los logs de eventos inmutables, lo que importa para el supresión según el RGPD (GDPR) y para los sistemas basados en event sourcing.

El precio es una nueva dependencia crítica. Si el KMS no está disponible, los datos cifrados no pueden leerse; si una clave se borra por error, los datos se pierden para siempre. Por eso los KMS en la nube imponen un periodo de espera antes del borrado, y los arquitectos deberían tratar el borrado de claves y los cambios en las políticas de claves como operaciones privilegiadas, registradas y de dos personas, guardar en caché brevemente las DEK desenvueltas donde importe la latencia de las llamadas al KMS e incluir el KMS en los planes de recuperación ante desastres.

SI ESTO ES NUEVO PARA TI

Paso a paso: una clave que cierra otras claves

La idea que requiere más atención aquí es el cifrado de sobre: los datos se cierran con su propia clave de datos, y esa clave se cierra a su vez con una clave maestra que nunca sale del servicio de gestión de claves (KMS). Sigue los datos bancarios para los pagos a un restaurante al entrar y salir del almacenamiento.

- Para guardar los nuevos datos bancarios de un restaurante, el servicio de pagos le pide al KMS una clave de datos nueva. El KMS la devuelve dos veces: en claro y envuelta, es decir, cerrada con la clave maestra de pagos, que permanece dentro del hardware resistente a la manipulación del KMS.
- El servicio cierra los datos con la clave de datos en claro en su propia máquina, borra esa clave de la memoria y guarda los datos cerrados junto con la clave envuelta. Los datos nunca viajan al KMS.
- Quien copie esa fila tendrá unos datos cerrados y una clave cerrada, inútiles sin permiso para usar la clave maestra.
- Cuando el proceso de pagos a restaurantes necesita los datos, envía solo la pequeña clave envuelta al KMS, que comprueba que ese proceso puede usar la clave maestra, registra la solicitud y devuelve la clave de datos en claro.
- De ello se derivan dos ventajas. Rotar la clave maestra no tiene por qué tocar los datos guardados: basta con volver a envolver las pequeñas claves de datos. Y destruir una clave hace ilegible cada copia de los datos cifrados con ella, copias de seguridad incluidas: el crypto-shredding.
- El precio es una nueva dependencia crítica. Si el KMS no está disponible, los datos no pueden leerse; si una clave se borra por error, los datos se pierden para siempre.
- **La lección:** mantén los datos voluminosos en local y las claves centralizadas. Cada registro puede tener su propia clave de datos, mientras que la clave maestra permanece dentro del KMS, donde cada uso se comprueba y se registra.

Gestión de secretos

Un secreto es cualquier credencial que da acceso: contraseñas de bases de datos, claves de API del proveedor de pagos, claves privadas, claves de firma, secretos de cliente de OAuth, secretos de webhooks. La gestión de secretos parte de una regla: **los secretos nunca viven en el código fuente**, ni en imágenes de contenedor, ni en archivos de configuración confirmados en repositorios, porque estos se copian, se guardan en caché, se bifurcan y los leen muchas más personas y sistemas de los que deberían acceder al secreto. Una vez que está en el historial de un repositorio, un secreto permanece ahí aunque se borre la línea.

Los secretos deben estar en una bóveda (vault) (HashiCorp Vault, AWS Secrets Manager, Azure Key Vault, Google Secret Manager) que los guarde cifrados, los entregue solo a cargas de trabajo autenticadas según la política, registre cada acceso y admita la rotación. Mejor que rotar secretos de larga duración es no tenerlos: una bóveda puede emitir **secretos dinámicos**, como una cuenta de base de datos creada para una instancia de servicio con una vigencia (lease) de una hora, y la identidad de carga de trabajo permite que los servicios se autentiquen ante las API de la nube y entre sí sin ningún secreto almacenado. Los secretos inyectados en tiempo de ejecución deberían leerse de archivos o de la API de la bóveda en lugar de variables de entorno visibles para todo el mundo, que tienden a acabar en volcados de memoria por fallos y en logs de depuración.

La prevención necesita automatización, porque las personas se equivocan. Los hooks de pre-commit y los escáneres del pipeline (como gitleaks y detect-secrets) detectan los secretos antes de que se suban; los servicios de alojamiento de repositorios buscan formatos de secretos conocidos, y la protección de push (push protection) de GitHub bloquea los push que los contienen. Los atacantes escanean los repositorios públicos a los pocos minutos de cada push. Las consecuencias están bien documentadas: en 2016, unos atacantes que lograron acceder a un repositorio privado de GitHub usado por ingenieros de Uber encontraron allí una clave de acceso al almacenamiento en la nube de Uber y descargaron datos personales de unos 57 millones de pasajeros y conductores.

Lista de comprobación: cuando se ha filtrado un secreto

Trata como comprometido cualquier secreto que haya llegado a un repositorio, un log, un ticket o un chat, sea o no privado el repositorio.

- [ ] **Revoca o rota el secreto de inmediato**
  Haz esto primero. Borrar el commit o el mensaje no sirve de nada: los clones, las bifurcaciones (forks), las cachés y los escáneres ya lo tienen.
- [ ] **Determina a qué podía acceder**
  Enumera los sistemas, los datos y los permisos que concedía el secreto, y cualquier otro secreto al que se pueda llegar desde allí.
- [ ] **Revisa el uso desde la exposición**
  Revisa los registros de acceso del proveedor y de la bóveda en busca de usos desde identidades, direcciones u horas inesperadas desde que apareció el secreto por primera vez.
- [ ] **Contén y limpia**
  Rota las credenciales dependientes, invalida las sesiones y los tokens emitidos con él y, después, elimina el secreto del historial y de las cachés (después de rotarlo, no en lugar de rotarlo).
- [ ] **Decide sobre la notificación**
  Si es posible que se haya accedido a datos personales, inicia la evaluación de la brecha; según el RGPD, el plazo para notificar a la autoridad de control empieza a contar cuando tienes conocimiento de ella.
- [ ] **Corrige la causa**
  Traslada el secreto a una bóveda, sustitúyelo por la identidad de carga de trabajo o por un secreto dinámico, y activa el escaneo de secretos y la protección de push para que el mismo error quede bloqueado la próxima vez.

Tokenización y reducción del alcance de PCI DSS

La tokenización sustituye un valor sensible por un token que no tiene ninguna relación matemática con él; el valor real se guarda en una bóveda aparte, muy protegida, y solo la bóveda puede volver a asociar el token con su valor. A diferencia del cifrado, un token robado no puede revertirse con una clave, porque no hay clave, solo una consulta en un sistema que el atacante no controla.

Su uso más importante son los pagos con tarjeta. El Payment Card Industry Data Security Standard (PCI DSS, actualmente en la versión 4.0.1, publicada en junio de 2024) se aplica a todo componente del sistema que almacene, procese o transmita datos del titular de la tarjeta, o que esté conectado a esos componentes o pueda afectar a su seguridad. Una plataforma que deja pasar los números de tarjeta por sus servidores mete en el alcance su API gateway, sus servicios, sus logs, sus redes y sus administradores, con cientos de requisitos y una evaluación anual. Una plataforma que usa los campos de pago alojados o la redirección del proveedor de pagos, de modo que los números de tarjeta van directamente del dispositivo del cliente al proveedor, solo recibe un token y quizá los cuatro últimos dígitos, y su alcance de PCI DSS se reduce drásticamente: un comercio más pequeño puede optar al cuestionario de autoevaluación más breve, e incluso los comercios más grandes, que siguen necesitando una evaluación anual por parte de un evaluador cualificado, tienen muchos menos sistemas que evaluar. Este es el diseño del modelo de amenazas del proceso de pago: los datos de tarjeta más baratos de proteger son los que nunca guardas. Cifrar tú mismo los números de tarjeta no consigue la misma reducción, porque quien pueda descifrarlos sigue dentro del alcance.

La misma idea sirve para otros identificadores sensibles. Un servicio interno de tokenización puede sustituir los números de la seguridad social o los números de pasaporte por tokens en toda la plataforma, de modo que solo un servicio pequeño y muy protegido vea alguna vez los valores reales.

Clasificación de datos

Las decisiones sobre cifrado, tokenización, logging, conservación y acceso dependen de saber qué datos son sensibles y en qué medida. Un esquema de **clasificación de datos** establece unos pocos niveles con reglas de tratamiento asociadas, para que los equipos no decidan cada caso desde cero. Lo habitual son cuatro niveles; lo que importa es que cada conjunto de datos del catálogo de datos o del esquema de la plataforma lleve un nivel, y que las reglas se apliquen mediante plantillas y comprobaciones, no de memoria.

Un esquema de clasificación de datos para la plataforma de reparto de comida

Nivel	Ejemplos	Reglas de tratamiento
Público	Cartas, precios, horarios de apertura de los restaurantes	Solo protección de la integridad; puede guardarse en caché en cualquier lugar.
Interno	Cifras de ventas agregadas, configuración de servicios sin secretos	Solo personal y servicios; no en logs públicos.
Confidencial	Nombres, direcciones, números de teléfono e historiales de pedidos de los clientes, ubicaciones de los repartidores	TLS y cifrado del almacenamiento; acceso con mínimo privilegio; enmascarado en las herramientas de soporte; minimizado en los análisis; límites de conservación.
Restringido	Datos bancarios para los pagos a restaurantes, documentos de identidad de los repartidores, secretos, tokens de pago	Todo lo anterior, más cifrado a nivel de campo con una clave dedicada, acceso solo por servicios designados y registro de cada acceso.

Cifrado a nivel de campo

El cifrado a nivel de campo, la capa a nivel de aplicación de la primera tabla, cifra campos individuales antes de escribirlos, normalmente con cifrado de sobre y una KEK por clase de datos. Su fortaleza es que todo lo que hay después de la aplicación solo contiene texto cifrado: los administradores de bases de datos, las réplicas de lectura, las copias de seguridad y el almacén de datos (data warehouse) alimentado por captura de datos de cambios. Una brecha en cualquiera de ellos no expone nada legible, y el acceso al texto en claro se reduce a los servicios que pueden llamar al KMS con esa clave, y cada llamada queda registrada.

Sus costos son reales. La base de datos ya no puede indexar, buscar, ordenar ni hacer joins por el campo cifrado. Las búsquedas por igualdad pueden recuperarse con un **índice ciego** (blind index), una columna aparte que guarda un hash con clave (HMAC) del valor normalizado, a cambio de revelar qué filas comparten un valor. El cifrado determinista tiene la misma fuga, mientras que las consultas por rango necesitan esquemas especializados o, simplemente, no deberían hacerse sobre esos campos. Algunas bases de datos ofrecen cifrado a nivel de campo del lado del cliente o cifrado consultable, que empaquetan estas técnicas. Por estos costos, el cifrado a nivel de campo debería reservarse para los datos restringidos, cuyos campos rara vez se consultan de todos modos.

EJEMPLO PRÁCTICO

Ejemplo práctico: qué cifrar, dónde y con qué claves

El arquitecto de la plataforma de reparto de comida revisa los principales conjuntos de datos con el esquema de clasificación. Todos los servicios ya usan TLS (TLS mutuo internamente) y almacenamiento cifrado con una clave de KMS por servicio, así que la pregunta es qué necesita algo más.

Números de tarjeta	Nunca se guardan. Los campos de pago alojados los envían al proveedor de pagos; Pagos guarda el token del proveedor, la marca de la tarjeta y los cuatro últimos dígitos. El alcance de PCI DSS se reduce a la integración del proceso de pago con el proveedor.
Contraseñas de los clientes	Los servicios de producto no las guardan. El proveedor de identidad ofrece passkeys; las contraseñas que quedan se procesan con hash Argon2id con los parámetros de OWASP, con una pimienta guardada en un HSM.
Direcciones y números de teléfono (confidencial)	Cifrado del almacenamiento con la KEK de Pedidos; cuentas con mínimo privilegio; enmascarados en la herramienta de soporte; el almacén de datos solo recibe distritos postales, no direcciones completas.
Datos bancarios para los pagos a restaurantes (restringido)	Cifrado de sobre a nivel de campo con una KEK dedicada de Pagos; solo el proceso de pagos a restaurantes puede desenvolver; los cambios exigen volver a autenticarse y se notifican al propietario del restaurante, lo que protege tanto la integridad como la confidencialidad.
Documentos de identidad de los repartidores (restringido)	Se guardan como objetos, cada uno con su propia DEK envuelta con una KEK de incorporación; solo el servicio de verificación puede descifrar; cada acceso se registra y se revisa mensualmente; se borran al terminar el periodo legal de conservación.
Datos personales en los eventos y en el log de eventos	Los eventos llevan ID de cliente en lugar de direcciones; los pocos campos personales que deben viajar se cifran con una clave por cliente, de modo que destruir la clave los borra del log conservado.
Copias de seguridad	Cifradas con una KEK aparte guardada en una cuenta de nube aparte, para que comprometer las credenciales de producción no exponga las copias de seguridad; las restauraciones se prueban trimestralmente.
Secretos de los servicios	En la bóveda; las bases de datos usan credenciales dinámicas con vigencias de una hora; la clave de API del proveedor de pagos solo puede leerla la identidad de carga de trabajo de Pagos.

Unas diez KEK, cada una propiedad de un servicio y rotada automáticamente cada año, cubren toda la plataforma; la palanca está en las decisiones, no en el número de claves.

Dos decisiones eliminan datos en lugar de protegerlos (los números de tarjeta y las direcciones completas en los análisis), y esa es siempre la opción más fuerte.

El diseño se documenta como un ADR por cada conjunto de datos restringido y como el concepto de protección de datos en la descripción de arquitectura.

COMPRUEBA LO QUE HAS ENTENDIDO

▼ **¿Frente a qué actores no protege el cifrado transparente de la base de datos?**

Frente a cualquiera que pueda consultar la base de datos con una cuenta válida, incluidos la aplicación, un atacante que use inyección SQL y los administradores de bases de datos, porque la base de datos les descifra los datos.

▼ **¿Por qué es barato rotar una clave de cifrado de claves en el cifrado de sobre?**

Solo hay que volver a envolver las pequeñas claves de datos envueltas con la nueva versión de la clave; los datos voluminosos siguen cifrados con sus claves de datos, que no cambian.

▼ **Un desarrollador propone guardar las contraseñas cifradas con AES para que el personal de soporte pueda decir a los usuarios su contraseña. ¿Qué tiene de malo?**

Las contraseñas deben guardarse con un hash de contraseñas unidireccional, lento y con sal, para que nadie, ni el personal ni quien robe la clave, pueda recuperarlas; la recuperación debe restablecer la contraseña, no revelarla.

▼ **¿Por qué usar los campos alojados de un proveedor de pagos reduce el alcance de PCI DSS más que cifrar tú mismo los números de tarjeta?**

Con los campos alojados, los números de tarjeta nunca llegan a los sistemas de la plataforma, así que no están en el alcance; los números de tarjeta cifrados que la plataforma puede descifrar mantienen sus sistemas dentro del alcance.

▼ **Inténtalo: La plataforma ya usa TLS y almacenamiento cifrado. Decide qué más hacer con los números de tarjeta, las contraseñas y las notas sobre alergias de los clientes. Para cada uno, indica la protección y la amenaza que aborda. Supón que la clasificación considera sensibles las notas sobre alergias.**

Números de tarjeta: nunca los guardes; usa campos de pago alojados y el token del proveedor, lo que reduce el alcance de PCI DSS. Contraseñas: sin cifrado, sino un hash lento con sal como Argon2id, o passkeys a través de un proveedor de identidad, de modo que una tabla robada sirva de poco. Notas sobre alergias: cifrado a nivel de campo con claves de sobre en un KMS, que protege frente a los administradores y las copias posteriores. Cuesta capacidad de búsqueda y añade una dependencia del KMS, así que resérvalo para los campos sensibles.

FUENTES

- OWASP Password Storage Cheat Sheet: https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html
- OWASP Secrets Management Cheat Sheet: https://cheatsheetseries.owasp.org/cheatsheets/Secrets_Management_Cheat_Sheet.html
- NIST SP 800-57 Part 1 Rev. 5: Recommendation for Key Management: https://csrc.nist.gov/pubs/sp/800/57/pt1/r5/final
- Google Cloud KMS: Envelope encryption: https://cloud.google.com/kms/docs/envelope-encryption
- PCI Security Standards Council: https://www.pcisecuritystandards.org/

SECCIÓN 8.5 DE 8.8 · AVANZADO · 34 MIN DE LECTURA · ÚLTIMA REVISIÓN: NOVIEMBRE DE 2026

Comunicación segura y zero trust

ANTES DE LEER ESTO

Esta sección se basa en:

- 0.4 Comunicación en red
- 8.3 Identidad y acceso
- 8.4 Protección de datos

Términos anteriores que utiliza: TLS y HTTPS, OAuth 2.0, Identidad de carga de trabajo, Falacias de la computación distribuida, API gateway (puerta de enlace de API)

LA IDEA EN BREVE

Imagina un hotel donde la recepción comprueba tu identidad una sola vez y luego te da una llave que abre todas las habitaciones del edificio. Si un ladrón logra pasar la recepción, todas las habitaciones quedan abiertas. Un hotel mejor te da una tarjeta que abre solo tu habitación, solo durante las noches que reservaste, y registra cada puerta que abre. Zero trust (confianza cero) aplica la segunda idea al software: ninguna solicitud se considera de confianza por venir de dentro de la red de la empresa. Cada persona y cada programa debe demostrar quién es y tener permiso para hacer esa acción concreta, cada vez.

LO QUE NECESITAS SABER

- El modelo perimetral confía en todo lo que está dentro de la red, así que una sola credencial robada o una máquina comprometida permiten el movimiento lateral; además, la nube, los dispositivos móviles y el trabajo remoto han eliminado cualquier perímetro único que defender.
- NIST SP 800-207 (agosto de 2020) define zero trust como no conceder ninguna confianza implícita basada en la ubicación en la red ni en la propiedad, con siete principios que incluyen proteger toda la comunicación, el acceso por sesión y una política dinámica aplicada de forma estricta.
- Un punto de aplicación de políticas situado delante de cada recurso consulta a un punto de decisión de políticas (motor de políticas más administrador de políticas), que decide usando la identidad, la postura del dispositivo o de la carga de trabajo, los datos de amenazas y la política.

- BeyondCorp de Google (2014) aplicó zero trust al acceso del personal con un inventario de dispositivos, un proxy de acceso y niveles de confianza, y eliminó la red privilegiada de la oficina y la VPN.
- El TLS mutuo da a cada llamada entre servicios cifrado y una identidad verificada en ambos extremos; SPIFFE estandariza identidades de carga de trabajo de corta duración, y las service meshes automatizan mTLS y la política por llamada con un costo operativo real.
- Los certificados deben emitirse y renovarse automáticamente (ACME, RFC 8555), sobre todo ahora que la vida útil de los certificados TLS públicos baja a 200 días en 2026, 100 días en 2027 y 47 días en 2029.
- Los API gateways autentican tokens, limitan la tasa de solicitudes y validan la entrada, pero la autorización a nivel de objeto debe seguir haciéndose en el servicio que es propietario de los datos.

Por qué dejó de funcionar el perímetro de red

Durante décadas, la mayoría de las organizaciones protegieron sus sistemas como un castillo con foso. Un firewall separaba el interior de confianza de un internet que no lo era, el personal se conectaba al interior mediante una red privada virtual (VPN) y todo lo que ya estaba dentro se trataba como amigo. Los servicios del centro de datos se llamaban entre sí por HTTP sin cifrar y sin autenticación, porque la propia red era la credencial. El Capítulo 6 incluyó “la red es segura” entre las falacias de la computación distribuida; el modelo perimetral es esa falacia convertida en arquitectura.

Tres cosas lo rompieron. Primero, el interior se volvió demasiado grande para defenderlo: los portátiles viajan, el personal trabaja desde casa, los socios se conectan, y un solo correo electrónico de phishing o una contraseña de VPN robada pone a un atacante dentro del foso. Una vez allí, el atacante se desplaza de máquina en máquina, lo que se llama movimiento lateral. En la brecha de 2013 en el minorista estadounidense Target, los atacantes usaron primero credenciales de red robadas a un contratista de calefacción y ventilación y desde ahí llegaron a los terminales de pago, llevándose datos de unos 40 millones de tarjetas. Segundo, el interior se disolvió: las cargas de trabajo se ejecutan ahora en varias cuentas de nube, productos SaaS y API de terceros, así que no hay un único muro donde poner un firewall. Tercero, el interior se volvió dinámico: los contenedores aparecen y desaparecen cada minuto y las direcciones IP se reutilizan, de modo que reglas escritas como *permitir que 10.0.4.0/24 llegue a 10.0.7.12* no describen nada estable.

La idea de abandonar el perímetro es más antigua que el nombre. El Jericho Forum, un grupo de responsables de seguridad de grandes empresas, defendió la *desperimetrización* desde 2004; John Kindervag, entonces analista de Forrester Research, acuñó el término *zero trust* hacia

2010; y los artículos de Google sobre BeyondCorp a partir de 2014 mostraron que funcionaba a gran escala. El lema es *nunca confíes, verifica siempre*: ninguna solicitud es de confianza por su procedencia.

SI ESTO ES NUEVO PARA TI

Si esto es nuevo para ti

Piensa en un edificio de oficinas. El enfoque antiguo comprueba tu tarjeta en la puerta principal y luego te deja entrar en todas las salas. Zero trust comprueba tu tarjeta, y pregunta si deberías estar ahí en ese momento, en cada puerta que abres. El edificio ya no tiene un único interior de confianza, así que un ladrón que se cuela detrás de alguien por la puerta principal sigue sin poder entrar en la oficina de finanzas.

El resto de esta sección usa algunos términos de secciones anteriores: TLS y certificados (Sección 0.4), tokens, OAuth 2.0 e identidad de carga de trabajo (Sección 8.3) y cifrado y claves (Sección 8.4).

Zero trust según la definición del NIST

El texto de referencia es la publicación especial 800-207 del NIST, *Zero Trust Architecture*, publicada en agosto de 2020 (Rose, Borchert, Mitchell y Connelly). Describe zero trust (confianza cero) como un conjunto de principios que trasladan las defensas desde perímetros estáticos basados en la red hacia los usuarios, los activos y los recursos, y una arquitectura zero trust como el plan de una organización para aplicarlos. Su afirmación central es que no se concede ninguna confianza implícita a un activo o a una cuenta de usuario basándose solo en su ubicación física o en la red, ni en quién es su propietario. La autenticación y la autorización son funciones independientes que se realizan antes de establecer una sesión con un recurso.

La publicación enumera siete principios. Vale la pena conocerlos en sus propios términos, porque muchos productos dicen ser zero trust aunque solo cumplen uno o dos.

Los siete principios de NIST SP 800-207, con la plataforma de reparto de comida

Principio (parafraseado)	Qué significa	Ejemplo de reparto de comida
1. Todas las fuentes de datos y servicios de cómputo son recursos	Un portátil, un contenedor, una herramienta SaaS y una base de datos son cosas que proteger, no solo los servidores del centro de datos.	El backend de la app del repartidor, la API de la tableta del restaurante y el bucket de analítica son cada uno un recurso con su propia política.
2. Toda la comunicación se protege, sea cual sea la ubicación en la red	El tráfico dentro del centro de datos recibe el mismo cifrado y la misma autenticación que el tráfico por internet.	Pedidos llama a Pagos mediante TLS mutuo aunque ambos se ejecuten en el mismo clúster.
3. El acceso se concede por sesión	La confianza se evalúa para cada sesión con el mínimo privilegio necesario, y conceder un recurso no concede nada más.	La sesión de un agente de soporte puede leer el historial de pedidos de un cliente; no puede llegar además a la API de reembolsos.
4. El acceso lo decide una política dinámica	La decisión usa la identidad, el estado del dispositivo o de la carga de trabajo y otros atributos como la hora, la ubicación y el comportamiento.	Un reembolso por encima de cierto importe desde un portátil no gestionado se rechaza; la misma solicitud desde un dispositivo gestionado con una MFA reciente se permite.
5. La empresa supervisa la integridad y la postura de todos los activos	Ningún dispositivo ni carga de trabajo es de confianza por defecto; su nivel de parches, su configuración y su integridad se miden de forma continua.	Una imagen de contenedor sin una firma válida, o con un CVE crítico sin parchear, no se admite en el clúster.
6. La autenticación y la autorización son dinámicas y se aplican estrictamente antes del acceso	La identidad se comprueba antes de cada acceso y se vuelve a comprobar durante las sesiones largas; las credenciales son de corta duración y se rotan.	Los certificados de los servicios duran horas y se renuevan automáticamente; las sesiones de usuario se reevalúan cuando cambia el riesgo.
7. La empresa recopila toda la información posible sobre el estado de los activos, la red y las comunicaciones	La telemetría retroalimenta la política para que las decisiones mejoren con el tiempo.	Los registros de acceso del gateway, la malla y el proveedor de identidad fluyen a la plataforma de monitorización de seguridad.

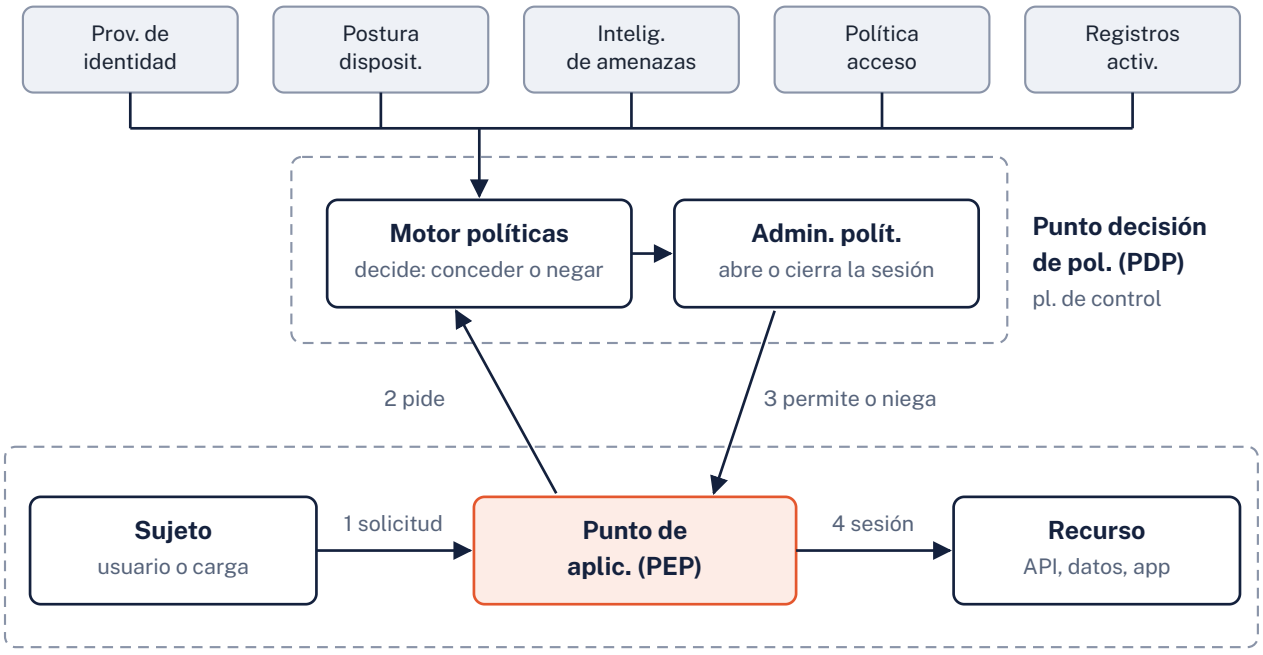
Puntos de decisión y de aplicación de políticas

SP 800-207 describe la maquinaria con un pequeño conjunto de componentes lógicos. Un punto de aplicación de políticas (PEP) se sitúa delante de un recurso y deja pasar el tráfico solo cuando se le indica: en la práctica, un proxy con reconocimiento de identidad, un API gateway,

un sidecar de una service mesh o un proxy de base de datos. El punto de decisión de políticas (PDP) toma la decisión. El NIST lo divide en un motor de políticas, que evalúa la solicitud frente a la política y las entradas y decide, y un administrador de políticas, que ejecuta la decisión abriendo o cerrando la vía, por ejemplo emitiendo un token de sesión de corta duración o configurando el PEP. Las entradas incluyen el proveedor de identidad, la postura del dispositivo y de la carga de trabajo (nivel de parches, configuración, integridad), la inteligencia de amenazas, las propias políticas de acceso y los registros de actividad.

Separar la decisión de la aplicación es el mismo movimiento que separar la política del mecanismo en los sistemas operativos, y compensa por la misma razón: las políticas pueden escribirse una vez, revisarse, probarse y cambiarse de forma centralizada, mientras que la aplicación se reparte por todos los lugares por donde fluye el tráfico. Los motores de políticas de la Sección 8.3, como Open Policy Agent, suelen ser el motor que hay detrás de un PDP. La pregunta de diseño para un arquitecto es dónde poner los puntos de aplicación para que no exista ningún camino que los rodee, y cómo mantener la decisión rápida y disponible, porque un PDP en la ruta crítica de cada solicitud es ahora una dependencia con su propio presupuesto de disponibilidad y de latencia. Las respuestas habituales son almacenar en caché las decisiones durante unos segundos, enviar políticas compiladas al PEP para que pueda decidir localmente y decidir de antemano qué ocurre si no se puede contactar con el PDP (normalmente denegar, con una pequeña lista de permitidos para las comprobaciones de estado).

Los componentes lógicos de una arquitectura zero trust



Plano de datos: sin confianza implícita por ubicación en red; toda solicitud cruza un PEP

Las solicitudes solo viajan a través de un punto de aplicación; la decisión se toma en un plano de control separado, alimentado con datos de identidad, postura, amenazas y política.

BeyondCorp: zero trust para las personas

Google empezó a abandonar su red corporativa privilegiada después de la intrusión de 2009 conocida como Operación Aurora, y describió el resultado en *BeyondCorp: A New Approach to Enterprise Security* (Rory Ward y Betsy Beyer, ;login:, diciembre de 2014), al que siguieron varios artículos más. El diseño tiene tres partes principales. Cada dispositivo se registra en un inventario de dispositivos y se identifica mediante un certificado. Un proxy de acceso, accesible desde internet, se sitúa delante de cada aplicación interna; los empleados llegan a las aplicaciones a través de él desde cualquier red, incluida la de la oficina, que no recibe ningún privilegio especial. Un componente de inferencia de confianza combina quién es el usuario con el estado del dispositivo para asignar un nivel de confianza, y cada aplicación declara el nivel mínimo que exige.

La consecuencia que sorprendió fue la eliminación de la VPN: un empleado en una cafetería y un empleado en su escritorio de la oficina pasan las mismas comprobaciones. Hoy la idea se vende en productos con los nombres de proxy con reconocimiento de identidad (identity-aware proxy) y acceso a la red zero trust (ZTNA). El Zero Trust Maturity Model de la Cybersecurity and Infrastructure Security Agency de EE. UU. (versión 2.0, abril de 2023) es una herramienta

útil de autoevaluación: puntúa cinco pilares (identidad, dispositivos, redes, aplicaciones y cargas de trabajo, y datos) desde tradicional hasta óptimo, con visibilidad, automatización y gobernanza transversales a todos ellos.

TLS mutuo, certificados e identidad de carga de trabajo

Zero trust entre servicios necesita dos cosas en cada llamada: cifrado y una identidad sólida para ambos extremos. Transport Layer Security proporciona las dos. TLS 1.3 (RFC 8446, agosto de 2018) completa su negociación (handshake) en un solo viaje de ida y vuelta: el cliente envía los algoritmos que admite y una parte de la clave (key share), el servidor responde con su parte de la clave y su certificado, y ambos derivan las claves de sesión con Diffie-Hellman efímero, de modo que cada sesión tiene secreto hacia adelante (forward secrecy): una clave de servidor robada no puede descifrar el tráfico pasado que se haya grabado. TLS 1.3 eliminó las opciones débiles de versiones anteriores, como el intercambio de claves RSA, los cifrados en modo CBC, RC4, las firmas SHA-1, la compresión y la renegociación; TLS 1.0 y 1.1 quedaron formalmente obsoletos con el RFC 8996 en 2021. Un modo opcional sin viaje de ida y vuelta (0-RTT) permite a un cliente que vuelve enviar datos de inmediato, pero un atacante puede reenviar esos datos tempranos, así que deben limitarse a solicitudes idempotentes. Los principales navegadores y proveedores de nube también han empezado a usar un intercambio de claves híbrido poscuántico que combina X25519 con ML-KEM (estandarizado por el NIST como FIPS 203 en 2024), lo que protege el tráfico grabado hoy frente a una futura computadora cuántica.

Un certificado vincula una clave pública a un nombre y está firmado por una autoridad de certificación (CA); el cliente comprueba la cadena de firmas hasta una raíz en la que ya confía. Este sistema de CA, certificados, almacenes de confianza y revocación es una infraestructura de clave pública (PKI). Los certificados públicos también se anotan en los registros de Certificate Transparency, de modo que se puede detectar un certificado emitido por error para tu dominio. El HTTPS normal solo autentica al servidor. En el TLS mutuo (mTLS), el cliente también presenta un certificado, así que el servidor sabe qué carga de trabajo está llamando. Los certificados internos proceden de una CA privada que gestiona la propia organización, normalmente dentro de su plataforma de secretos o de su malla, porque las CA públicas no emiten certificados para nombres internos.

¿Qué nombre va en el certificado? Los nombres de host y las direcciones IP son malas identidades para los contenedores. SPIFFE (Secure Production Identity Framework for Everyone, un proyecto graduado de la Cloud Native Computing Foundation) estandariza la identidad de carga de trabajo: cada carga de trabajo recibe un URI como `spiffe://fooddelivery.example/ns/prod/sa/orders` y un SPIFFE Verifiable Identity Document de corta duración, ya sea un certificado X.509 para mTLS o un JWT. Su implementación de referencia, SPIRE, atestigua la carga de trabajo (comprueba qué cuenta de

servicio de Kubernetes o qué instancia de nube es realmente) antes de emitir el documento, y lo rota automáticamente, normalmente más o menos cada hora. Así, las reglas de autorización pueden decir *Pedidos puede llamar a Pagos* en lugar de nombrar direcciones.

Service meshes

Implementar mTLS, la rotación de certificados, los reintentos y la autorización por llamada dentro de cada servicio, en cada lenguaje, es caro e incoherente. Una service mesh (malla de servicios) traslada ese trabajo a la infraestructura. En el diseño clásico, cada instancia de servicio recibe un proxy sidecar (Envoy en Istio, un microproxy escrito en Rust en Linkerd) que intercepta su tráfico; un plano de control emite identidades, distribuye políticas y recopila telemetría. Los diseños más recientes sin sidecar, como el modo ambient de Istio, ponen un proxy compartido por nodo a cargo de mTLS y añaden un proxy aparte solo donde se necesita política a nivel de HTTP. NIST SP 800-207A (septiembre de 2023) describe este patrón como la forma de aplicar zero trust a las aplicaciones nativas de la nube: segmentación basada en la identidad, con políticas que la malla aplica en cada llamada.

Una malla es una decisión de plataforma importante. Añade latencia en cada salto (normalmente pequeña, pero medible en el p99), memoria por pod, un plano de control que a su vez debe tener alta disponibilidad y estar protegido, y una curva de aprendizaje para cada equipo. Además, solo protege el tráfico que pasa por ella: un job que se conecta directamente a la base de datos la esquiva. Los sistemas pequeños con un puñado de servicios suelen arreglarse mejor con mTLS proporcionado por una biblioteca o por los balanceadores de carga de la plataforma. Las preguntas decisivas son cuántos servicios y lenguajes tienes y si existe un equipo de plataforma que la opere (consulta la Sección 10.7).

Automatizar los certificados

Los certificados caducan, y los certificados caducados son una de las causas más comunes de caídas evitables. El 6 de diciembre de 2018, un certificado caducado en software de Ericsson usado por la red móvil británica O2 dejó a unos 32 millones de clientes sin servicio de datos durante la mayor parte de un día. La cura es la automatización. El protocolo ACME (Automatic Certificate Management Environment, RFC 8555, 2019), desarrollado para la autoridad de certificación Let's Encrypt, permite que el software demuestre el control de un dominio (sirviendo un token por HTTP, añadiendo un registro DNS o respondiendo a un desafío TLS), obtenga un certificado y lo renueve sin intervención de una persona. Let's Encrypt, una CA gratuita y sin ánimo de lucro que empezó a emitir certificados en 2015, convirtió en norma los certificados de corta duración renovados automáticamente; herramientas como cert-manager hacen lo mismo dentro de Kubernetes, y todos los grandes balanceadores de carga en la nube pueden gestionar los certificados por ti.

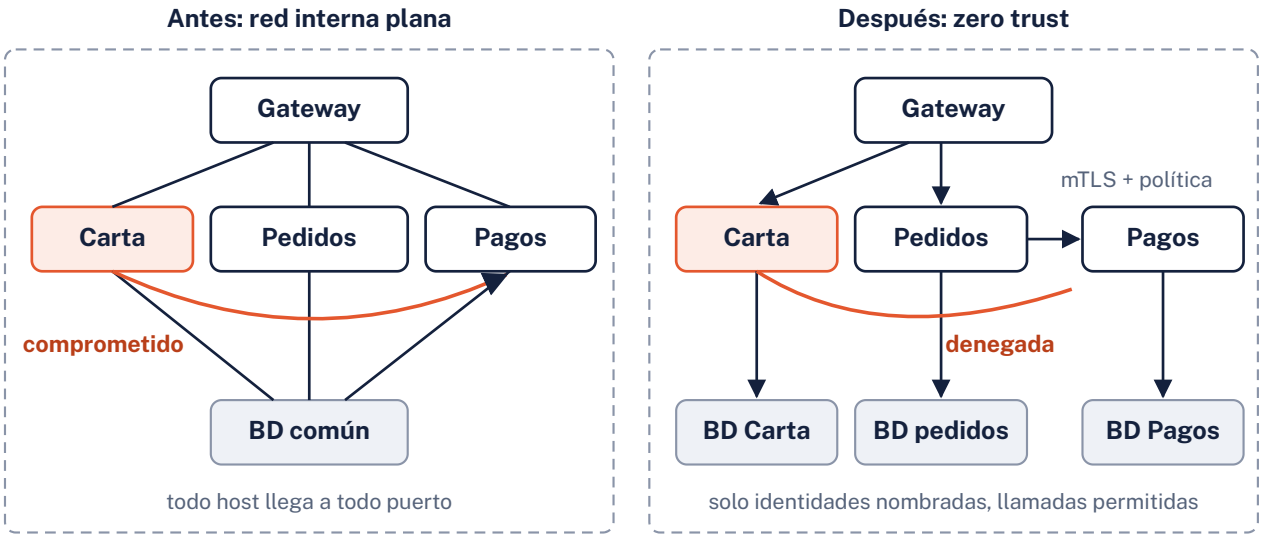
La automatización ya no es opcional. Según la votación SC-081 del CA/Browser Forum, la vida útil máxima de los certificados TLS de confianza pública bajó a 200 días para los certificados emitidos a partir del 15 de marzo de 2026, y baja a 100 días a partir del 15 de marzo de 2027 y a 47 días a partir del 15 de marzo de 2029. Las vidas útiles cortas limitan el daño de una clave robada y reducen la dependencia de la revocación, que siempre ha funcionado mal. En términos de arquitectura, esto significa que cada certificado, público o interno, necesita un propietario, renovación automatizada, monitorización de las fechas de caducidad con alertas mucho antes del plazo y un procedimiento manual alternativo ensayado.

Segmentación y microsegmentación

Zero trust no significa que la red ya no importe; significa que la red ya no es el único control. La segmentación de red divide la infraestructura en zonas con caminos controlados entre ellas: subredes públicas para los balanceadores de carga, subredes privadas para los servicios, subredes aisladas para las bases de datos y cuentas de nube separadas para producción, preproducción y herramientas de seguridad. La microsegmentación reduce las zonas a cargas de trabajo individuales, de modo que cada servicio solo puede recibir tráfico de los llamadores que necesita y solo puede enviar tráfico a sus dependencias declaradas.

En Kubernetes, todos los pods pueden llegar a todos los demás pods salvo que los recursos NetworkPolicy lo restrinjan (y esas políticas solo surten efecto si el plugin de red del clúster las aplica), así que el primer paso es una política de denegación por defecto por namespace, seguida de permisos explícitos. El control del tráfico de salida (egress) es tan importante como el de entrada (ingress): un servicio comprometido que no puede abrir conexiones hacia internet no puede sacar fácilmente datos robados ni descargar más herramientas. Los grupos de seguridad de la nube, las políticas de red de Kubernetes y las políticas de autorización de la malla trabajan en capas distintas (direcciones y puertos, pods y etiquetas, identidades de carga de trabajo y métodos HTTP) y lo mejor es usarlas juntas: las capas exteriores dan una protección gruesa que sobrevive a un error de configuración de la malla, y la capa interior expresa la intención con precisión. Esto es la defensa en profundidad (Sección 8.1) aplicada a la red.

De una red plana a la segmentación basada en la identidad



Zero trust no impide que un servicio se vea comprometido; limita a qué puede llegar el servicio comprometido.

Proteger el API gateway

El API gateway es el principal punto de aplicación entre internet y los servicios, así que asume varias funciones. **Autenticación:** termina TLS y luego valida cada token de acceso (firma con un algoritmo fijado, emisor, audiencia, caducidad y momento de validez inicial), rechaza las solicitudes que no lo traen y pasa a los servicios una identidad verificada, nunca las afirmaciones en bruto del cliente. **Limitación de tasa (rate limiting):** limita las solicitudes por cliente, por usuario y por ruta, normalmente con un token bucket, respondiendo HTTP 429 con una cabecera `Retry-After`; límites separados y más estrictos protegen el inicio de sesión, el restablecimiento de contraseña y el canje de cupones frente al relleno de credenciales (credential stuffing) y la enumeración. **Validación de la entrada:** impone tamaños máximos de solicitud y de número de cabeceras, acepta solo los tipos de contenido declarados y valida los cuerpos frente al esquema OpenAPI para que las solicitudes mal formadas nunca lleguen a un servicio. Un firewall de aplicaciones web puede añadir firmas de ataques genéricas, pero es un complemento, no un sustituto de un código correcto.

Conoce lo que un gateway no puede hacer. Ve la ruta y el token, pero no si el pedido 81723 pertenece al cliente que lo pide. La autorización rota a nivel de objeto (broken object level authorisation), en la que un usuario cambia un identificador en la URL y lee los datos de otra persona, ocupó el primer puesto del OWASP API Security Top 10 tanto en su edición de 2019 como en la de 2023, y solo el servicio propietario de los datos puede impedirla. Por tanto, los servicios deben autorizar ellos mismos cada solicitud, usando la identidad verificada que reenvía el gateway. Cuando un servicio llama a otro en nombre de un usuario, debe pasar un token que nombre tanto al usuario como al servicio que llama, obtenido por ejemplo mediante

el intercambio de tokens de OAuth 2.0 (RFC 8693), en lugar de su propia credencial todopoderosa; de lo contrario, un bug en un servicio lo convierte en un delegado confundido (confused deputy) que puede actuar en nombre de cualquiera.

EJEMPLO PRÁCTICO

Ejemplo práctico: llevar la plataforma de reparto de comida de una red plana a zero trust

La plataforma ejecuta unos veinte servicios en un único clúster de Kubernetes y una única cuenta de nube. Los servicios se llaman entre sí por HTTP sin cifrar, seis servicios usan un mismo usuario de base de datos compartido, el personal accede a la consola de administración por VPN, y el modelo de amenazas de la Sección 8.2 señaló como riesgo principal el movimiento lateral tras cualquier compromiso individual. Un cambio de golpe rompería el tráfico, así que la migración se hace por etapas y cada paso es reversible.

1. Inventariar y observar (semanas 1 a 3)	Haz una lista de cada servicio, su propietario y sus llamadores. Instala la malla en modo permisivo, que acepta tanto tráfico sin cifrar como mTLS, y usa su telemetría para construir el grafo de llamadas real. Sorpresa encontrada: el job de analítica llama directamente a la base de datos de Pagos.
2. Dar una identidad a cada carga de trabajo (semanas 3 a 5)	Una cuenta de servicio de Kubernetes por servicio; la malla emite certificados SPIFFE válidos durante 24 horas, rotados automáticamente. El usuario de base de datos compartido se sustituye por una credencial por servicio, emitida por el gestor de secretos para una hora cada vez.
3. Cifrarlo todo (semanas 5 a 8)	Pasa los namespaces a mTLS estricto de uno en uno, empezando por los de bajo riesgo. Los paneles muestran los llamadores que siguen usando texto sin cifrar; cada uno se corrige antes de que su namespace cambie. El rollback es un único cambio de configuración por namespace.
4. Autorizar cada llamada (semanas 8 a 12)	Escribe políticas de permiso a partir del grafo de llamadas observado: Pedidos puede llamar a Pagos en `POST /authorisations`; Carta no puede llamar a nada de Pagos. Ejecútalas en modo auditoría durante dos semanas, revisa las denegaciones y luego aplícalas. Añade NetworkPolicies de Kubernetes de denegación por defecto y reglas de salida como capa exterior.
5. Sustituir la VPN del personal (semanas 10 a 16)	Pon la consola de administración y las herramientas internas detrás de un proxy con reconocimiento de identidad que exija inicio de sesión único, MFA resistente al phishing y un dispositivo gestionado y actualizado. Los reembolsos por encima de un umbral necesitan un nuevo paso de MFA. La VPN se apaga cuando ya nada la usa.
6. Reforzar el borde (en paralelo)	El gateway valida los tokens, aplica límites de tasa por ruta y validación de esquema, y reenvía una identidad verificada; cada servicio sigue comprobando que quien llama puede ver ese pedido o esa cuenta concretos.
7. Medir y mantenerlo vigente	Las funciones de aptitud hacen fallar la compilación si un servicio nuevo no tiene política de autorización, y se disparan alertas ante llamadas denegadas y certificados a menos de siete días de caducar. Cada dependencia nueva es un cambio de política revisado.

El orden importa: observar antes de aplicar, identidad antes que política, modo auditoría antes que denegación. Cada paso puede revertirse por separado, lo que mantiene el riesgo de la migración por debajo del riesgo que elimina.

El resultado no es que nada pueda vulnerarse. Un servicio de Carta comprometido todavía puede hacer mal uso de sus propios datos y de las llamadas que tiene permitidas, pero ya no puede llegar a pagos ni a las bases de datos de otros servicios, y sus intentos quedan registrados y generan alertas.

ATENCIÓN

Ideas erróneas habituales sobre zero trust

- **No es un producto.** El NIST describe zero trust como unos principios y una arquitectura; los proveedores venden componentes (proxies, proveedores de identidad, mallas) que implementan partes de ella.
- **No elimina los controles de red.** La segmentación y los firewalls se mantienen como capas exteriores; simplemente ya no son lo único que hay entre un atacante y los datos.
- **mTLS por sí solo no es zero trust.** El cifrado con identidad pero con una política que lo permite todo sigue dejando que cualquier servicio comprometido llame a cualquier cosa; la clave es la autorización por llamada.
- **No se termina en un solo proyecto.** El modelo de madurez de la CISA existe porque las organizaciones avanzan pilar a pilar a lo largo de años.
- **No debe hacer inutilizable la seguridad.** La aceptabilidad psicológica de Saltzer y Schroeder sigue vigente: las solicitudes de confirmación frecuentes enseñan a la gente a aprobarlo todo, así que las comprobaciones adicionales (step-up) deben reservarse para las acciones de riesgo.

Preguntas frecuentes

¿Seguimos necesitando una VPN?
Para las personas que acceden a aplicaciones web, un proxy con reconocimiento de identidad suele sustituirla. Las VPN o los enlaces privados siguen siendo razonables para conexiones entre sedes y para protocolos que un proxy no puede manejar, pero no deben conceder una confianza amplia a lo que haya al otro lado.

¿Hace falta una service mesh para zero trust?
No. Es una forma cómoda de tener mTLS, identidad de carga de trabajo y política por llamada en todas partes. Los balanceadores de carga de la plataforma, las bibliotecas y las funciones de identidad de la nube pueden lograr lo mismo para un número pequeño de servicios.

¿Zero trust hace más lento el sistema?
Cada comprobación añade un poco de latencia, normalmente muy por debajo de un milisegundo con un proxy local de la malla y unos pocos milisegundos cuando hace falta una decisión remota. Almacenar en caché las decisiones, llevar la política al punto de aplicación y reutilizar las conexiones la mantienen pequeña; mídela en el p99 en las rutas críticas.

Seguridad del navegador: la política del mismo origen, CORS y CSRF

Hasta ahora, quienes llamaban eran servicios y apps que tienen sus propias credenciales. Un navegador es distinto: ejecuta código de muchos sitios en paralelo y adjunta automáticamente las cookies del usuario. Por eso, el sitio web de clientes y el portal de restaurantes de la plataforma tienen un límite de confianza dentro del navegador del cliente, donde ningún gateway puede llegar. El punto de partida es la **política del mismo origen**. Dos URL comparten origen solo si coinciden el esquema, el host y el puerto, y el script de un origen tiene restricciones sobre cómo puede acceder a los recursos de otro. La restricción recae principalmente en la *lectura*: las lecturas entre orígenes suelen bloquearse, mientras que las escrituras entre orígenes, como los envíos de formularios, suelen permitirse.

El **uso compartido de recursos entre orígenes** (CORS) es la relajación controlada de esa regla. Si el sitio de `app.example.com` llama a `api.example.com`, son orígenes distintos, así que la API debe responder con `Access-Control-Allow-Origin` nombrando el sitio. Para cualquier cosa que vaya más allá de una solicitud simple (un `PUT` o `DELETE`, una cabecera personalizada, un cuerpo JSON), el navegador pide primero permiso con una solicitud preflight `OPTIONS`. Hay tres puntos que a menudo se malinterpretan. Es el navegador quien aplica CORS, y el servidor solo declara una política, así que herramientas como `curl` lo ignoran y la API debe seguir autenticando y autorizando cada llamada. CORS solo relaja la política del mismo origen; no añade ninguna protección. Y una solicitud simple sigue llegando al servidor y puede surtir efecto: solo se retiene la respuesta. Un origen comodín se rechaza en las solicitudes con credenciales, y devolver tal cual cualquier `Origin` que llegue anula el propósito, así que compáralo con una lista de permitidos.

La **falsificación de solicitudes entre sitios** (CSRF) explota exactamente esa brecha. Una página de otro sitio hace que el navegador envíe una solicitud a la plataforma, la cookie del cliente puede ir con ella y el servidor ve lo que parece el cliente cambiando una dirección de entrega. Las principales defensas de OWASP son un token impredecible comprobado en cada solicitud que cambie el estado (por sesión cuando el servidor guarda estado, una cookie double-submit firmada cuando no) o una comprobación de la cabecera `Sec-Fetch-Site` con la verificación de `Origin` como alternativa. El atributo de cookie `SameSite` de la Sección 8.3 es una defensa en profundidad útil, pero no un sustituto: `Lax` sigue enviando cookies en las navegaciones `GET` de nivel superior entre sitios, así que un `GET` nunca debe cambiar el estado. Un token enviado por script en una cabecera `Authorization` no se adjunta automáticamente, así que no queda expuesto de esta forma, pero un backend for frontend devuelve las cookies a escena y necesita protección contra CSRF. Una advertencia: el cross-site scripting anula todas estas defensas, lo que lleva al siguiente control.

Content Security Policy y cabeceras de seguridad

Content Security Policy (CSP) es una cabecera de respuesta que indica al navegador qué puede cargar y ejecutar una página. Su principal uso es limitar el daño del cross-site scripting: con una política en vigor, los scripts en línea inyectados, los manejadores de eventos en línea y `eval()` se bloquean salvo que la política los permita explícitamente. La especificación del W3C deja claro que CSP se usa mejor como defensa en profundidad: reduce el daño de una inyección, pero no sustituye a la validación de entradas ni a la codificación de salidas. CSP Level 3 sigue siendo un borrador de trabajo, así que consulta la versión actual y la compatibilidad de los navegadores.

Las listas de hosts permitidos a menudo se pueden eludir y son difíciles de mantener, por lo que las recomendaciones actuales prefieren una **CSP estricta** basada en nonces o hashes, por ejemplo `script-src 'nonce-RANDOM'; object-src 'none'; base-uri 'none'`. Un nonce debe ser un valor nuevo e impredecible en cada respuesta, lo que tiene una consecuencia arquitectónica: las páginas deben generarse por respuesta, mientras que a las páginas estáticas en caché les convienen los hashes, que hay que recalcular cada vez que cambia un script. Todos los scripts de terceros, como un widget de pago o una etiqueta de analítica, deben inventariarse y permitirse. El despliegue seguro es `Content-Security-Policy-Report-Only`: la política notifica las infracciones a un endpoint sin bloquear nada, y se aplica cuando los informes están limpios. La directiva `frame-ancestors 'none'` también impide que otros sitios enmarquen la página de pago, una defensa contra el clickjacking.

Unas pocas cabeceras más son baratas y merece la pena configurarlas en todas partes. `Strict-Transport-Security` (HSTS, RFC 6797) indica al navegador que use solo HTTPS para el sitio; como la primera visita no está protegida, los sitios también pueden solicitar su inclusión en la lista de precarga de los navegadores. `X-Content-Type-Options: nosniff` impide que el

navegador adivine los tipos de contenido. `Referrer-Policy: strict-origin-when-cross-origin` mantiene las rutas y las referencias de pedidos fuera del referrer enviado a otros sitios. `Permissions-Policy` desactiva las funciones del navegador que una página no necesita; la página de los repartidores puede necesitar la geolocalización, la de la carta no. OWASP desaconseja confiar en la antigua cabecera `X-XSS-Protection` y recomienda establecerla en `0`; consulta su cheat sheet para ver los valores actuales.

Lo que preocupa al arquitecto es dónde residen. Si cada equipo las configura a mano, algún servicio lo olvidará. Configúralas una sola vez en el borde o en el gateway, o en una plantilla compartida, y prueba las respuestas desplegadas en el pipeline como una función de aptitud (Sección 11.3).

COMPRUEBA LO QUE HAS ENTENDIDO

▼ ¿Por qué el TLS mutuo entre servicios, con todos los servicios autorizados a llamar a todos los demás, no llega a ser zero trust?

Proporciona cifrado e identidad, pero no autorización. Un servicio comprometido todavía puede llamar a cualquier cosa; zero trust necesita una política por llamada que permita solo las llamadas que cada identidad necesita realmente.

▼ ¿Qué diferencia hay entre el motor de políticas y el punto de aplicación de políticas?

El motor de políticas decide si una solicitud debe permitirse, usando la política y entradas como la identidad y la postura; el punto de aplicación está en la ruta del tráfico y aplica esa decisión permitiendo o bloqueando la solicitud.

▼ ¿Por qué son atractivos los certificados de corta duración y qué exigen?

Una clave robada solo es útil hasta que caduca el certificado, lo que reduce la dependencia de la revocación. Exigen una emisión y una renovación totalmente automatizadas, además de monitorizar la caducidad.

▼ Un cliente cambia el número de pedido en una URL y ve el pedido de otro cliente. ¿Qué componente debería haberlo impedido?

El servicio propietario de los pedidos, comprobando que el cliente autenticado es el dueño de ese pedido concreto. El gateway autenticó correctamente al usuario, pero no puede saber quién es el dueño de cada objeto.

▼ **Inténtalo: Un equipo dice: "El API gateway valida todos los tokens y nuestros servicios están en una red privada, así que no necesitan más comprobaciones". ¿Qué podría salir mal y qué añadirías con un enfoque zero trust (confianza cero)?**

Un solo servicio comprometido o una credencial robada pueden moverse lateralmente y llamar a todo, porque la ubicación se está tratando como confianza. Además, un token válido no demuestra que este cliente pueda ver este pedido. Da a cada servicio una identidad verificada con TLS mutuo, idealmente de corta duración y emitida automáticamente, y permite solo las llamadas que necesita. Cada servicio debe hacer la comprobación a nivel de objeto sobre los datos de los que es dueño. La pasarela sigue siendo útil para los tokens, los límites de tasa y la validación de entradas.

FUENTES

- NIST SP 800-207: Zero Trust Architecture (2020): https://csrc.nist.gov/pubs/sp/800/207/final
- Ward and Beyer: BeyondCorp, A New Approach to Enterprise Security (2014): https://research.google/pubs/beyondcorp-a-new-approach-to-enterprise-security/
- RFC 8446: The Transport Layer Security (TLS) Protocol Version 1.3: https://www.rfc-editor.org/rfc/rfc8446
- RFC 8555: Automatic Certificate Management Environment (ACME): https://www.rfc-editor.org/rfc/rfc8555
- SPIFFE: overview of the SPIFFE standards: https://spiffe.io/docs/latest/spiffe-about/overview/
- MDN: Same-origin policy: https://developer.mozilla.org/en-US/docs/Web/Security/Same-origin_policy
- MDN: Cross-Origin Resource Sharing (CORS): https://developer.mozilla.org/en-US/docs/Web/HTTP/Guides/CORS
- OWASP Cross-Site Request Forgery Prevention Cheat Sheet: https://cheatsheetseries.owasp.org/cheatsheets/Cross-Site_Request_Forgery_Prevention_Cheat_Sheet.html
- MDN: Content Security Policy (CSP): https://developer.mozilla.org/en-US/docs/Web/HTTP/Guides/CSP
- W3C: Content Security Policy Level 3: https://www.w3.org/TR/CSP3/
- OWASP HTTP Headers Cheat Sheet: https://cheatsheetseries.owasp.org/cheatsheets/HTTP_Headers_Cheat_Sheet.html
- RFC 6797: HTTP Strict Transport Security (HSTS): https://www.rfc-editor.org/rfc/rfc6797.html

SECCIÓN 8.6 DE 8.8 · INTERMEDIO · 29 MIN DE LECTURA · ÚLTIMA REVISIÓN: NOVIEMBRE DE 2026

Seguridad de la cadena de suministro de software

ANTES DE LEER ESTO

Esta sección se basa en:

- 0.3 Organizar el código
- 0.8 Cómo se hace el software
- 8.1 Principios de seguridad

Términos anteriores que utiliza: Dependencia, Pipeline de CI/CD, Imagen de contenedor, Superficie de ataque, Mínimo privilegio

LA IDEA EN BREVE

Un restaurante no cultiva su propia harina ni fabrica sus propios hornos; confía en sus proveedores. Si la harina de un proveedor está contaminada, todos los restaurantes que la compraron se ven afectados, por limpia que esté su propia cocina. Los buenos restaurantes llevan una lista de cada ingrediente y de su origen, compran a proveedores que precintan y etiquetan sus entregas, y comprueban los precintos al recibirlas. El software se construye igual, a partir de cientos de componentes escritos por otras personas y ensamblados por máquinas de compilación. La seguridad de la cadena de suministro consiste en saber qué hay dentro de tu software, de dónde vino y que nadie lo manipuló por el camino.

LO QUE NECESITAS SABER

- La mayor parte del código de un servicio moderno procede de dependencias, herramientas de compilación, acciones de CI e imágenes base; cada una de ellas, y cualquiera que pueda modificarlas, forma parte de la superficie de ataque.
- event-stream (2018) abusó de la confianza en un mantenedor, SolarWinds (2020) de un sistema de compilación comprometido, Log4Shell (2021) de una biblioteca vulnerable enterrada en lo profundo de los árboles de dependencias, y xz utils (2024) de una paciente toma de control de un proceso de lanzamiento.
- Una SBOM en formato SPDX o CycloneDX enumera los componentes con identificadores exactos, de modo que la exposición a una nueva vulnerabilidad se puede encontrar con una consulta; la Orden Ejecutiva 14028 de EE. UU. (2021) y el Reglamento de Ciberresiliencia de la UE generalizaron las SBOM.

- SLSA (versión 1.2) define los niveles Build L0 a L3, que terminan en compilaciones reforzadas y aisladas, y niveles Source que llegan hasta la revisión por dos personas; la procedencia registra quién compiló un artefacto y a partir de qué.
- La firma sin claves (keyless) de Sigstore vincula un certificado de corta duración a una identidad de CI y anota las firmas en un registro de transparencia público; las comprobaciones de admisión verifican las firmas y la procedencia antes del despliegue.
- Los lockfiles con hashes, fijar las acciones de CI y las imágenes por digest, las actualizaciones pequeñas y automatizadas con un breve retraso para las versiones nuevas, el análisis priorizado y las compilaciones efímeras y herméticas reducen el riesgo en el trabajo diario.
- Las obligaciones de notificación del CRA se aplican desde el 11 de septiembre de 2026 (alerta temprana en 24 horas, notificación en 72 horas) y la mayoría de las demás obligaciones desde el 11 de diciembre de 2027.

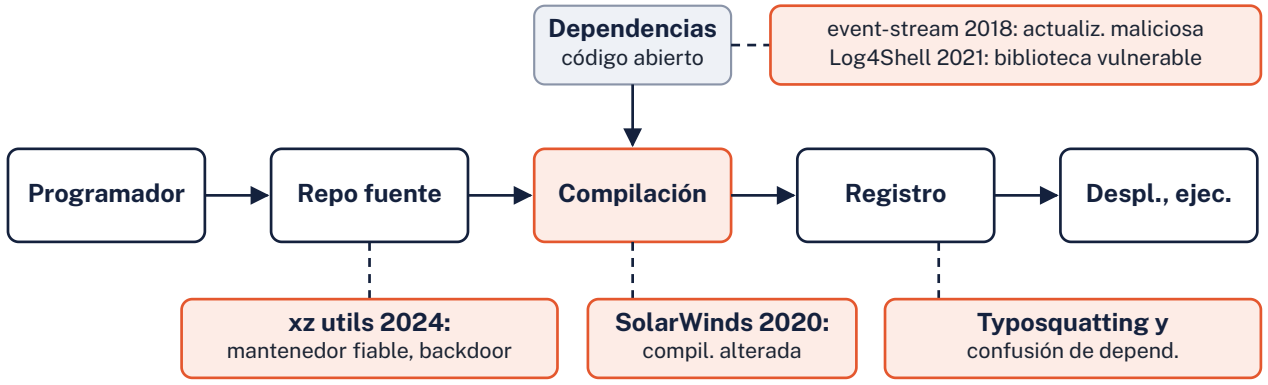
Por qué las dependencias son una superficie de ataque

La mayor parte del código de un servicio moderno no la escribió el equipo que lo opera. Un pequeño servicio web puede declarar veinte dependencias directas y, a través de las dependencias de estas, instalar varios cientos de paquetes; el contenedor en el que se entrega añade el espacio de usuario de un sistema operativo, y el pipeline que lo compila ejecuta compiladores, plug-ins y acciones de CI escritos por todavía más personas. La Sección 0.3 presentó las dependencias como un ahorro de esfuerzo con un costo de mantenimiento. Esta sección examina su costo en seguridad: cada uno de esos componentes, y cada persona y sistema capaz de modificarlos, forma parte de tu superficie de ataque.

Un ataque a la cadena de suministro de software compromete algo que consumes para llegar hasta ti, normalmente a muchas víctimas a la vez. Puede llegar de cuatro formas generales. **Código malicioso desde una fuente de confianza:** se roba mediante phishing la cuenta de un mantenedor, un mantenedor se vuelve hostil o un proyecto se entrega a un desconocido. **Imitaciones maliciosas:** el typosquatting publica `reqeusts` junto a `requests`; la *confusión de dependencias* (dependency confusion), demostrada por Alex Birsan en febrero de 2021, publica en el registro público un paquete con el mismo nombre que un paquete interno de una empresa y un número de versión más alto, que algunas herramientas de compilación prefieren. **Manipulación en el pipeline:** el código fuente está bien, pero el sistema de compilación, las claves de firma o el registro están comprometidos. **Vulnerabilidades honestas:** ningún atacante intervino al escribir el código, pero un bug grave en una dependencia transitiva muy usada expone a todos los que la entregan.

Los últimos años han mostrado lo rápido que se propaga el primer tipo. En marzo de 2025, las etiquetas de versión de una GitHub Action popular, tj-actions/changed-files, se movieron a un commit malicioso que imprimía los secretos de CI en los registros de compilación, lo que afectó a miles de repositorios que hacían referencia a la acción por etiqueta. En septiembre de 2025, un mantenedor de paquetes npm muy usados, como `chalk` y `debug`, sufrió phishing mediante un falso sitio de soporte de npm, y se publicaron versiones maliciosas que se retiraron en cuestión de horas; ese mismo mes, un gusano autopropagable, apodado Shai-Hulud, robó tokens de npm y los usó para publicar versiones infectadas de más paquetes.

La cadena de suministro y las relaciones de confianza de las que abusan los atacantes



Cada flecha es una relación de confianza; cada incidente famoso abusó de una distinta

El código fuente, la compilación, el registro y las dependencias son relaciones de confianza distintas, y cada incidente famoso atacó una diferente.

Cuatro incidentes, contados con precisión

event-stream (2018). event-stream era un paquete npm popular, descargado unos dos millones de veces por semana, cuyo autor original ya no tenía tiempo para él. En septiembre de 2018 cedió los derechos de publicación a un voluntario que se había ofrecido a ayudar. El nuevo mantenedor añadió una dependencia, `flatmap-stream`, cuyo paquete publicado contenía una carga útil cifrada que solo se activaba dentro de la compilación de Copay, una aplicación de monedero de bitcoin, e intentaba robar monederos con saldos grandes. Un desarrollador detectó el código sospechoso en noviembre de 2018. Nada del modelo de npm se había roto: simplemente, un nombre de confianza había cambiado de manos.

SolarWinds (2020). Unos atacantes, que el Gobierno de EE. UU. atribuyó más tarde al servicio de inteligencia exterior de Rusia, lograron acceso al entorno de compilación de SolarWinds, el fabricante del producto de gestión de redes Orion. Un implante (que los investigadores llamaron SUNSPOT) vigilaba las compilaciones de Orion y sustituía un archivo fuente por una versión con puerta trasera durante la compilación, y después volvía a poner el original. Las actualizaciones resultantes, publicadas entre marzo y junio de 2020, las había compilado el

propio pipeline de SolarWinds y estaban firmadas con su certificado auténtico. Unos 18 000 clientes las instalaron; después, los atacantes eligieron un número mucho menor de organismos públicos y empresas para continuar la intrusión. FireEye descubrió la campaña en diciembre de 2020 mientras investigaba su propia brecha. La lección para los arquitectos: una firma demuestra quién compiló algo, no que la compilación estuviera limpia.

Log4Shell (2021). CVE-2021-44228, divulgada el 9 de diciembre de 2021 y con una puntuación de 10.0 en CVSS, era un defecto en Log4j 2, la biblioteca de logging dominante en Java. Log4j resolvía las búsquedas (lookups) escritas dentro de los mensajes de registro, incluidas las búsquedas JNDI que podían cargar una clase desde un servidor remoto, así que un atacante que consiguiera que se registrara una cadena, como un nombre de usuario o una cabecera de solicitud, podía ejecutar código en el servidor. Ningún atacante había tocado la cadena de suministro; el daño vino de lo extendida, y lo profundamente integrada, que estaba la biblioteca. Las organizaciones dedicaron semanas a averiguar dónde la usaban, hicieron falta varias versiones posteriores para cerrar defectos relacionados, y quienes tenían un inventario preciso respondieron a la pregunta en horas.

xz utils (2024). xz es una biblioteca de compresión que usa, a través de otras bibliotecas, el servidor OpenSSH en varias distribuciones de Linux. A lo largo de unos dos años, una cuenta llamada Jia Tan aportó trabajo útil y, con la ayuda de otras cuentas que presionaban al sobrecargado mantenedor original para que aceptara ayuda, se convirtió en comantenedor. Las versiones 5.6.0 y 5.6.1, lanzadas en febrero y marzo de 2024, contenían una puerta trasera (backdoor) que permitiría al poseedor de una clave privada concreta ejecutar comandos a través de SSH. El código malicioso estaba oculto en archivos de prueba binarios y lo activaba un script de compilación modificado que solo existía en los tarballs de la versión, no en el repositorio público. El 29 de marzo de 2024, Andres Freund, desarrollador de Microsoft, lo reveló tras investigar por qué los inicios de sesión SSH en un sistema de pruebas usaban una cantidad inusual de CPU y tardaban alrededor de medio segundo más; se registró como CVE-2024-3094. Como se detectó pronto, había llegado sobre todo a distribuciones de desarrollo y de lanzamiento continuo (rolling release) y no a las estables.

Lo que enseña cada incidente

Incidente	Qué ocurrió	Eslabón atacado	Controles que habrían ayudado
event-stream (npm), 2018	Un nuevo mantenedor voluntario, al que el agotado autor original había cedido el paquete, añadió una dependencia (flatmap-stream) que llevaba una carga útil cifrada dirigida a una aplicación de monedero de bitcoin concreta.	La confianza en los mantenedores y las dependencias transitivas	Lockfiles y un retraso antes de adoptar versiones nuevas; revisión de las nuevas dependencias transitivas; financiación y relevo para los paquetes críticos.
SolarWinds Orion, 2020	Atacantes dentro de la red del proveedor implantaron código que sustituía un archivo fuente durante la compilación, de modo que actualizaciones firmadas legítimamente contenían la puerta trasera SUNBURST.	Compilación	Compilaciones aisladas, efímeras y monitorizadas; procedencia; compilaciones reproducibles para comparar resultados; mínimo privilegio para la propia herramienta de gestión de redes.
Log4Shell (Log4j 2), 2021	Una funcionalidad antigua que resolvía búsquedas JNDI en el texto registrado permitía a cualquier atacante que lograra que se registrara una cadena ejecutar código en el servidor.	Una dependencia vulnerable y profundamente anidada	Una SBOM para encontrar en minutos cada sistema afectado; filtrado del tráfico de salida; aplicación de parches rápida y ensayada.
xz utils, 2024	Tras unos dos años de contribuciones pacientes, un comantenedor entregó tarballs de la versión con una puerta trasera oculta dirigida a OpenSSH en algunas distribuciones de Linux.	El código fuente y el proceso de lanzamiento	Compilar desde el repositorio revisado, no desde tarballs hechos a mano; compilaciones reproducibles; examen atento de los archivos de prueba binarios y de los scripts de compilación.

Listas de materiales de software

Una lista de materiales de software (SBOM) es una lista legible por máquina de los componentes de un software, con sus versiones, sus identificadores y las relaciones entre ellos. No hace el software más seguro por sí misma; hace que las preguntas tengan respuesta. Cuando se anuncie el próximo Log4Shell, un almacén consultable de SBOM de cada artefacto desplegado convierte “¿estamos afectados, y dónde?” en una consulta.

Dos formatos dominan. SPDX, de la Linux Foundation, nació como formato de cumplimiento de licencias; su versión 2.2.1 es la norma internacional ISO/IEC 5962:2021, y SPDX 3.0 (2024) añade perfiles para seguridad, compilaciones e IA. CycloneDX, un proyecto de OWASP diseñado para casos de uso de seguridad, está normalizado por Ecma International como ECMA-424; la versión 1.7 se lanzó en octubre de 2025. Ambos pueden identificar componentes con una package URL (purl), como `pkg:npm/express@4.21.2`, que es con lo que se cotejan las bases de datos de vulnerabilidades. La idea complementaria de VEX (Vulnerability Exploitability eXchange) permite a un proveedor declarar que un componente vulnerable de la lista no es explotable en su producto, por ejemplo porque la función vulnerable nunca se llama, lo que reduce el ruido para los clientes.

Las políticas públicas llevaron las SBOM a la corriente principal. La Orden Ejecutiva 14028 de EE. UU., *Improving the Nation's Cybersecurity*, firmada el 12 de mayo de 2021, estableció medidas de cadena de suministro de software para el software vendido al Gobierno federal, entre ellas proporcionar a los compradores una SBOM; la National Telecommunications and Information Administration publicó después los elementos mínimos de una SBOM (proveedor, nombre del componente, versión, otros identificadores únicos, relaciones de dependencia, autor de los datos de la SBOM y una marca de tiempo). Desde entonces, la política de EE. UU. se ha alejado de los mandatos uniformes: en enero de 2026, el memorando M-26-05 de la Office of Management and Budget revocó el requisito anterior de que los proveedores certificaran sus prácticas de desarrollo seguro, y dejó que los organismos fijaran requisitos basados en el riesgo y pidieran SBOM cuando lo consideraran oportuno. En la UE, el Reglamento de Ciberresiliencia, descrito más abajo, hace que la SBOM forme parte de la documentación técnica de los productos con elementos digitales.

Genera las SBOM durante la compilación, a partir de lo que realmente se resolvió y empaquetó, y no del manifiesto que escribió un desarrollador; guárdalas junto al artefacto, indexadas por su digest; y pásalas a un servicio que vuelva a comprobar cada SBOM almacenada cuando se publique una nueva vulnerabilidad.

```
{
  "bomFormat": "CycloneDX",
  "specVersion": "1.7",
  "metadata": {"timestamp": "2026-10-20T09:14:02Z",
               "component": {"bom-ref": "orders-service", "type": "application",
                             "name": "orders-service", "version": "4.12.0"}},
  "components": [
    {"bom-ref": "log4j-core", "type": "library", "name": "log4j-core",
     "group": "org.apache.logging.log4j", "version": "2.24.3",
     "purl": "pkg:maven/org.apache.logging.log4j/log4j-core@2.24.3",
     "hashes": [{"alg": "SHA-256", "content": "9f2c..."}]},
    {"bom-ref": "jackson-databind", "type": "library", "name": "jackson-databind",
     "version": "2.18.2",
     "purl": "pkg:maven/com.fasterxml.jackson.core/jackson-databind@2.18.2"}
  ],
  "dependencies": [
    {"ref": "orders-service", "dependsOn": ["log4j-core", "jackson-databind"]}
  ]
}
```

Una SBOM de CycloneDX abreviada para el servicio de pedidos de la plataforma de reparto de comida. El purl y el hash identifican cada componente con exactitud, para que un escáner pueda cotejarlo con las bases de datos de vulnerabilidades.

SLSA: niveles de integridad de la compilación

SLSA (Supply-chain Levels for Software Artifacts, que se pronuncia *salsa*) es un marco, mantenido ahora bajo la Open Source Security Foundation, que describe niveles progresivos de protección frente a la manipulación. La versión 1.0 (2023) definió una vía de compilación (Build track); la versión 1.2 reestructuró la especificación en varias vías y añadió una vía de código fuente (Source track). El artefacto central es la procedencia (provenance): una declaración firmada, normalmente en el formato de atestación in-toto, de qué plataforma de compilación produjo un artefacto, a partir de qué revisión del código fuente y con qué definición y parámetros de compilación. Un consumidor verifica la procedencia frente a expectativas como *compilado por nuestra CI a partir de la rama main de github.com/fooddelivery/orders*.

La vía Source tiene cuatro niveles: control de versiones (L1), historial continuo e inmutable con procedencia para cada revisión (L2), controles técnicos aplicados de forma continua, como las ramas protegidas (L3), y revisión por dos personas de cada cambio en las ramas protegidas (L4). Para la mayoría de las organizaciones, el objetivo realista es Build L3 en una plataforma de compilación compartida y bien gestionada, más Source L3 o L4 para los repositorios importantes; los niveles dicen lo que una plataforma debe garantizar, y un equipo de plataforma puede cumplirlos una sola vez para todos los servicios.

Niveles de la vía Build de SLSA

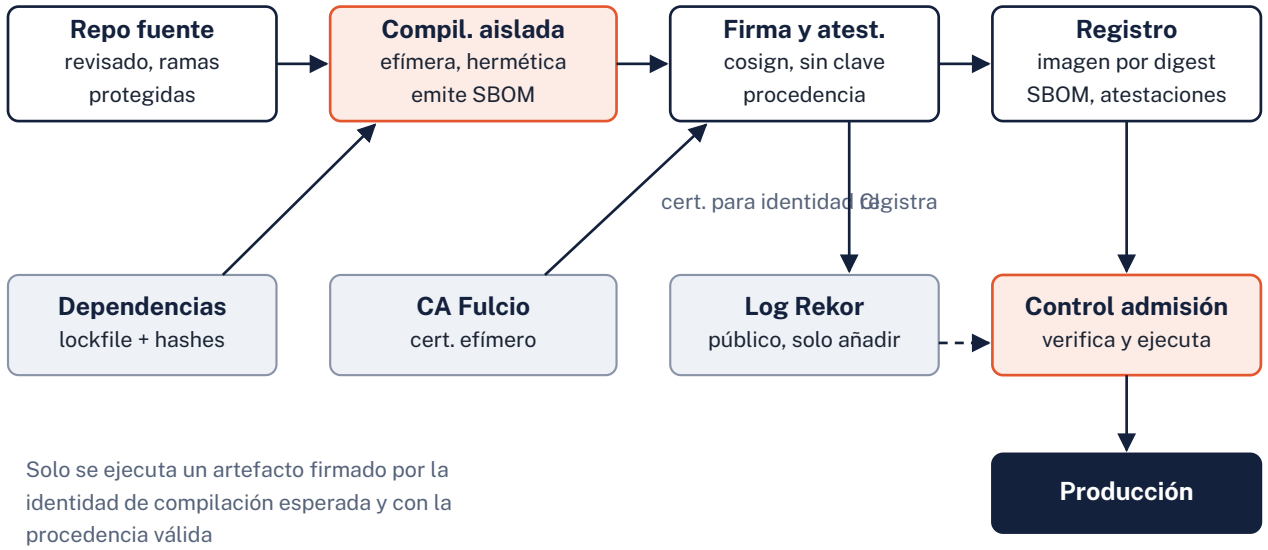
Nivel	Nombre	Qué garantiza
Build L0	Sin garantías	Sin requisitos: la ausencia de SLSA.
Build L1	Existe procedencia	La compilación produce una procedencia que describe cómo se compiló el artefacto. Detecta errores, pero es fácil de falsificar.
Build L2	Plataforma de compilación alojada	Un servicio de compilación alojado genera y firma la procedencia, lo que protege frente a la manipulación posterior a la compilación.
Build L3	Compilaciones reforzadas	La plataforma de compilación aísla las compilaciones entre sí y mantiene el material de firma fuera del alcance de los pasos de compilación definidos por el usuario, lo que protege frente a la manipulación durante la compilación.

Firma y procedencia con Sigstore

Firmar artefactos es posible desde hace mucho, pero gestionar claves de firma de larga duración era tan engorroso que pocos proyectos de código abierto lo hacían, y las claves robadas eran en sí mismas un riesgo. Sigstore, lanzado en 2021 y ahora parte de la Open Source Security Foundation, abarata la firma mediante la firma *sin claves* (keyless). Un job de CI demuestra su identidad ante Fulcio, la autoridad de certificación de Sigstore, con un token de OpenID Connect; Fulcio emite un certificado válido durante unos minutos que nombra esa identidad, por ejemplo un workflow concreto de un repositorio concreto; la herramienta firma el artefacto y anota la firma en Rekor, un registro de transparencia público en el que solo se pueden añadir entradas; después se descarta la clave privada efímera. Los verificadores comprueban la firma, la identidad del certificado y la entrada del registro, en lugar de confiar en una clave que podría haberse filtrado.

La herramienta cosign aplica esto a las imágenes de contenedor y a otros artefactos, y los ecosistemas de paquetes han adoptado las mismas ideas: npm puede publicar paquetes con declaraciones de procedencia que los vinculan con el repositorio de código y la compilación (2023), y PyPI admite la publicación de confianza (trusted publishing) desde CI sin tokens de larga duración (2023) y atestaciones basadas en Sigstore (2024). En el despliegue, un controlador de admisión de Kubernetes, como el policy controller de Sigstore o Kyverno, puede rechazar cualquier imagen que no esté firmada por la identidad esperada o que carezca de la procedencia esperada. Recuerda SolarWinds: la firma y la procedencia demuestran de dónde vino algo; solo el aislamiento de la compilación, la revisión y la detección hacen fiable lo que salió.

Un pipeline reforzado del commit a producción



Solo se ejecuta un artefacto firmado por la identidad de compilación esperada y con la procedencia válida

Cada etapa produce pruebas que la siguiente puede verificar, de modo que producción solo ejecuta lo que el pipeline esperado compiló a partir de código fuente revisado.

Lockfiles, fijación de versiones y actualizaciones automatizadas

Un manifiesto como `package.json` o `pyproject.toml` indica los rangos de versiones aceptables; un lockfile (archivo de bloqueo) (`package-lock.json` , `poetry.lock` , `uv.lock` , `go.sum` , `Cargo.lock`) registra las versiones exactas resueltas, normalmente con un hash criptográfico de cada una. Hacer commit del lockfile e instalar a partir de él en CI (`npm ci` , `pip install --require-hashes`) significa que cada compilación usa los mismos bytes revisados, y que un paquete republicado con un contenido distinto no supera la comprobación del hash. La fijación de versiones (pinning) se aplica también a las herramientas de compilación: haz referencia a las acciones de CI por su hash de commit completo, no por una etiqueta que su propietario, o un atacante, puede mover, que es justo lo que hizo que se propagara el incidente de tj-actions; haz referencia a las imágenes base por digest.

Fijar sin actualizar crea el riesgo contrario: las versiones antiguas acumulan vulnerabilidades conocidas y las actualizaciones se vuelven demasiado grandes para intentarlas. Los bots de actualización como Dependabot y Renovate abren pull requests pequeñas y frecuentes que la CI prueba automáticamente. Un refinamiento útil es una antigüedad mínima de la versión (minimum release age): esperar unos días antes de adoptar una versión nueva rutinaria, porque la mayoría de las versiones maliciosas se detectan y se retiran en horas o días; las correcciones de seguridad quedan exentas y se aceleran.

Las herramientas de análisis de composición de software (SCA), como OWASP Dependency-Check, Trivy, Grype o servicios comerciales, cotejan tus dependencias con las bases de datos de vulnerabilidades. Su punto débil es el ruido: cientos de hallazgos, la mayoría en rutas de código que nunca llamas. Prioriza con datos de explotación (¿está en el catálogo Known Exploited Vulnerabilities de la CISA?, ¿cuál es su puntuación EPSS?), con análisis de alcanzabilidad cuando tus herramientas lo ofrezcan, y con la exposición (¿el servicio está expuesto a internet?). La Sección 8.8 trata la gestión de vulnerabilidades como proceso.

SI ESTO ES NUEVO PARA TI

Paso a paso: el mismo código, una compilación distinta

El punto en el que más gente tropieza aquí es que el mismo código fuente puede producir un software distinto en días distintos si las dependencias exactas no están bloqueadas. Sigue una biblioteca a lo largo de dos días de compilaciones del servicio de pedidos.

- El manifiesto del servicio de pedidos pide una biblioteca en la 'versión 2.18 o cualquier 2.x posterior'. El lunes, la compilación descarga la coincidencia más reciente, 2.18.2, y las pruebas pasan.
- El martes aparece la versión 2.18.3, publicada por el mantenedor o quizá por un atacante que ha robado mediante phishing la cuenta del mantenedor. Nadie del equipo cambia una línea, y aun así la compilación del martes descarga la 2.18.3, porque también encaja en el rango.
- Lo mismo puede ocurrir en las propias dependencias transitivas de la biblioteca, que el equipo nunca eligió.
- Un lockfile lo impide. El lunes, el gestor de paquetes registra la versión exacta de cada paquete que resolvió, incluidos los transitivos, normalmente con un hash criptográfico: una huella del contenido exacto del paquete. El equipo hace commit del archivo, y la CI instala solo lo que figura en él.
- Si alguna vez la 2.18.2 se republica con un contenido distinto, la huella deja de coincidir y la compilación falla. Las herramientas de compilación necesitan el mismo cuidado: una acción de CI referenciada por una etiqueta puede moverse a otro código, como en el incidente de tj-actions, mientras que un hash de commit completo siempre nombra el mismo código.
- Bloquear no es el final. Las versiones antiguas acumulan vulnerabilidades conocidas, así que un bot de actualización propone pequeñas actualizaciones para que la CI las pruebe. Esperar unos días antes de adoptar las versiones rutinarias permite detectar antes la mayoría de las maliciosas; las correcciones de seguridad pasan rápido.
- **La lección:** decide deliberadamente cuándo cambian tus dependencias. Bloquea las versiones exactas para que cada compilación use los mismos bytes revisados y luego actualiza en pasos pequeños y probados.

Contenedores e integridad de la compilación

Una imagen de contenedor lleva el espacio de usuario de un sistema operativo además de tu código, y cada paquete que contiene forma parte de la cadena de suministro. Parte de imágenes base mínimas (distroless, Alpine o imágenes slim de distribuciones), que contienen menos paquetes que puedan ser vulnerables; recompila las imágenes con regularidad para que

las correcciones de la imagen base lleguen aunque tu código no haya cambiado; analiza las imágenes en CI y de nuevo de forma continua en el registro, porque cada día se publican nuevas vulnerabilidades que afectan a imágenes antiguas; y despliega las imágenes por digest para que lo que se analizó sea lo que se ejecuta.

El sistema de compilación es el objetivo más valioso de la cadena porque lo toca todo, como demostró SolarWinds. Ejecuta las compilaciones en runners efímeros que se crean para un solo job y se destruyen después, para que un atacante no pueda persistir entre compilaciones. Haz que las compilaciones sean herméticas, con todas las entradas declaradas y descargadas por hash y sin acceso a la red durante la compilación, para que no puedan colarse entradas no declaradas. Mantén los secretos de larga duración fuera de la CI usando la federación OIDC para obtener credenciales de nube de corta duración para cada job, y da a cada pipeline el mínimo privilegio que necesita, para que una pull request desde un fork no pueda llegar a las credenciales de despliegue. Protege la ruta de lanzamiento con protección de ramas y revisión por dos personas. Por último, las compilaciones reproducibles, en las que el mismo código fuente produce siempre un resultado idéntico bit a bit, permiten que terceros independientes recompilen y comparen; el proyecto Reproducible Builds y distribuciones como Debian han hecho reproducibles la mayoría de sus paquetes, y una discrepancia es un indicio sólido de manipulación.

El Reglamento de Ciberresiliencia de la UE

El Reglamento de Ciberresiliencia (Cyber Resilience Act) (CRA), Reglamento (UE) 2024/2847, establece requisitos horizontales de ciberseguridad para los *productos con elementos digitales* introducidos en el mercado de la UE: productos de hardware y software, incluidas sus soluciones de tratamiento de datos a distancia. Entró en vigor el 10 de diciembre de 2024 y se aplica por etapas. Las normas sobre la notificación de los organismos de evaluación de la conformidad se aplican desde el 11 de junio de 2026. Las obligaciones de notificación del artículo 14 se aplican desde el 11 de septiembre de 2026: un fabricante debe notificar una vulnerabilidad aprovechada activamente o un incidente grave que afecte a la seguridad de su producto con una alerta temprana en un plazo de 24 horas desde que tenga conocimiento, una notificación más completa en un plazo de 72 horas y un informe final más adelante (para una vulnerabilidad, en un plazo de 14 días desde que haya disponible una medida correctora; para un incidente grave, en un plazo de un mes), a través de la plataforma única de notificación que gestiona ENISA, la agencia de ciberseguridad de la UE. La mayoría de las demás obligaciones se aplican desde el 11 de diciembre de 2027.

Los requisitos esenciales del anexo I son tanto de arquitectura como de procedimiento: los productos deben diseñarse con un nivel de seguridad adecuado, entregarse con una configuración segura por defecto, limitar las superficies de ataque, proteger la confidencialidad y la integridad de los datos, minimizar los datos y ser capaces de registrar

eventos de seguridad y de recibir actualizaciones de seguridad. Los fabricantes deben identificar y documentar los componentes, entre otras cosas elaborando una SBOM que cubra al menos las dependencias de máximo nivel, gestionar las vulnerabilidades mediante una política de divulgación coordinada y proporcionar actualizaciones de seguridad durante un periodo de soporte que normalmente debería ser de al menos cinco años, salvo que se prevea que el producto se use durante menos tiempo. Las multas por incumplir los requisitos esenciales o las obligaciones de los fabricantes pueden llegar a 15 millones de EUR o al 2.5% del volumen de negocios anual mundial, si esta cifra es superior.

El ámbito importa. El software libre y de código abierto desarrollado o suministrado fuera de una actividad comercial queda fuera del CRA, y se aplica un régimen más ligero a los *administradores de comunidad de programas informáticos de código abierto* que dan soporte a ese software para uso comercial. El software como servicio puro suele quedar fuera del CRA, salvo que sea la solución de tratamiento de datos a distancia de un producto; los proveedores de servicios en la nube y otros servicios digitales pueden quedar sujetos, en cambio, a la Directiva NIS2. Si un producto concreto entra en el ámbito, y qué vía de evaluación de la conformidad se aplica, es una cuestión jurídica para cada organización; la consecuencia arquitectónica es que las SBOM, la gestión de vulnerabilidades, los mecanismos de actualización y el registro de eventos de seguridad se convierten en pruebas que tienen que existir, no en buenas intenciones.

> **NOTA**
>
> ### Información, no asesoramiento jurídico
>
> Las descripciones de la Orden Ejecutiva, el memorando de la OMB y el Reglamento de Ciberresiliencia resumen cómo afectan estas medidas a la arquitectura de software. No constituyen asesoramiento jurídico; las obligaciones dependen del producto, del mercado y del papel de la organización, y deben confirmarse con los textos oficiales y con asesores cualificados.

Lista de comprobación: seguridad de la cadena de suministro para un equipo de servicio

La mayoría de estos puntos son funcionalidades de la plataforma una vez y hábitos a partir de entonces.

- [ ] **Mantén un inventario**
  Genera una SBOM en cada compilación y guárdala con el artefacto, para que "¿estamos afectados?" sea una consulta y no un proyecto.
- [ ] **Bloquea y verifica las dependencias**
  Haz commit de los lockfiles, instala con comprobación de hashes en CI y haz fallar la compilación si el lockfile y el manifiesto no coinciden.
- [ ] **Elige las dependencias de forma deliberada**
  Antes de añadir una, comprueba su mantenimiento, el número de mantenedores, los resultados de OpenSSF Scorecard, la licencia y cuánto código arrastra de forma transitiva.
- [ ] **Actualiza de forma continua, pero no al instante**
  Deja que un bot proponga pequeñas actualizaciones cada semana; aplica una antigüedad mínima de la versión corta a las actualizaciones rutinarias; acelera las correcciones de seguridad.
- [ ] **Analiza y luego prioriza**
  Ejecuta el análisis de composición de software y el análisis de imágenes en CI y de forma continua en el registro; prioriza por explotabilidad y alcanzabilidad, no por cantidad.
- [ ] **Fija las herramientas de compilación además de las bibliotecas**
  Fija las acciones de CI y los plug-ins a hashes de commit completos y las imágenes base a digests; revisa sus cambios como si fueran código.
- [ ] **Aísla y refuerza la compilación**
  Runners efímeros, sin secretos de larga duración (usa la federación OIDC), tokens de mínimo privilegio, ramas protegidas y revisión por dos personas para las rutas de lanzamiento.
- [ ] **Firma y registra la procedencia**
  Firma los artefactos con Sigstore o un equivalente, publica la procedencia SLSA y aspira a Build L3 en la plataforma.

- [ ] **Verifica en el despliegue**
  Las políticas de admisión rechazan las imágenes sin firmar, firmadas por una identidad equivocada o que carecen de procedencia del repositorio esperado.

- [ ] **Prepárate para el próximo Log4Shell**
  Ensaya cómo encontrar y parchear una biblioteca en todos los servicios en un día, y conoce tus obligaciones de notificación según el CRA, los contratos y los clientes.

Preguntas frecuentes

¿Deberíamos copiar (vendoring) todas las dependencias en nuestro repositorio?
El vendoring da control y compilaciones sin conexión, pero dificulta las actualizaciones y oculta las correcciones de origen. Un lockfile con hashes más un espejo privado del registro que guarda en caché los paquetes aprobados da la mayor parte del control con menos mantenimiento.

¿Basta un registro privado para evitar la confusión de dependencias?
Solo si las compilaciones están configuradas para resolver los nombres internos exclusivamente desde él. Usa nombres de paquete con ámbito o con espacio de nombres, registra tus nombres internos en los registros públicos y configura el gestor de paquetes para que un paquete público nunca pueda sustituir a uno interno.

¿Quién es responsable de la seguridad de la cadena de suministro en la organización?
Normalmente es compartida: un equipo de plataforma o de seguridad proporciona una sola vez el pipeline reforzado, la firma, el análisis y las políticas; cada equipo de servicio mantiene sus dependencias pocas, actualizadas y justificadas.

COMPRUEBA LO QUE HAS ENTENDIDO

▼ **¿Por qué no protegieron a los clientes de SolarWinds sus comprobaciones de firma?**

La puerta trasera se insertó durante la propia compilación de SolarWinds, así que la actualización troyanizada estaba firmada auténticamente por SolarWinds. Una firma muestra quién produjo un artefacto, no que la compilación estuviera limpia.

▼ **¿Qué añade un lockfile a un manifiesto?**

El manifiesto indica los rangos de versiones aceptables; el lockfile registra las versiones exactas resueltas y normalmente sus hashes, de modo que cada compilación instala los mismos bytes y un paquete modificado no supera la verificación.

▼ **¿Por qué hacer referencia a una acción de CI por etiqueta es más arriesgado que por hash de commit?**

Quien controle el repositorio, incluido un atacante, puede mover una etiqueta, así que la misma referencia puede ejecutar en silencio un código distinto; un hash de commit completo identifica un contenido inmutable.

▼ **¿Qué pregunta responde una SBOM que un escáner de vulnerabilidades por sí solo no responde rápidamente en todo un parque de sistemas?**

Qué artefactos desplegados, en qué servicios y entornos, contienen un componente y una versión determinados, para poder encontrar en todas partes a la vez la exposición a una vulnerabilidad recién anunciada.

▼ **Inténtalo: Un pipeline de CI usa build-action@main, una imagen base etiquetada latest e instala paquetes sin lockfile. Nadie sabe decir qué servicios usan una biblioteca que acaba de resultar vulnerable. Ordena las correcciones que harías, con una razón para cada una.**

Primero, genera un SBOM por compilación, para poder encontrar una nueva vulnerabilidad mediante una consulta en minutos. A continuación, confirma los lockfiles con hashes y fija la acción de CI y la imagen base por digest, ya que las etiquetas pueden moverse y quien pueda cambiarlas puede llegar a tu compilación. Después, automatiza las actualizaciones pequeñas de dependencias, con un breve retraso para los lanzamientos nuevos. Por último, firma los artefactos con su procedencia y compruébalos en el despliegue. Cada paso limita lo que puede hacer un proveedor previo comprometido.

FUENTES

- SLSA specification v1.2: https://slsa.dev/spec/v1.2/
- Sigstore documentation: https://docs.sigstore.dev/
- Regulation (EU) 2024/2847 (Cyber Resilience Act), EUR-Lex: https://eur-lex.europa.eu/eli/reg/2024/2847/oj
- NVD: CVE-2024-3094 (xz utils backdoor): https://nvd.nist.gov/vuln/detail/CVE-2024-3094
- CycloneDX specification: https://cyclonedx.org/

SECCIÓN 8.7 DE 8.8 · INTERMEDIO · 34 MIN DE LECTURA · ÚLTIMA REVISIÓN: NOVIEMBRE DE 2026

Privacidad desde el diseño y el RGPD (GDPR)

ANTES DE LEER ESTO

Esta sección se basa en:

- 7.5 CQRS y event sourcing
- 8.2 Modelado de amenazas
- 8.4 Protección de datos

Términos anteriores que utiliza: Crypto-shredding (destrucción criptográfica), Residencia de datos, Diagrama de flujo de datos (DFD), Tiempo de vida (TTL)

LA IDEA EN BREVE

Imagina que le prestas tu diario a un amigo. Esperas que lea solo la página que le mostraste, que no la fotocopie, que no se lo quede para siempre y que te lo devuelva cuando se lo pidas. La privacidad desde el diseño incorpora esas expectativas al software desde el principio: recopilar solo lo necesario, usarlo solo para el fin indicado, conservarlo solo el tiempo necesario y poder mostrar a las personas qué se tiene sobre ellas y eliminarlo cuando lo pidan. Leyes como el RGPD (GDPR) convierten estas expectativas en obligaciones, y muchas de ellas solo pueden cumplirse si el sistema se diseñó para ello.

LO QUE NECESITAS SABER

- La privacidad se pregunta si los datos deberían recopilarse y usarse en absoluto, y qué control conservan las personas; un sistema puede ser seguro y aun así vulnerar la privacidad.
- El RGPD (GDPR, Reglamento (UE) 2016/679) y el UK GDPR junto con la Data Protection Act 2018, modificada por la Data (Use and Access) Act 2025, comparten los principios del artículo 5: licitud, lealtad y transparencia, limitación de la finalidad, minimización, exactitud, limitación del plazo de conservación, integridad y confidencialidad, y responsabilidad proactiva.
- El artículo 25 exige la protección de datos desde el diseño (medidas adecuadas como la seudonimización) y por defecto (que solo se traten, conserven y hagan accesibles los datos necesarios), a partir de los siete principios de Ann Cavoukian.
- Los derechos de acceso, rectificación, supresión, limitación, portabilidad y oposición requieren un mapa de datos, manejadores de exportación y de supresión en cada servicio, indicadores de limitación y listas de exclusión, con respuestas normalmente en el plazo de

un mes.

- Una EIPD es obligatoria antes de un tratamiento que probablemente entrañe un alto riesgo (artículo 35) y funciona mejor junto con el modelo de amenazas, sobre el mismo diagrama de flujo de datos.
- Los datos seudonimizados siguen siendo datos personales para quien pueda reidentificarlos; la anonimización debe resistir todos los medios que razonablemente puedan utilizarse, y es difícil de lograr.
- La supresión debe llegar a las cachés, los índices de búsqueda, la analítica, los logs de eventos (a menudo mediante crypto-shredding), los encargados del tratamiento y las copias de seguridad (que quedan fuera de uso hasta que caducan); esta sección es información sobre cómo afecta la ley a la arquitectura, no asesoramiento jurídico.

NOTA

Información, no asesoramiento jurídico

Esta sección explica cómo afecta la legislación de protección de datos a la arquitectura de software. Resume la ley con exactitud en el momento de redactarla, pero no es asesoramiento jurídico: si una norma se aplica, y cómo, depende de los hechos, y las organizaciones deben asesorarse con abogados especializados en protección de datos y con su delegado de protección de datos.

La privacidad es una propiedad del diseño

La seguridad y la privacidad se solapan, pero no son lo mismo. La seguridad se pregunta si los datos están protegidos frente a quienes no deberían tenerlos. La privacidad se pregunta además si la propia organización debería recopilar esos datos en absoluto, qué puede hacer con ellos, durante cuánto tiempo y qué control conserva la persona a la que describen. Un sistema puede no sufrir ninguna brecha y aun así invadir la privacidad, por ejemplo si conserva para siempre el rastro de ubicaciones de cada repartidor o si reutiliza discretamente los historiales de pedidos para entrenar un modelo de precios.

La plataforma de reparto de comida es un buen ejemplo de por qué la privacidad es arquitectónica. Guarda nombres, números de teléfono y domicilios; historiales de pedidos que pueden revelar la religión o la salud a través de las preferencias alimentarias; referencias de pago; y un flujo continuo de ubicaciones de los repartidores. Estos datos circulan desde las apps a través de decenas de servicios, eventos, cachés, logs, almacenes de analítica y copias de seguridad, y llegan a encargados del tratamiento como el proveedor de pagos, la pasarela de SMS y el SaaS de soporte. Las decisiones sobre dónde se almacenan los datos personales,

con qué clave se identifican y cómo pueden encontrarse y eliminarse son costosas de cambiar más adelante, que es exactamente la prueba para saber si algo es arquitectónicamente significativo.

La ley en breve: el RGPD y el UK GDPR

El Reglamento General de Protección de Datos, Reglamento (UE) 2016/679, se aplica desde el 25 de mayo de 2018. Protege los datos personales, que el artículo 4(1) define como toda información sobre una persona física identificada o identificable, directa o indirectamente, por ejemplo mediante un nombre, un número de identificación, datos de localización o un identificador en línea. El responsable del tratamiento decide por qué y cómo se tratan los datos y tiene la responsabilidad principal; un encargado del tratamiento trata los datos por cuenta del responsable en virtud de un contrato (artículo 28), como hace la pasarela de SMS de la plataforma. El RGPD se aplica a las organizaciones establecidas en la UE y, según el artículo 3, a otras que ofrecen bienes o servicios a personas en la UE o controlan su comportamiento.

En el Reino Unido, el RGPD se mantuvo tras el Brexit como UK GDPR, que funciona junto con la Data Protection Act 2018. La Data (Use and Access) Act 2025 modificó ambos; la mayoría de sus cambios en materia de protección de datos entraron en vigor el 5 de febrero de 2026, incluidos cambios en las bases jurídicas, las decisiones automatizadas y las transferencias internacionales, y a partir del 19 de junio de 2026 se suma la obligación de que los responsables gestionen las reclamaciones sobre protección de datos. La Comisión Europea renovó sus decisiones de adecuación para el Reino Unido en diciembre de 2025, de modo que los datos personales pueden seguir fluyendo de la UE al Reino Unido sin garantías adicionales. Para los arquitectos, los dos regímenes comparten los mismos principios, derechos y obligaciones de diseño; las diferencias son cuestiones de detalle para los abogados.

Las multas máximas por las infracciones más graves son de 20 millones de EUR o el 4% del volumen de negocio anual mundial según el RGPD (17.5 millones de GBP o el 4% según el UK GDPR), la cifra que sea mayor, aunque las consecuencias más habituales son órdenes de detener el tratamiento, daños a la reputación y el costo de rediseñar los sistemas. En noviembre de 2025, la Comisión Europea propuso modificaciones del RGPD como parte de un paquete *Digital Omnibus*; en el momento de redactar esto, siguen en tramitación legislativa y todavía no cambian nada.

Los principios del artículo 5 y lo que significan para los sistemas

Principio	En pocas palabras	Consecuencia arquitectónica
Licitud, lealtad y transparencia (5(1)(a))	Tratar los datos solo con una base jurídica, de formas que las personas esperarían razonablemente, y contarles lo que haces.	Registrar una base jurídica por finalidad; generar los avisos de privacidad a partir del mismo mapa de datos que usan los sistemas; nada de seguimiento oculto.
Limitación de la finalidad (5(1)(b))	Recopilar con fines determinados y explícitos, y no reutilizar los datos para fines incompatibles.	Etiquetar los datos con sus finalidades; comprobar la etiqueta antes de permitir que un nuevo consumidor (como un pipeline de aprendizaje automático) los lea.
Minimización de datos (5(1)(c))	Adecuados, pertinentes y limitados a lo necesario.	No recopilar la fecha de nacimiento si solo necesitas un indicador de mayor de 18 años; guardar el rastro de ubicaciones del repartidor solo mientras se necesite.
Exactitud (5(1)(d))	Exactos y, si fuera necesario, actualizados.	Un único propietario por dato; permitir que las personas corrijan sus datos y propagar las correcciones a las copias.
Limitación del plazo de conservación (5(1)(e))	Mantenidos de forma que permitan identificar a las personas durante no más tiempo del necesario.	Un plazo de conservación para cada conjunto de datos, aplicado automáticamente mediante TTL, eliminación de particiones y tareas de caducidad, también en los logs y las copias de seguridad.
Integridad y confidencialidad (5(1)(f))	Seguridad adecuada, incluida la protección contra el tratamiento no autorizado, la pérdida y los daños.	Todo lo de las Secciones 8.1 a 8.6: control de acceso, cifrado, gestión de claves, zero trust, controles de la cadena de suministro.
Responsabilidad proactiva (5(2))	El responsable del tratamiento responde del cumplimiento y debe poder demostrarlo.	Los registros de actividades de tratamiento, las EIPD, los registros de consentimiento, los registros de supresiones y los logs de auditoría son pruebas que la arquitectura debe producir.

Las bases jurídicas, brevemente

El artículo 6 solo permite el tratamiento sobre una de seis bases jurídicas: el consentimiento, la ejecución de un contrato con la persona, una obligación legal, intereses vitales, una misión realizada en interés público, o intereses legítimos que no prevalezcan sobre los intereses y derechos de la persona. Las categorías especiales de datos, como la salud o las creencias religiosas, necesitan una condición adicional según el artículo 9. La base se elige por finalidad,

no por sistema, y cambia lo que debe hacer la arquitectura. En la plataforma de reparto de comida, la dirección de entrega se trata para ejecutar el contrato; las facturas se conservan para cumplir una obligación legal derivada de la normativa fiscal; la detección de fraude suele basarse en intereses legítimos; y los correos electrónicos de marketing normalmente necesitan consentimiento, que también regulan las normas sobre marketing electrónico.

El consentimiento tiene la mecánica más estricta. Según el artículo 7, el responsable debe poder demostrar que se dio el consentimiento, este debe ser específico para una finalidad, y retirarlo debe ser tan fácil como darlo. Eso significa que el consentimiento no puede ser una sola columna booleana: el sistema necesita un registro de solo anexión (append-only) de qué aceptó la persona, cuándo, por qué canal y en respuesta a qué texto, y todos los consumidores deben respetar una retirada con prontitud.

```
{
  "event": "ConsentGiven",
  "subjectRef": "cust_4411",
  "purpose": "marketing_email",
  "noticeVersion": "privacy-notice-2026-09",
  "wording": "Send me offers from restaurants near me",
  "channel": "ios-app 7.3.0",
  "method": "unticked checkbox, user ticked",
  "at": "2026-10-03T18:22:41Z"
}
{
  "event": "ConsentWithdrawn",
  "subjectRef": "cust_4411",
  "purpose": "marketing_email",
  "channel": "email unsubscribe link",
  "at": "2026-10-19T07:05:12Z"
}
```

El consentimiento como registro de eventos de solo anexión: el estado actual se deriva del último evento por finalidad, y el historial es la prueba que exige el artículo 7.

Protección de datos desde el diseño y por defecto

El artículo 25 convierte la privacidad desde el diseño en una obligación legal. El artículo 25(1), la protección de datos desde el diseño, exige al responsable, tanto en el momento de decidir cómo tratar los datos como durante el propio tratamiento, aplicar medidas técnicas y organizativas apropiadas, como la seudonimización, que hagan efectivos los principios, teniendo en cuenta el estado de la técnica, el costo, la naturaleza y los fines del tratamiento y los riesgos. El artículo 25(2), por defecto, exige que por defecto solo se traten los datos necesarios para cada fin específico, lo que abarca la cantidad recopilada, la extensión del tratamiento, el plazo de conservación y la accesibilidad; en particular, los datos no deben ser

accesibles a un número indeterminado de personas sin la intervención de la persona afectada. Las Directrices 4/2019 sobre el artículo 25 del Comité Europeo de Protección de Datos (EDPB) ofrecen ejemplos prácticos.

La ley tomó la idea de Ann Cavoukian, comisionada de Información y Privacidad de Ontario, que desarrolló la privacidad desde el diseño en la década de 1990 y publicó sus siete principios fundamentales en 2009; la conferencia internacional de comisionados de protección de datos los respaldó en 2010. Siguen siendo una lista de comprobación útil para revisar diseños.

Los siete principios fundamentales de la privacidad desde el diseño de Cavoukian

Principio	Lo que pide a un diseño
1. Proactivo, no reactivo; preventivo, no correctivo	Anticipar los daños a la privacidad en el momento del diseño en lugar de corregirlos después de un incidente.
2. La privacidad como configuración predeterminada	La persona no tiene que hacer nada para estar protegida; el intercambio opcional está desactivado hasta que lo active.
3. Privacidad incorporada en el diseño	La privacidad forma parte de la arquitectura y de los procesos de negocio, no es un añadido.
4. Funcionalidad total: suma positiva, no suma cero	Buscar diseños que logren a la vez la privacidad y el objetivo de negocio en lugar de sacrificar uno de los dos.
5. Seguridad de extremo a extremo: protección durante todo el ciclo de vida	Proteger los datos desde su recopilación hasta su destrucción segura.
6. Visibilidad y transparencia	Las prácticas son abiertas y verificables por los usuarios y por revisores independientes.
7. Respeto por la privacidad del usuario	Mantener el foco en el usuario: valores predeterminados sólidos, avisos claros y opciones fáciles de usar.

Derechos que la arquitectura debe soportar

Los derechos de los interesados son el punto en el que la legislación de privacidad se convierte más directamente en requisitos del sistema. Un responsable debe responder normalmente en el plazo de un mes desde la solicitud, prorrogable hasta otros dos meses en el caso de solicitudes complejas o numerosas (artículo 12(3)), y debe poder encontrar los datos de la persona dondequiera que estén. En un sistema de veinte servicios, cinco encargados del tratamiento SaaS y un almacén de datos, eso es imposible sin un mapa de datos mantenido: el

registro de actividades de tratamiento que el artículo 30 ya exige, mantenido lo bastante cerca del código como para ser exacto. Con las modificaciones del Reino Unido de 2025, un responsable que atiende una solicitud de acceso solo está obligado a realizar búsquedas razonables y proporcionadas, algo que un buen mapa de datos agiliza.

Los derechos de los interesados y lo que cada uno exige de los sistemas

Derecho	Lo que la persona puede pedir	Lo que necesitan los sistemas
Acceso (art. 15)	Confirmación de que se tratan sus datos, una copia e información como los fines, los destinatarios, la conservación y el origen.	Un mapa de datos de cada almacén que contenga datos personales; una API de exportación en cada servicio con un ID de cliente estable como clave; verificación de identidad; una forma de reunir y entregar el resultado dentro del plazo.
Rectificación (art. 16)	Corrección de los datos inexactos y compleción de los datos incompletos.	Un único propietario autorizado por campo; correcciones propagadas a las copias y las cachés mediante eventos; destinatarios informados (art. 19).
Supresión (art. 17)	Eliminación cuando, entre otros motivos, los datos ya no son necesarios o se retira el consentimiento, salvo excepciones como las obligaciones legales.	Un orquestador de supresión, reglas de supresión por almacén (eliminar, anonimizar, conservar con un motivo), crypto-shredding para los almacenes inmutables y un registro que demuestre que se ha completado.
Limitación (art. 18)	Datos conservados pero sin ningún otro uso, por ejemplo mientras se discute su exactitud.	Un indicador de limitación que comprueba cada consumidor, de modo que los datos se almacenan pero quedan excluidos de tratamientos como el marketing o el entrenamiento de modelos.
Portabilidad (art. 20)	Los datos que la persona facilitó, tratados por medios automatizados sobre la base del consentimiento o de un contrato, en un formato estructurado, de uso común y de lectura mecánica.	Una exportación en JSON o CSV con campos documentados; opcionalmente, transmisión directa a otro responsable cuando sea técnicamente posible.
Oposición (art. 21)	Detener el tratamiento basado en intereses legítimos o en una misión pública salvo que existan motivos imperiosos; absoluta en el caso del marketing directo.	Interruptores por finalidad; una lista de exclusión (que guarde lo mínimo necesario, como un hash de la dirección de correo electrónico) para que una importación posterior no vuelva a añadir a la persona.
Decisiones automatizadas (art. 22)	Garantías frente a decisiones basadas únicamente en el tratamiento automatizado que produzcan efectos jurídicos o significativos similares.	Por ejemplo, la desactivación automática de un repartidor: una vía de revisión humana, una explicación y una forma de impugnarla.

Evaluaciones de impacto relativas a la protección de datos

El artículo 35 exige una evaluación de impacto relativa a la protección de datos (EIPD) antes de un tratamiento que probablemente entrañe un alto riesgo para los derechos y libertades de las personas, en particular la elaboración de perfiles sistemática y exhaustiva con efectos significativos, el tratamiento a gran escala de categorías especiales de datos o de datos relativos a infracciones penales, y la observación sistemática a gran escala de una zona de acceso público. Las autoridades de control publican listas de otros casos; el seguimiento continuo de la ubicación de los repartidores y la gestión algorítmica de su trabajo son candidatos típicos. Una EIPD describe el tratamiento y sus fines, evalúa su necesidad y proporcionalidad, evalúa los riesgos para las personas y registra las medidas que los abordan; si persiste un alto riesgo, el responsable debe consultar a la autoridad de control antes de empezar (artículo 36).

Los arquitectos deberían tratar una EIPD como la contrapartida de privacidad del modelo de amenazas de la Sección 8.2, hecha al mismo tiempo y sobre el mismo diagrama de flujo de datos. El marco LINDDUN, desarrollado en la KU Leuven, hace con las amenazas a la privacidad lo que STRIDE hace con las amenazas a la seguridad, con categorías como la vinculación, la identificación, la detección, la divulgación de datos y el desconocimiento. Las conclusiones de la EIPD se convierten en requisitos arquitectónicos, como *los rastros de ubicación se eliminan 30 días después de la entrega*, que pueden probarse.

Transferencias internacionales y residencia de datos

El capítulo V del RGPD restringe las transferencias de datos personales a países fuera del Espacio Económico Europeo. Una transferencia es lícita si la Comisión ha declarado adecuado el país (artículo 45), si existen garantías adecuadas como las cláusulas contractuales tipo de la Comisión (artículo 46), o al amparo de excepciones limitadas (artículo 49). Después de que el Tribunal de Justicia anulara el Escudo de la Privacidad UE-EE. UU. (Privacy Shield) en la sentencia Schrems II de julio de 2020, las transferencias basadas en cláusulas contractuales necesitan una evaluación de la legislación del país de destino; el Marco de Privacidad de Datos UE-EE. UU. (Data Privacy Framework), declarado adecuado en julio de 2023, cubre las transferencias a empresas estadounidenses que se certifiquen en él, y superó una primera impugnación ante el Tribunal General de la UE en septiembre de 2025 (*Latombe/Comisión*), aunque nuevos litigios, incluido un posible recurso de casación ante el Tribunal de Justicia, todavía podrían cambiar la situación.

De ello se derivan dos cuestiones arquitectónicas. Primera: una transferencia no es solo una base de datos en otra región; el acceso remoto de un equipo de soporte en el extranjero, los subencargados de un encargado SaaS y un servicio de logging alojado en otro lugar también cuentan, así que el mapa de datos debe registrar dónde se almacenan los datos y desde dónde

se accede a ellos. Segunda: los requisitos de residencia de datos, ya sean legales, contractuales o elegidos, se cumplen más fácilmente con células de despliegue regionales: cada región guarda los datos personales y las claves de sus clientes, los servicios globales solo ven referencias seudónimas y la replicación entre regiones es explícita y revisada. Incorporar la residencia a posteriori en una única base de datos global es uno de los cambios más costosos a los que puede enfrentarse una plataforma, y por eso pertenece a las restricciones que se recogen desde el principio.

Cumplimiento normativo fuera de la UE y el Reino Unido: SOC 2, ISO 27001, HIPAA y la CCPA

A medida que la plataforma vende a grandes cadenas de restaurantes o entra en nuevos países, aparecen otros nombres. Este pasaje es información, no asesoramiento jurídico: que una ley sea aplicable depende de los hechos, y las normas cambian, así que comprueba la situación actual con asesores cualificados. Los nombres se dividen en dos grupos: esquemas de aseguramiento que piden los clientes y leyes que te obligan.

SOC 2 es un informe de atestación, no una certificación ni una ley. Conforme a las normas del American Institute of CPAs (AICPA), un auditor independiente examina los controles de una organización de servicios frente a criterios de cinco categorías: seguridad, disponibilidad, integridad del procesamiento, confidencialidad y privacidad. La seguridad es obligatoria en todo informe, la organización añade las demás categorías que encajan con sus compromisos, y no se prescribe ninguna lista fija de controles. Un informe de **Tipo 1** examina el diseño de los controles en un momento dado; un informe de **Tipo 2** comprueba además que funcionaron eficazmente durante un período, normalmente de seis a doce meses, y es el más común de los dos. **ISO/IEC 27001:2022** (modificada en 2024 en relación con la acción por el clima) es una norma internacional de requisitos para un sistema de gestión de la seguridad de la información: una forma de gestionar la seguridad basada en el riesgo, con 93 controles de referencia en el Anexo A. A diferencia de SOC 2, conduce a un certificado, que solo es tan útil como su alcance declarado.

HIPAA es una ley federal de EE. UU., y su Security Rule (45 CFR Parts 160 y 164) exige salvaguardas administrativas, físicas y técnicas para la información de salud protegida en formato electrónico. Obliga a las entidades cubiertas (planes de salud, centros de intercambio de información sanitaria y muchos proveedores de atención médica) y a sus socios comerciales (business associates), no a todo aquel que tenga datos de salud. Normalmente, la plataforma quedaría fuera de su ámbito, pero repartir recetas para farmacias sería un motivo para buscar asesoramiento sobre si se ha convertido en socio comercial. HHS propuso normas más estrictas a finales de 2024, incluidos el cifrado y la autenticación multifactor obligatorios; comprueba si se han aprobado definitivamente.

La **California Consumer Privacy Act** (CCPA), modificada por la California Privacy Rights Act, se aplica a las empresas con ánimo de lucro que hacen negocios en California y superan un umbral: en líneas generales, unos ingresos brutos anuales superiores a una cifra fijada en $25 millones y ajustada a la inflación cada año impar ($26 625 000 desde enero de 2025; comprueba la cifra actual), o comprar, vender o compartir la información personal de 100 000 o más residentes u hogares de California, o obtener la mitad o más de sus ingresos de la venta o el intercambio de esa información. Los residentes pueden conocer, eliminar y corregir sus datos, excluirse de su venta o de su cesión para publicidad entre contextos (cross-context advertising) y limitar el uso de la información sensible. Las normas reglamentarias en vigor desde enero de 2026 añaden evaluaciones de riesgos y, por fases, auditorías de ciberseguridad. Otros estados de EE. UU. tienen sus propias leyes de privacidad.

La lección arquitectónica es que las exigencias se solapan: el mínimo privilegio, el cifrado, el logging, la recuperación probada y un inventario de datos se repiten en todas. Crea un único conjunto de controles y asígnalo a cada esquema, en lugar de un programa distinto para cada uno, y reduce el alcance mediante la clasificación y la segmentación de datos, como mostró la Sección 8.4 para PCI DSS.

Evidencia continua de cumplimiento

Un informe de Tipo 2 pregunta si un control funcionó todo el año, así que las capturas de pantalla reunidas la semana anterior a la auditoría son débiles y costosas. Es mejor recopilar la evidencia como subproducto del trabajo normal: logs del pipeline que muestran que cada lanzamiento pasó la revisión y el análisis, definiciones de infraestructura, revisiones de accesos, la pista de auditoría (Sección 8.8). Cada elemento se etiqueta con los controles que respalda, y un control que falla genera una alerta cuando falla, no meses después. La Sección 11.4 describe la parte de política como código. Open Security Controls Assessment Language (OSCAL), del NIST, ofrece formatos XML, JSON y YAML legibles por máquina para catálogos de controles, detalles de implementación y resultados de evaluación, y sustituye los documentos por datos estructurados. La automatización reúne evidencia; no sustituye el juicio del auditor ni a las personas responsables de los controles.

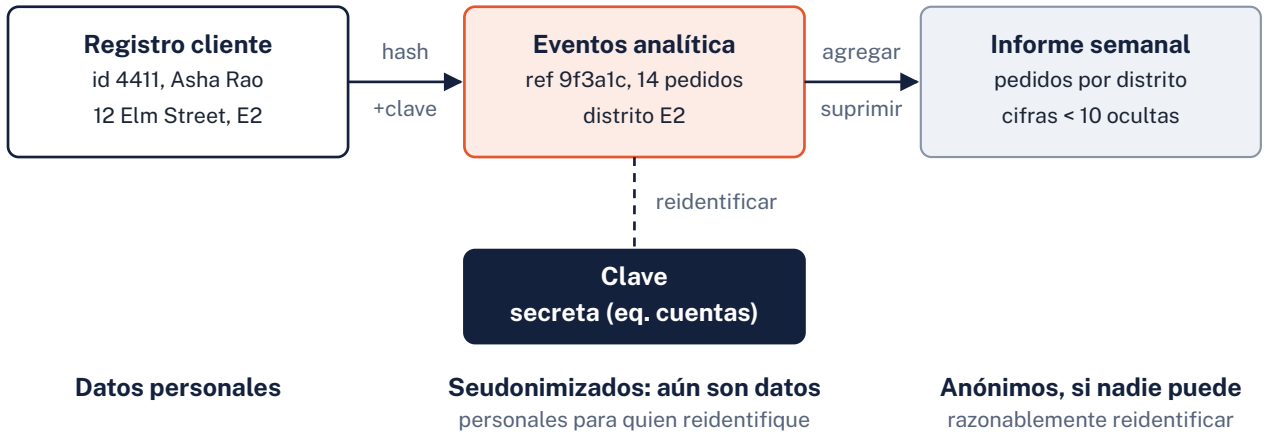
Minimización, seudonimización y anonimización

Los datos personales más baratos de proteger son los que nunca recopilas. La minimización se nota en decisiones pequeñas: un indicador de mayor de 18 años en lugar de una fecha de nacimiento, un distrito postal en lugar de una dirección completa para la analítica, un número de teléfono enmascarado que la plataforma retransmite en lugar de revelar al repartidor el número del cliente, y un muestreo de la ubicación solo mientras hay una entrega en curso.

La seudonimización (artículo 4(5)) trata los datos de manera que ya no puedan atribuirse a una persona sin información adicional, como una clave, que se guarda por separado y protegida. Un hash con clave (HMAC) del ID de cliente, cuya clave guarda el equipo de cuentas, permite al equipo de analítica contar los pedidos por cliente sin ver quiénes son los clientes. Es una salvaguarda valiosa que el artículo 25 menciona expresamente, pero el considerando 26 deja claro que los datos seudonimizados que podrían atribuirse a una persona mediante información adicional siguen siendo datos personales. En *SEPD/JUR* (asunto C-413/23 P, 4 de septiembre de 2025), el Tribunal de Justicia añadió un matiz: si los datos son personales se evalúa desde la perspectiva de quien los tiene, de modo que los datos seudonimizados pueden no ser datos personales para un destinatario que no tenga medios razonables de reidentificar a nadie, aunque siguen siendo datos personales para el responsable que guarda la clave, y el responsable debe seguir informando a las personas sobre esos destinatarios. Las Directrices 01/2025 del EDPB sobre seudonimización describen cómo hacerla bien.

La anonimización va más allá: los datos ya no se refieren a una persona identificable, teniendo en cuenta todos los medios que razonablemente puedan utilizarse para identificar a alguien, y entonces el RGPD deja de aplicarse. La anonimización real es difícil. Eliminar los nombres no basta cuando la combinación de código postal, fecha de nacimiento y sexo identifica a la mayoría de las personas, o cuando el rastro de ubicaciones de un solo repartidor revela dónde vive. La agregación con supresión de los recuentos pequeños, la generalización y, para las estadísticas, la privacidad diferencial son las técnicas habituales, y el resultado debería probarse frente a intentos realistas de reidentificación.

Datos personales, seudonimizados y anonimizados



La seudonimización reduce el riesgo, pero los datos siguen siendo personales para quien pueda revertirla; la anonimización debe resistir cualquier medio de reidentificación que razonablemente pueda utilizarse.

SI ESTO ES NUEVO PARA TI

Paso a paso: los nombres ocultos no son anónimos

La idea más difícil de esta sección es la diferencia entre datos seudonimizados y anonimizados. Ambos ocultan los nombres, pero, como explica la sección, el RGPD los trata de forma muy distinta. Sigue un estudio sobre la seguridad de los repartidores.

- Un registro de repartidor contiene un ID, un nombre y un número de teléfono: claramente datos personales.
- Para un estudio de seguridad vial, la plataforma sustituye el ID del repartidor por un hash con clave: un código de referencia generado con una clave secreta que el equipo de operaciones de repartidores guarda por separado. Los investigadores ven el código, un número de entregas y la zona de la ciudad, no nombres ni números de teléfono.
- Cualquiera que tenga la clave puede convertir un ID de repartidor en el mismo código y encontrar las filas de ese repartidor. Los datos están seudonimizados: los nombres están ocultos, pero todavía se puede identificar a las personas. Como la plataforma tiene la clave, se aplica la regla de la sección: los datos siguen siendo datos personales para la plataforma, incluidos sus investigadores.
- Aun así, merece la pena hacerlo: los investigadores pueden comparar cargas de trabajo sin ver quién es nadie.
- A continuación, la plataforma elabora un informe mensual de entregas por zona de la ciudad, que omite las zonas con menos de diez repartidores. Si nadie pudiera identificar a nadie a partir de él por ningún medio que razonablemente pueda utilizarse, el informe está anonimizado y, como explica la sección, el RGPD deja entonces de aplicársele.
- Eliminar solo los nombres a menudo no basta. El código postal, la fecha de nacimiento y el sexo juntos identifican a la mayoría de las personas, y el rastro de ubicaciones de un repartidor puede revelar su domicilio.
- **La lección:** un nombre oculto no es una persona anónima. La seudonimización reduce el riesgo, pero los datos siguen siendo personales para quien pueda reidentificar a las personas; llama anónimos a unos datos solo cuando nadie pueda razonablemente reidentificar a nadie a partir de ellos, y compruébalo frente a intentos realistas.

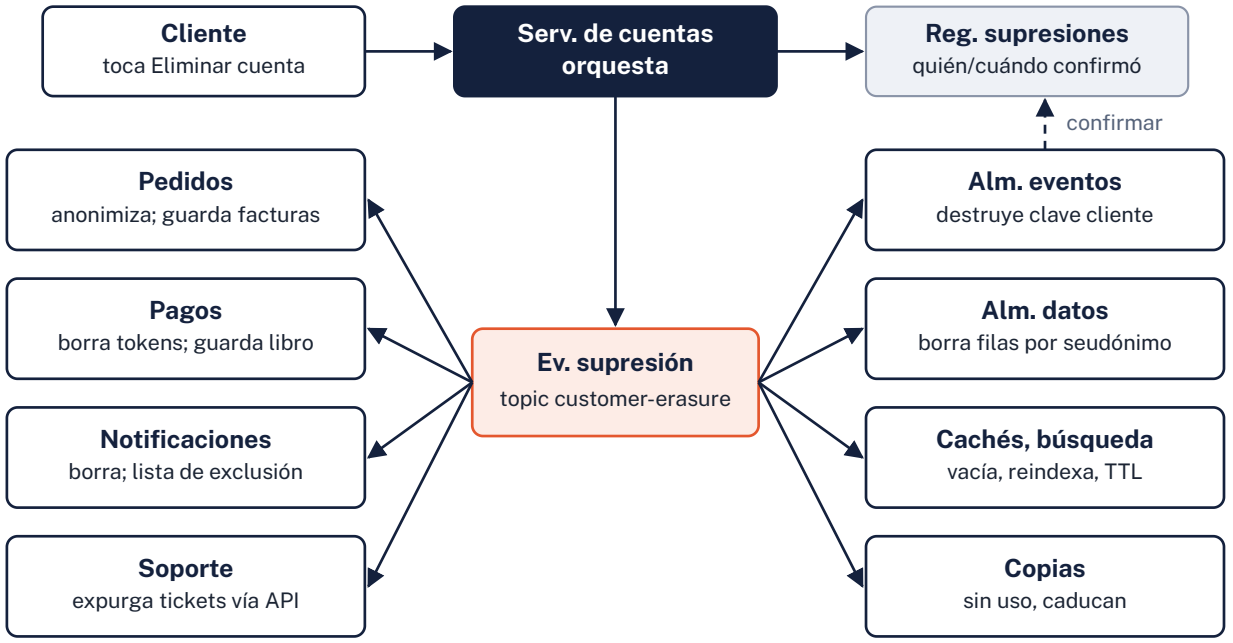
Conservación y supresión en todas las copias

La limitación del plazo de conservación necesita un calendario de conservación: para cada categoría de datos, cuánto tiempo se conservan, por qué y qué ocurre al final (eliminación, anonimización o archivo). Los calendarios solo funcionan cuando los sistemas los aplican automáticamente: TTL en las entradas de caché y de almacenes clave-valor, tablas

particionadas por tiempo cuyas particiones antiguas se eliminan, reglas de ciclo de vida del almacenamiento de objetos, ajustes de conservación de logs y tareas programadas cuyo éxito se monitoriza.

La supresión es más difícil en los lugares por los que se filtran los datos personales. **Las cachés y los índices de búsqueda** necesitan desalojo y reindexación explícitos, con TTL como red de seguridad. Los almacenes de **analítica** deberían guardar referencias seudónimas para que las filas puedan encontrarse y eliminarse, o bien los datos deberían anonimizarse al entrar. **Los logs de eventos y los almacenes con event sourcing** son inmutables por diseño; la técnica habitual es el crypto-shredding, que cifra los campos de cada persona con una clave propia de esa persona y destruye la clave, como se describe en la Sección 7.5, o bien mantener los datos personales fuera de los eventos y almacenar solo referencias. **Los logs de aplicación** no deberían contener datos personales innecesarios, algo sobre lo que vuelve la Sección 8.8. **Las copias de seguridad** normalmente no pueden editarse; la Information Commissioner's Office (ICO) del Reino Unido acepta que los datos permanezcan en las copias de seguridad hasta que se sobrescriban según un calendario establecido, siempre que entretanto queden fuera de uso y se informe claramente a la persona de lo que ocurrirá, y si alguna vez se restaura una copia de seguridad, las supresiones deben volver a aplicarse, lo que exige conservar la lista de los ID suprimidos. Por último, hay que dar instrucciones de eliminación a los **encargados del tratamiento**, ya que solo actúan siguiendo las instrucciones del responsable (artículo 28), y hay que informar a los demás **destinatarios** (artículo 19), así que la supresión debe llegar a la pasarela de SMS, al servicio de soporte y a la plataforma de marketing a través de sus API.

Propagación de la supresión en la plataforma de reparto de comida



Cada almacén con datos del cliente debe actuar y confirmar que lo ha hecho

Una solicitud, muchos almacenes: cada propietario aplica su propia regla (eliminar, anonimizar, destruir la clave o conservar con un motivo) y confirma que ha terminado.

EJEMPLO PRÁCTICO

Ejemplo práctico: diseñar "eliminar mi cuenta" para la plataforma de reparto de comida

Un cliente de Londres pide eliminar su cuenta. El diseño debe suprimir lo que se puede suprimir, conservar lo que exige la ley con un motivo declarado, terminar con holgura dentro del plazo de un mes, sobrevivir a fallos a mitad del proceso y producir pruebas. Los plazos de conservación que aparecen a continuación son opciones ilustrativas para este ejemplo, no requisitos legales para ninguna empresa concreta.

1. Verificar y confirmar	La solicitud llega a la app desde una sesión iniciada; el servicio de cuentas exige una autenticación reciente, explica qué se eliminará y qué se conservará, y ofrece primero una exportación de los datos. Un breve periodo de reflexión (por ejemplo, 14 días) permite al cliente cancelar; los pedidos y reembolsos pendientes deben completarse antes.
2. Registrar la solicitud	El servicio de cuentas escribe una entrada en el registro de supresiones con un ID de solicitud, el ID de cliente y la lista de almacenes de los que se espera respuesta, tomada del mapa de datos. El registro no guarda más datos personales que el ID necesario para tramitar la solicitud.
3. Propagar	Publica `CustomerErasureRequested` a través de la bandeja de salida transaccional (outbox) en un topic. Cada consumidor es idempotente, porque la entrega es al menos una vez, y procesa el evento aunque el cliente ya estuviera eliminado en parte.
4. Aplicar las reglas de cada almacén	Pedidos sustituye el nombre, el teléfono y la dirección por un marcador de borrado (tombstone), pero conserva las líneas de pedido y las facturas durante el plazo de conservación fiscal, con la obligación legal registrada como motivo. Pagos pide al proveedor de pagos que elimine los tokens de tarjeta almacenados y conserva el libro de transacciones. Notificaciones elimina los datos de contacto y añade un hash del correo electrónico a la lista de exclusión. El almacén de eventos destruye la clave de datos del cliente. El almacén de datos elimina las filas por seudónimo en su siguiente ejecución diaria. Las cachés se vacían y el índice de búsqueda se actualiza.
5. Llegar a los encargados del tratamiento	Los conectores llaman a las API de eliminación del SaaS de soporte, la pasarela de SMS y la plataforma de marketing, y registran cada respuesta.
6. Confirmar y reclamar	Cada consumidor escribe una confirmación en el registro. Una tarea diaria lanza una alerta por cualquier almacén que no haya confirmado en siete días, muy por debajo del límite de un mes.
7. Gestionar las copias de seguridad	Las copias de seguridad de las bases de datos se conservan durante 35 días y no se usan para nada salvo la recuperación ante desastres. Los ID de los clientes suprimidos se guardan en una lista para que una restauración vuelva a aplicar todas las supresiones antes de reabrir el sistema.

8. Informar al cliente	Un mensaje final confirma que se ha completado, indica qué se ha conservado, por qué y hasta cuándo, y que las copias de seguridad caducarán en un plazo de 35 días.

El mapa de datos es el artefacto crítico: un almacén que falta en él es un almacén que se queda con los datos. Una función de aptitud puede hacer fallar la compilación si un servicio nuevo declara un campo de datos personales sin registrar un manejador de supresión.

Conservar con un motivo no es un fallo de la supresión. El artículo 17(3) enumera excepciones, entre ellas el cumplimiento de una obligación legal; el diseño conserva los datos mínimos necesarios para esa obligación y los elimina cuando la obligación termina.

Preguntas frecuentes

¿Basta con cifrar los datos personales para cumplir el RGPD?
No. El cifrado es una medida de seguridad importante, y puede que no haya que comunicar a las personas una filtración de datos cifrados, pero los principios, las bases jurídicas, los derechos y las normas de conservación se aplican a los datos cifrados igual que a cualquier otro.

¿Podemos conservarlo todo de forma anonimizada en lugar de eliminarlo?
Solo si es realmente anónimo, es decir, si nadie puede razonablemente reidentificar a las personas a partir de ello. Aplicar un hash a un ID o quitar los nombres suele producir datos seudonimizados, que siguen siendo datos personales para quien tenga los medios para reidentificar.

¿Quién decide los plazos de conservación: los arquitectos o los abogados?
Los abogados y el negocio deciden los plazos y los motivos; los arquitectos se aseguran de que cada almacén tenga un plazo, de que se aplique automáticamente y de que el sistema pueda demostrarlo.

COMPRUEBA LO QUE HAS ENTENDIDO

▼ **¿Por qué una sola columna booleana `marketing_consent` es un diseño débil?**

No puede mostrar cuándo, cómo y con qué texto se dio el consentimiento, algo que el artículo 7 exige al responsable demostrar, y pierde el historial de retiradas. Un registro de solo anexión de eventos de consentimiento aporta esa prueba.

▼ **El equipo de analítica recibe datos de pedidos identificados por un HMAC del ID de cliente. ¿Son datos personales?**

Para la plataforma, que tiene la clave, sí: son datos seudonimizados. Para un destinatario sin medios razonables de reidentificación puede que no lo sean, según SEPD/JUR, pero las obligaciones del responsable se mantienen.

▼ **¿Por qué una copia de seguridad restaurada pone en peligro las supresiones y qué lo evita?**

La restauración recupera datos que se suprimieron después de hacer la copia. Lo evita conservar una lista de los ID suprimidos y volver a aplicar las supresiones antes de usar el sistema restaurado.

▼ **¿Qué quiere decir el artículo 25(2) con protección de datos por defecto?**

Sin que la persona haga nada, solo se recopilan, tratan, almacenan y hacen accesibles los datos necesarios para cada finalidad; por ejemplo, compartir el perfil, que es opcional, está desactivado por defecto.

▼ **Inténtalo: Marketing quiere conservar para siempre el rastro GPS completo de cada repartidor "para analítica", con los nombres de los repartidores incluidos. Aplica la privacidad desde el diseño: ¿qué cuestionarías y qué diseño propondrías? Esto trata de arquitectura, no es asesoramiento jurídico.**

Cuestiona primero la finalidad: ¿qué pregunta necesita esto? La limitación del plazo de conservación y la minimización de datos desaconsejan guardarlo para siempre. Conserva los recorridos precisos solo mientras los necesiten el despacho y las disputas, y luego elimínalos automáticamente. Para la analítica, usa datos agregados, como velocidades por zona, sin nombres. Sustituir los nombres por ID es solo seudonimización y siguen siendo datos personales. Restringe el acceso por defecto, y realiza una EIPD junto con el modelo de amenazas si es probable que el seguimiento suponga un alto riesgo.

FUENTES

- Regulation (EU) 2016/679 (GDPR), EUR-Lex: https://eur-lex.europa.eu/eli/reg/2016/679/oj
- EDPB Guidelines 4/2019 on Article 25 Data Protection by Design and by Default: https://www.edpb.europa.eu/our-work-tools/our-documents/guidelines/guidelines-42019-article-25-data-protection-design-and_en
- ICO: Right to erasure: https://ico.org.uk/for-organisations/uk-gdpr-guidance-and-resources/individual-rights/individual-rights/right-to-erasure/
- SOC 2 services and report types (Schellman): https://www.schellman.com/soc-2

- An Introduction to SOC 2 (Accounting Today): https://www.accountingtoday.com/news/an-introduction-to-soc-2
- AICPA & CIMA: SOC suite of services: https://www.aicpa-cima.com/resources/landing/system-and-organization-controls-soc-suite-of-services
- ISO/IEC 27001:2022/Amd 1:2024: https://www.iso.org/standard/88435.html
- KPMG: ISO/IEC 27001:2022 overview: https://assets.kpmg.com/content/dam/kpmg/cn/pdf/en/2023/02/iso-iec-27001-2022.pdf
- HHS: Summary of the HIPAA Security Rule: https://www.hhs.gov/hipaa/for-professionals/security/laws-regulations/index.html
- HHS: HIPAA Security Rule NPRM fact sheet: https://www.hhs.gov/hipaa/for-professionals/security/hipaa-security-rule-nprm/factsheet/index.html
- California Attorney General: CCPA: https://oag.ca.gov/privacy/ccpa
- CPPA: Updated monetary thresholds in CCPA: https://www.cppa.ca.gov/regulations/cpi_adjustment.html
- CPPA: California finalizes regulations (23 September 2025): https://www.cppa.ca.gov/announcements/2025/20250923.html
- NIST OSCAL: https://pages.nist.gov/OSCAL/

SECCIÓN 8.8 DE 8.8 · INTERMEDIO · 32 MIN DE LECTURA · ÚLTIMA REVISIÓN: NOVIEMBRE DE 2026

Seguridad en operación: logging, detección y respuesta

ANTES DE LEER ESTO

Esta sección se basa en:

- 2.7 Seguridad, usabilidad y más
- 8.2 Modelado de amenazas
- 8.7 Privacidad desde el diseño

Términos anteriores que utiliza: Datos personales, Objetivo de tiempo de recuperación (RTO), Objetivo de punto de recuperación (RPO), Bandeja de salida transaccional (transactional outbox), Lista de materiales de software (SBOM)

LA IDEA EN BREVE

Un banco no confía solo en unas puertas sólidas. También tiene cámaras que graban quién entró, alarmas que suenan cuando ocurre algo inusual, un plan de lo que hace el personal durante un atraco y dinero de reserva guardado bajo llave en otro lugar por si se pierde la cámara acorazada principal. Operar el software de forma segura funciona igual. Los sistemas llevan un registro cuidadoso de las acciones importantes, vigilan esos registros en busca de señales de ataque, corrigen rápidamente las debilidades recién descubiertas, siguen un plan ensayado cuando algo sale mal, guardan copias de seguridad que los atacantes no pueden alcanzar y avisan a tiempo a los reguladores y a los clientes cuando se ven afectados datos personales.

LO QUE NECESITAS SABER

- Da por hecho que habrá brechas: diseña para detectar, contener y recuperarte, con escenarios de seguridad que fijen tiempos de respuesta medibles.
- Registra los eventos de seguridad (autenticación, fallos de autorización, acciones administrativas y de alto riesgo) con quién, qué, cuándo y dónde; nunca registres contraseñas, tokens, claves, datos de tarjetas ni datos personales innecesarios; protege los logs con un almacenamiento separado de escritura única.
- Una pista de auditoría es un registro completo, inmutable y atribuible de acciones definidas, emitido por el servicio propietario cuando la acción se confirma.
- Un SIEM correlaciona la telemetría y genera alertas según reglas derivadas del modelo de amenazas y de MITRE ATT&CK; las reglas deben versionarse, probarse y ajustarse para evitar la fatiga de alertas.

- Prioriza las vulnerabilidades según su explotación conocida (KEV), su probabilidad (EPSS), su alcanzabilidad y su exposición, no solo según la puntuación base de CVSS, y acuerda de antemano los plazos de corrección.
- NIST SP 800-61 Revisión 3 (abril de 2025) enmarca la respuesta a incidentes en las funciones del CSF 2.0; la resiliencia frente al ransomware depende de copias de seguridad inmutables con credenciales separadas y de restauraciones probadas.
- Según el artículo 33 del RGPD, las violaciones de datos personales deben notificarse a la autoridad de control sin dilación indebida y, si es posible, en un plazo de 72 horas desde que se tenga constancia de ellas, y a las personas afectadas según el artículo 34 cuando el riesgo sea alto.

La seguridad no termina en el despliegue

Las secciones anteriores de este capítulo buscan prevenir los ataques. Ninguna prevención es perfecta, así que las organizaciones maduras también diseñan partiendo de que se producirá una brecha: quieren detectarla rápido, limitar sus daños, recuperarse limpiamente y aprender de ella. El Capítulo 2 describió las tácticas de seguridad como resistir los ataques, detectarlos, reaccionar ante ellos y recuperarse de ellos, y mostró cómo escribir escenarios de seguridad con respuestas medibles como *una credencial filtrada se detecta y se revoca en un plazo de 15 minutos*. Esta sección trata de la arquitectura y las prácticas que hacen posibles esas respuestas.

Gran parte de ello se apoya en los fundamentos operativos de la observabilidad y la gestión de incidentes del Capítulo 9. La diferencia es el adversario: un disco que falla no borra su rastro, no elimina sus logs ni vuelve la semana siguiente por una puerta que dejó abierta.

Logging de seguridad: qué registrar y qué no registrar nunca

Los logs de seguridad existen para responder, a posteriori, quién hizo qué, sobre qué recurso, cuándo, desde dónde y con qué resultado. La OWASP Logging Cheat Sheet es la referencia estándar para los profesionales. Recomienda registrar eventos como los inicios de sesión correctos y fallidos, los fallos de autorización, los fallos de gestión de sesiones, los fallos de validación de entradas, los errores de la aplicación, las funciones de alto riesgo, como las acciones administrativas y las exportaciones de datos, y el uso de los propios controles de seguridad. Cada evento debería incluir cuándo (una marca de tiempo sincronizada), dónde (aplicación, servicio, host, entorno), quién (la identidad autenticada, la dirección de origen) y qué (la acción, el objeto, el resultado y el motivo).

Igual de importante es lo que debe quedar fuera. Los logs se copian mucho, se conservan durante mucho tiempo y los leen muchas personas y herramientas, así que son un lugar predilecto por el que se filtran secretos y datos personales. Nunca registres contraseñas, identificadores de sesión ni tokens de acceso (registra un hash o un ID de clave si necesitas correlacionar), claves de cifrado ni cadenas de conexión, datos de tarjetas de pago ni datos personales de categorías especiales. Trata los datos de los logs como datos personales cuando identifiquen a personas, y aplica las reglas de minimización y conservación de la Sección 8.7. La ocultación de datos sensibles se aplica mejor de forma centralizada en el pipeline de logs además de en el código, porque una sola instrucción de depuración olvidada puede deshacerla.

Registra esto, nunca registres aquello (según la OWASP Logging Cheat Sheet)

Registrar	Excluir, enmascarar o aplicar hash
Inicios de sesión correctos y fallidos, desafíos de MFA, restablecimientos de contraseña	Contraseñas, incluso las incorrectas (una contraseña mal tecleada suele ser casi la correcta)
Fallos de autorización (acceso denegado), sobre todo los repetidos	Identificadores de sesión, tokens de acceso, tokens de actualización, claves de API (en su lugar, registra un hash o el ID de la clave)
Fallos de validación de entradas y solicitudes rechazadas por el gateway	Claves de cifrado, cadenas de conexión a bases de datos y otros secretos
Acciones administrativas: cambios de rol, cambios de políticas, creación de usuarios, cambios de configuración	Números completos de tarjetas de pago, códigos de verificación de tarjetas, datos de cuentas bancarias
Acciones de negocio de alto riesgo: reembolsos, pagos a terceros, exportaciones de datos, eliminación de cuentas	Datos de salud, identificadores oficiales y otros datos personales de categorías especiales
Errores y excepciones relevantes para la seguridad; arranque y parada de los controles de seguridad	Datos personales más allá de lo que necesita una investigación, y cualquier cosa de la que un usuario haya pedido quedar excluido
Uso de acceso de emergencia (break-glass)	Código fuente y datos con una clasificación superior a la que el sistema de logging está autorizado a guardar

```
{
  "ts": "2026-10-21T19:42:07.318Z",
  "event": "authz.denied",
  "service": "orders",
  "env": "prod",
  "actor": {"type": "user", "id": "cust_4411", "session": "sha256:7c1e..."},
  "client": {"ip": "203.0.113.24", "ua_family": "iOS app 7.3"},
  "action": "GET /orders/81723",
  "resource_owner": "cust_9902",
  "reason": "not_owner",
  "trace_id": "4bf92f3577b34da6a3ce929d0e0e4736"
}
```

Un evento de seguridad estructurado del servicio de pedidos. Muestra quién intentó qué y por qué se rechazó, con un ID de traza para la correlación, pero sin token ni nombre de cliente.

Los logs también deben protegerse de los atacantes, cuyo primer paso tras conseguir acceso suele ser borrar las pruebas. Envía los logs de seguridad sin demora a una cuenta o servicio separado que las propias credenciales de la aplicación no puedan modificar; almacénalos con conservación de escritura única; restringe y registra el acceso a ellos; y mantén los relojes sincronizados para que los eventos de distintos sistemas cuadren. Trata el contenido de los logs como una entrada no confiable: un atacante que pueda introducir un salto de línea en un campo registrado puede falsificar entradas (inyección en logs, log injection), así que codifica o rechaza los caracteres de control, y recuerda que Log4Shell se desencadenaba al registrar texto controlado por el atacante.

Pistas de auditoría

Una pista de auditoría es un tipo de log más acotado y más sólido: un registro completo, cronológico y a prueba de manipulaciones de acciones definidas, que se conserva para establecer la rendición de cuentas. Los logs de depuración se muestrean, se rotan y se cambian cuando les parece a los desarrolladores; una pista de auditoría debe ser completa dentro de su alcance (cada reembolso, cada cambio de rol, cada acceso al expediente de soporte de un cliente), inmutable, atribuible a una identidad autenticada y conservada durante un periodo definido. Las normativas y los estándares a menudo la exigen; PCI DSS, por ejemplo, exige conservar el historial de logs de auditoría durante al menos doce meses, con los tres meses más recientes disponibles de inmediato para su análisis.

Desde el punto de vista arquitectónico, diseña la pista de auditoría como un producto del dominio, no como un subproducto del logging. Emite los eventos de auditoría desde el servicio propietario de la acción, en el momento en que la acción se confirma (la bandeja de salida transaccional garantiza que ocurran ambas cosas o ninguna), hacia un almacén de solo anexión con acceso de escritura restringido. La evidencia de manipulación puede añadirse mediante el

encadenamiento de hashes, en el que cada registro incluye un hash del anterior, o anclando periódicamente un resumen firmado en otro lugar. Los sistemas con event sourcing obtienen gran parte de esto gratis, como mostró la Sección 7.5. Los logs de auditoría del plano de control de los proveedores de nube, que registran cada llamada a la API hecha sobre la cuenta en la nube, forman parte del mismo panorama y deben activarse en todas las cuentas y regiones.

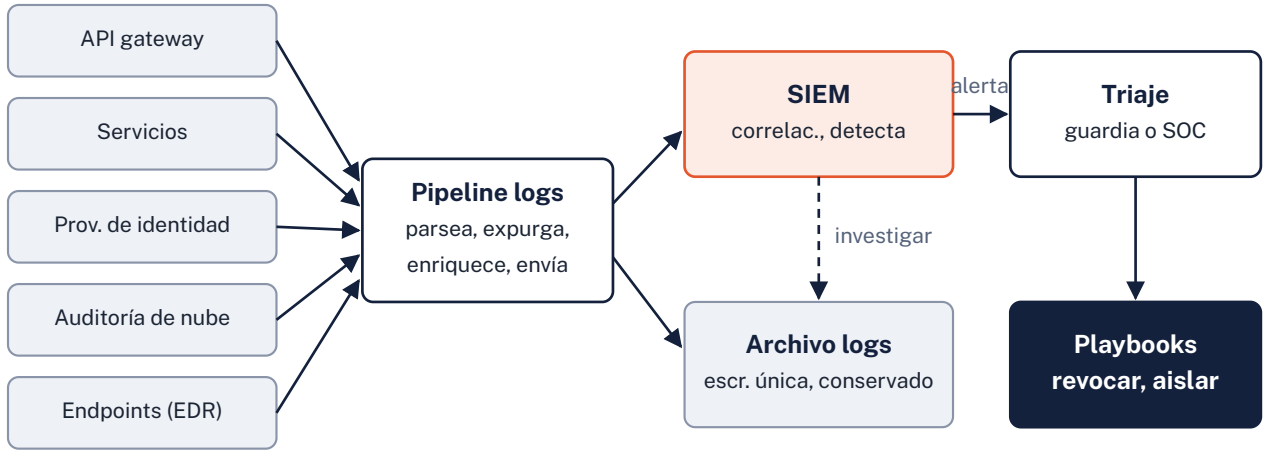
Detección: SIEM, reglas y automatización de la respuesta

Un sistema de gestión de información y eventos de seguridad (SIEM) recoge la telemetría de seguridad de toda la infraestructura, la normaliza, correlaciona los eventos y genera alertas cuando se cumplen los patrones de las reglas de detección: muchos inicios de sesión fallidos seguidos de uno correcto, una cuenta de servicio usada desde un país nuevo, un rol al que se conceden derechos de administrador fuera de una ventana de cambios, una exportación grande a las tres de la madrugada. Las fuentes típicas son el API gateway, los eventos de seguridad de las aplicaciones, el proveedor de identidad, los logs de auditoría de la nube, los logs de auditoría de Kubernetes y los agentes de detección y respuesta en endpoints (EDR) de los portátiles y servidores del personal.

Una buena detección se diseña, no se compra. Deriva las reglas del modelo de amenazas y de MITRE ATT&CK, una base de conocimiento pública de las tácticas y técnicas de atacantes reales que MITRE mantiene desde 2015; escríbelas como código en un formato compartido como Sigma, versiónalas y pruébalas simulando el ataque que deben detectar. Mide la calidad de las alertas: una regla que salta cincuenta veces al día y siempre resulta inofensiva enseña a la gente a ignorarla, y la fatiga de alertas es la forma en que se pasan por alto los ataques reales. Ayudan las trampas baratas y de señal alta: los honeytokens, como una clave de acceso falsa o un registro de cliente falso que ningún proceso legítimo toca nunca, casi no producen falsas alarmas. Después, las herramientas de orquestación, automatización y respuesta de seguridad (SOAR) ejecutan playbooks para las alertas bien conocidas, como deshabilitar un usuario, revocar una clave o aislar un contenedor, para que las personas se ocupen de las decisiones que requieren criterio.

La ingesta del SIEM suele cobrarse por volumen, lo que tienta a los equipos a descartar logs. Un diseño por niveles ayuda: los eventos de seguridad de alto valor van al SIEM para la detección en tiempo real; los logs voluminosos van a un almacenamiento de escritura única más barato, donde pueden consultarse durante una investigación.

Un pipeline de detección de seguridad



Reglas de detección como código: versionadas, probadas con ataques y ajustadas para que cada alerta importe

La telemetría se recoge y se expurga una sola vez, y después se usa tanto para la detección en tiempo real como, desde un archivo de escritura única, para la investigación.

Gestión de vulnerabilidades

Cada día se publican nuevas vulnerabilidades en el software que ejecutas. Un identificador CVE (Common Vulnerabilities and Exposures), como CVE-2021-44228, da a cada vulnerabilidad conocida públicamente un nombre único; el programa, que MITRE inició en 1999, asigna identificadores a través de cientos de autoridades de numeración CVE (CVE Numbering Authorities), entre ellas grandes fabricantes y fundaciones de código abierto. La National Vulnerability Database de EE. UU. añade análisis como los identificadores de los productos afectados y las puntuaciones, aunque un retraso acumulado en ese enriquecimiento desde 2024, y un breve susto en abril de 2025, cuando estuvo a punto de vencer el contrato que financia el programa CVE, mostraron el riesgo de depender de una sola fuente; la mayoría de las organizaciones combinan ahora varias fuentes, entre ellas la base de datos de avisos de GitHub y la base de datos de código abierto OSV.

CVSS (el Common Vulnerability Scoring System, mantenido por FIRST; la versión 4.0 se publicó en noviembre de 2023) califica la gravedad de 0 a 10, con los niveles baja (0.1 a 3.9), media (4.0 a 6.9), alta (7.0 a 8.9) y crítica (9.0 a 10.0). La puntuación base publicada describe la vulnerabilidad en general; las métricas de amenaza y de entorno de la versión 4.0 la ajustan según la actividad de explotación y según tu propio entorno. Una puntuación base no es una medida de tu riesgo. Dos señales mejoran considerablemente la priorización: el Exploit Prediction Scoring System (EPSS, también de FIRST), que estima la probabilidad de que una vulnerabilidad se explote en los próximos 30 días, y el catálogo Known Exploited Vulnerabilities (KEV) de la Cybersecurity and Infrastructure Security Agency de EE. UU., que

enumera las vulnerabilidades con pruebas de explotación real. Combínalas con la alcanzabilidad (¿se usa la función vulnerable?) y la exposición (¿el servicio está expuesto a internet?), y la mayoría de los cientos de hallazgos de un escáner pasan a la cola rutinaria, mientras que un puñado exige actuar hoy.

Acuerda de antemano los plazos de corrección, para que cada hallazgo no se convierta en una negociación, y mide el tiempo de corrección. Las cifras siguientes son ilustraciones típicas, no un estándar; los entornos regulados pueden imponer las suyas; por ejemplo, PCI DSS exige instalar los parches de seguridad críticos en el plazo de un mes desde su publicación.

Ejemplos de plazos de corrección

Hallazgo	Plazo
Explotación conocida (por ejemplo, en el catálogo KEV de CISA), expuesto a internet	Mitigar en un plazo de 24 a 72 horas; aplicar el parche en cuanto exista una corrección
Gravedad crítica, alcanzable, sin explotación conocida	En un plazo de 7 a 14 días
Gravedad alta	En un plazo de 30 días
Gravedad media	En un plazo de 90 días, o en la siguiente actualización rutinaria
Gravedad baja, o no alcanzable	Actualizaciones rutinarias; anotar el motivo si se aplaza

GUÍA DE DECISIÓN

Guía de decisión: clasificar una vulnerabilidad recién publicada

Úsala cuando un escáner, un aviso o una noticia mencione una vulnerabilidad en algo que podrías estar usando.

Q1 **¿Algún artefacto desplegado contiene el componente y la versión vulnerables (consulta el almacén de SBOM)?**

- **Sí** Pasa a la siguiente pregunta y anota cada servicio afectado y su propietario.
- **No** Anota "no afectado" con las pruebas; sigue atento a nuevas versiones afectadas.

Q2 **¿Se sabe que se está aprovechando activamente (por ejemplo, figura en el catálogo KEV de CISA) o hay código de explotación funcional público?**

- **Sí** Vía de emergencia: mitigar en un plazo de horas a días (aplicar un parche, desactivar la funcionalidad o bloquear en el gateway o el WAF) y después aplicar el parche como es debido.
- **No** Pasa a la siguiente pregunta.

Q3 **¿El código vulnerable es alcanzable en tu uso, y el servicio está expuesto a internet o maneja datos sensibles?**

- **Sí** Corrígelo dentro del SLA correspondiente a su gravedad, ajustado con las métricas de entorno de CVSS y con EPSS.
- **No** Pasa a la siguiente pregunta.

Q4 **¿Existe un control compensatorio que bloquee por completo la explotación (funcionalidad desactivada, sin ruta de red)?**

- **Sí** Documéntalo (una declaración VEX encaja aquí) y aplica el parche en el ciclo de actualizaciones rutinario.
- **No** Aplica el parche en el ciclo rutinario y vuelve a revisarlo si se informa de su explotación.

SI ESTO ES NUEVO PARA TI

Paso a paso: la puntuación más alta no siempre va primero

La idea que requiere más atención aquí es ordenar las vulnerabilidades según tu propio riesgo y no según su puntuación. La Sección 8.2 presentó CVSS: su puntuación base describe un defecto en general, no el riesgo que supone para ti. Sigue dos hallazgos que llegan la misma mañana.

- El hallazgo A es un defecto crítico, con una puntuación base de 9.1, en una biblioteca incluida en el servicio de pedidos. El hallazgo B es uno alto, con una puntuación base de 7.5, en software que ejecuta el API gateway. Ordenados por puntuación, A va primero.
- La primera comprobación es si el componente está realmente desplegado. El almacén de SBOM muestra que ambos lo están, y el equipo anota cada servicio afectado y su propietario.
- Después viene la explotación conocida. B figura en el catálogo Known Exploited Vulnerabilities que mantiene CISA, una agencia del Gobierno de EE. UU.: hay pruebas de que los atacantes lo están usando. Nada indica que alguien esté explotando A.
- Luego, la alcanzabilidad y la exposición. El gateway está expuesto a internet, así que los atacantes pueden alcanzar B directamente. A está en una parte de la biblioteca a la que el servicio de pedidos nunca llama.
- Así que B toma la vía de emergencia: mitigar en un plazo de horas a días, por ejemplo desactivando la funcionalidad afectada o bloqueando el ataque en el borde, y después aplicar el parche como es debido. A se anota como no alcanzable, con las pruebas, y se corrige en el ciclo de actualizaciones rutinario, para volver a revisarlo si se informa de su explotación.
- Para los muchos hallazgos que ni figuran en el catálogo ni son claramente inalcanzables, EPSS ayuda a ordenarlos: estima la probabilidad de que cada uno se explote en los próximos 30 días.
- **La lección:** ordena las vulnerabilidades según su explotación conocida, su probabilidad, su alcanzabilidad y su exposición, no solo según la puntuación base, y acuerda de antemano los plazos de corrección para que cada hallazgo no sea una nueva negociación.

Pruebas de penetración, bug bounties y divulgación

Una prueba de penetración es un intento autorizado y acotado en el tiempo, realizado por especialistas, de entrar en un sistema, idealmente conociendo su arquitectura y con un alcance acordado por escrito. Encuentra tipos de problemas que los escáneres no detectan, como la autorización a nivel de objeto rota o el abuso de la lógica de negocio (canjear el mismo vale dos

veces mediante una condición de carrera). Encarga una antes de un lanzamiento importante, después de un cambio arquitectónico significativo y de forma periódica; un ejercicio de red team va más allá y pone a prueba la detección y la respuesta simulando a un adversario real durante semanas.

Los investigadores externos también encontrarán problemas, y necesitan una forma segura de comunicarlos. Una política de divulgación de vulnerabilidades indica cómo informar, qué pruebas están permitidas y que la investigación de buena fe no recibirá amenazas legales; un archivo `security.txt` en `/.well-known/security.txt` (RFC 9116, 2022) indica a los investigadores adónde enviar los informes. Un bug bounty añade un pago por los hallazgos válidos, a menudo a través de una plataforma que clasifica los envíos. ISO/IEC 29147 (divulgación) e ISO/IEC 30111 (gestión) describen los procesos, y el Reglamento de Ciberresiliencia (Cyber Resilience Act) exige a los fabricantes de productos incluidos en su ámbito tener una política de divulgación coordinada de vulnerabilidades. Un bug bounty no sustituye al trabajo interno de seguridad: si se lanza sobre un sistema sin probar, suele producir una avalancha de informes que el equipo no puede gestionar.

Respuesta a incidentes

La respuesta a incidentes es la gestión organizada de un incidente de seguridad, desde la detección hasta la recuperación y el aprendizaje. Durante más de una década, la referencia estándar fue NIST SP 800-61 Revisión 2 (2012), con su ciclo de vida de cuatro fases: preparación; detección y análisis; contención, erradicación y recuperación; y actividad posterior al incidente. La Revisión 3, publicada en abril de 2025 como *Incident Response Recommendations and Considerations for Cybersecurity Risk Management: A CSF 2.0 Community Profile*, sustituye ese ciclo de vida por las seis funciones del NIST Cybersecurity Framework 2.0. Gobernar, Identificar y Proteger son una preparación continua que hace los incidentes menos probables y la respuesta más eficaz; Detectar, Responder y Recuperar son la respuesta propiamente dicha; y las lecciones aprendidas alimentan la mejora de todas ellas en lugar de ser una fase final.

En la práctica, una buena respuesta necesita roles con nombre (un coordinador del incidente que coordina y decide, responsables técnicos que investigan y actúan, un responsable de comunicación, un escriba que lleva la cronología, y acceso rápido a la asesoría jurídica y al delegado de protección de datos), niveles de gravedad que determinen a quién se despierta y un canal fuera de banda, porque un atacante que esté dentro de tu sistema de correo electrónico o de chat puede leer la respuesta. La contención es un compromiso: cortar el paso a un atacante rápidamente limita los daños, pero puede destruir pruebas y alertarlo, así que conserva las pruebas (snapshots, logs, capturas de memoria) antes de reconstruir, y asegúrate

de haber encontrado todos sus puntos de apoyo antes de anunciar que se ha ido. Después, una revisión sin culpas se pregunta cómo permitieron el incidente el sistema y sus procesos, y produce acciones arquitectónicas.

La respuesta a incidentes en NIST SP 800-61 Revisión 3



La 4.ª fase de la Rev. 2, actividad posincidente, pasa a ser mejora continua en la Rev. 3

La Revisión 3 asigna la respuesta a incidentes a las funciones del CSF 2.0 en lugar de a un ciclo de vida de cuatro fases.

Resiliencia frente al ransomware

El ransomware cifra los sistemas de una organización y exige un pago a cambio de la clave; la mayoría de los grupos ahora también roban antes los datos y amenazan con publicarlos. Las defensas arquitectónicas son las de todo este capítulo (MFA, mínimo privilegio, segmentación, parches, detección), pero la recuperación depende sobre todo de una cosa: copias de seguridad que el atacante no pueda alcanzar. Los atacantes con derechos de administrador buscan primero las copias de seguridad y las eliminan o las cifran.

Conserva al menos una copia de los datos críticos inmutable o fuera de línea: almacenamiento de objetos con bloqueos de conservación de escritura única que ni siquiera un administrador puede acortar, o soportes desconectados físicamente de la red. Pon las copias de seguridad en una cuenta o tenencia separada, con credenciales y MFA propias, para que comprometer las identidades de producción no llegue hasta ellas. La regla tradicional 3-2-1 (tres copias, en dos tipos de soporte, una fuera de las instalaciones) es una base razonable. Sobre todo, prueba las restauraciones con regularidad frente al RTO y el RPO que haya acordado el negocio, incluido el orden de recuperación: los servicios de identidad y el DNS van primero, porque sin ellos no se puede reconstruir nada más. En 2017, la naviera Maersk reconstruyó su red global después

de que el malware NotPetya destruyera sus controladores de dominio de Windows; la recuperación dependió del único controlador de dominio que sobrevivió, en Ghana, porque un corte de electricidad lo había dejado desconectado durante el ataque.

Violaciones de datos personales y plazos de notificación

Una violación de datos personales es, según el artículo 4(12) del RGPD, una violación de la seguridad que ocasione la destrucción, pérdida o alteración accidental o ilícita de datos personales, o la comunicación o el acceso no autorizados a ellos. Es más amplia que el robo: un ataque de ransomware que deja los datos indisponibles, o un correo electrónico enviado a los destinatarios equivocados, puede serlo. Según el artículo 33, el responsable debe notificarla a la autoridad de control sin dilación indebida y, si es posible, en un plazo de 72 horas desde que tenga constancia de ella, a menos que sea improbable que constituya un riesgo para los derechos y libertades de las personas; las notificaciones posteriores deben indicar los motivos del retraso. Según el artículo 34, el responsable también debe comunicarla sin dilación indebida a las personas afectadas si es probable que entrañe un alto riesgo para ellas. El artículo 33(5) exige documentar todas las violaciones, se notifiquen o no. El UK GDPR tiene las mismas normas, con las notificaciones dirigidas a la Information Commissioner's Office.

Puede haber otros regímenes en paralelo, con sus propios plazos, destinatarios y umbrales, y los contratos suelen añadir obligaciones frente a clientes y socios. La tabla resume los principales plazos de la UE y del Reino Unido. La propuesta Digital Omnibus de la Comisión Europea de noviembre de 2025 ampliaría el plazo del RGPD a 96 horas y limitaría la notificación a la autoridad a las violaciones de alto riesgo, pero en el momento de redactar esto no se ha adoptado y se aplica la regla de las 72 horas.

La lección arquitectónica es que un plazo contado en horas solo puede cumplirse si los sistemas pueden responder, rápido y con pruebas, qué datos se vieron afectados, de quién eran y si realmente se accedió a ellos. Eso requiere el mapa de datos de la Sección 8.7, logs de acceso a datos en los almacenes de datos personales y un cifrado que pueda hacer ininteligible una copia filtrada. Como en el resto de este capítulo, esto es información sobre cómo afecta la ley a los sistemas, no asesoramiento jurídico; la decisión de notificar corresponde al responsable, asesorado por su delegado de protección de datos (DPD) y sus abogados.

Plazos de notificación que se aplican habitualmente

Norma	Quién informa a quién	Plazo
RGPD / UK GDPR art. 33	Del responsable a la autoridad de control (en el Reino Unido, la ICO)	Sin dilación indebida y, si es posible, en un plazo de 72 horas desde que se tenga constancia, a menos que sea improbable que la violación constituya un riesgo para las personas. La información puede facilitarse por fases.
RGPD / UK GDPR art. 33(2)	Del encargado al responsable	Sin dilación indebida tras tener constancia.
RGPD / UK GDPR art. 34	Del responsable a las personas afectadas	Sin dilación indebida cuando sea probable que la violación entrañe un alto riesgo, salvo, por ejemplo, que los datos estuvieran protegidos con un cifrado que los haga ininteligibles.
Directiva NIS2 (UE) 2022/2555	De las entidades esenciales e importantes al CSIRT o a la autoridad	Alerta temprana en un plazo de 24 horas desde que se tenga constancia de un incidente significativo, notificación en un plazo de 72 horas, informe final en el plazo de un mes.
Reglamento de Ciberresiliencia, art. 14	Los fabricantes, a través de la plataforma única de notificación de ENISA	Para vulnerabilidades aprovechadas activamente e incidentes graves: alerta temprana en un plazo de 24 horas, notificación en un plazo de 72 horas (se aplica desde el 11 de septiembre de 2026).

Runbooks

Un runbook (manual de operaciones) es un procedimiento escrito y probado para una situación concreta: revocar y rotar una credencial filtrada, aislar un contenedor comprometido, restaurar la base de datos de pedidos a partir de una copia de seguridad inmutable, responder a una sospecha de violación de datos personales. Un buen runbook indica su desencadenante, las personas a las que hay que implicar, los pasos y comandos exactos, cómo verificar que cada paso ha funcionado, cómo deshacerlo y qué pruebas conservar. Los runbooks son código en espíritu: guárdalos en el control de versiones junto a los sistemas que describen, automatiza las partes mecánicas y ensáyalos en game days y en ejercicios de simulación de mesa (tabletop), porque un procedimiento que nunca se ha ejecutado suele fallar en el tercer paso. Cada regla de detección debería enlazar con el runbook de su alerta.

Ejemplo práctico: las primeras 24 horas de un incidente de filtración de credenciales en la plataforma de reparto de comida

Tiempo	Qué ocurre	Por qué importa
09:12	Un ingeniero sube a un repositorio público un commit que contiene una clave de acceso a la nube de larga duración para el rol de despliegue de producción.	La causa habitual: un script de depuración, no mala fe. Hay escáneres automáticos que vigilan los commits públicos, así que hay que dar por hecha la exposición de inmediato.
09:14	La alerta de detección de secretos llega al ingeniero de seguridad de guardia; a las 09:16 el SIEM muestra llamadas a la API con la clave desde una dirección desconocida.	Detectarlo en minutos depende de tener la detección de secretos y los logs de auditoría de la nube llegando al SIEM antes del incidente.
09:20	Se declara el incidente con la gravedad máxima. Se nombra a un coordinador del incidente, un escriba y un responsable técnico; se abre un canal dedicado.	Unos roles claros evitan que cinco personas hagan lo mismo y que nadie haga lo urgente.
09:25	Contener: se desactiva la clave, se revocan las sesiones emitidas con ella y se asocia al rol una política que lo deniega todo hasta reconstruirlo.	Desactivar, y no eliminar, conserva el ID de la clave para las búsquedas en los logs. Eliminar el commit no es contención: la clave ya se ha copiado.
09:45	Alcance: el equipo consulta el log de auditoría de la nube en busca de cada llamada hecha con el ID de la clave, desde el momento en que se creó, no solo desde la subida.	La clave puede haberse filtrado antes por otra vía; el log responde qué se hizo, desde dónde y cuándo.
10:30	Hallazgos: el atacante listó los buckets de almacenamiento, arrancó 40 instancias de cómputo grandes en una región sin usar y creó un usuario nuevo con su propia clave de acceso.	Los atacantes establecen primero la persistencia; limpiar solo la clave original los dejaría dentro.
11:00	Erradicar: instancias terminadas, el usuario nuevo y su clave eliminados, las políticas cambiadas recientemente revisadas y revertidas; antes se toman snapshots como prueba.	Conserva las pruebas antes de destruirlas; se necesitan para la revisión, para las aseguradoras y quizá para los reguladores.

Tiempo	Qué ocurre	Por qué importa
11:45	La cuestión de los datos: los logs de acceso a nivel de objeto muestran descargas correctas del bucket que contiene las exportaciones de clientes. Se incorporan el DPD y la asesoría jurídica.	Sin logs a nivel de objeto, el equipo no podría demostrar que los datos no se leyeron y tendría que suponer que sí.
12:30	El DPD evalúa que se trata de una violación de datos personales que probablemente entrañe un riesgo; se considera que el plazo de 72 horas corre desde el momento en que se tuvo constancia. Se redacta una primera notificación.	El artículo 33 permite facilitar la información por fases; una notificación tardía necesita motivos. Que haya que informar a las personas depende del nivel de riesgo.
15:00	Recuperar: el rol de despliegue se reconstruye con credenciales de corta duración mediante federación OIDC desde la CI; se añaden alarmas de facturación y restricciones de región; se contacta con el proveedor por los cargos fraudulentos.	Corrige el tipo de problema, no solo el caso concreto: sin claves de larga duración, no hay nada que filtrar.
18:00	Protección de subidas frente a secretos activada en todos los repositorios; todos los demás secretos que eran legibles desde la máquina del mismo ingeniero se rotan como precaución.	Da por hecho que las credenciales relacionadas también pueden estar expuestas hasta que se demuestre lo contrario.
Día siguiente, 09:00	Informe de situación a la dirección; notificación a la autoridad de control presentada dentro de las 72 horas con lo que se sabe; revisión sin culpas programada en un plazo de cinco días hábiles.	La revisión se pregunta por qué existía una clave de larga duración y por qué se pudo subir en un commit, no quién la subió.

La cronología muestra por qué la operación forma parte de la arquitectura. Cada paso que salió bien dependió de una decisión de diseño tomada meses antes: la detección de secretos y los logs de auditoría alimentando el SIEM, los logs de acceso a nivel de objeto en el bucket de exportaciones, un mapa de datos que indicó al DPD de quién eran los datos que contenía, y un diseño de identidades que permitió reconstruir el rol de despliegue sin claves de larga duración. Cada paso que salió mal (que existiera siquiera una clave de larga duración, un bucket de exportaciones de clientes que ya nadie necesitaba) fue también una decisión arquitectónica.

Lista de comprobación: ¿está el sistema preparado para ser atacado?

- [ ] **Los eventos de seguridad se registran, estructurados y centralizados**
  Autenticación, fallos de autorización, acciones administrativas y de alto riesgo, con quién, qué, cuándo, dónde y un ID de traza; sin secretos ni datos personales innecesarios.
- [ ] **Los logs están protegidos y se conservan**
  Cuenta o almacenamiento separados con conservación de escritura única, acceso restringido, relojes sincronizados y un plazo de conservación definido.
- [ ] **Los logs de auditoría de la nube y de acceso a datos están activados**
  Incluido el acceso a nivel de objeto en los almacenes de datos personales o de pago, para poder demostrar el alcance.
- [ ] **Existen detecciones para tus principales amenazas**
  Reglas derivadas del modelo de amenazas, probadas con ataques simulados, con un propietario y un runbook para cada alerta.
- [ ] **Los SLA de vulnerabilidades están acordados y se miden**
  Priorizadas según explotación, alcanzabilidad y exposición; el almacén de SBOM responde "¿dónde está?" en minutos.
- [ ] **Existe una vía de divulgación**
  Una política de divulgación de vulnerabilidades y un archivo security.txt; pruebas de penetración antes de los lanzamientos importantes y después de los cambios importantes.
- [ ] **El plan de respuesta a incidentes está ensayado**
  Roles, niveles de gravedad, un canal de comunicación fuera de banda y contactos de la asesoría jurídica, el DPD, las aseguradoras y los proveedores; ejercicios al menos una vez al año.
- [ ] **Las copias de seguridad sobreviven a un atacante con derechos de administrador**
  Copias inmutables o fuera de línea en una cuenta separada con credenciales separadas; restauraciones probadas frente al RTO y el RPO.
- [ ] **Las obligaciones de notificación están identificadas**
  A qué reguladores, clientes y socios hay que informar, quién lo hace y en qué plazos, con plantillas preparadas.

COMPRUEBA LO QUE HAS ENTENDIDO

▼ **¿Por qué un log debería registrar un hash o un ID de clave en lugar del propio token de acceso?**

Los logs se copian y se leen mucho, así que un token registrado puede robarse y reutilizarse. Un hash o un ID de clave sigue permitiendo a los investigadores correlacionar eventos sin exponer una credencial utilizable.

▼ **Un escáner informa de una vulnerabilidad con CVSS 9.8 en una biblioteca cuya función vulnerable nunca llamas, y de otra con CVSS 7.5 que figura en el catálogo KEV en tu gateway expuesto a internet. ¿Cuál va primero?**

La vulnerabilidad del gateway que figura en el catálogo KEV: se sabe que se explota y está expuesta. La de 9.8, inalcanzable, puede seguir el ciclo rutinario de parches, con el razonamiento anotado.

▼ **¿Por qué las copias de seguridad deben estar en una cuenta separada con credenciales separadas?**

Los atacantes que consiguen derechos de administrador en producción buscan las copias de seguridad y las destruyen; la separación hace que comprometer las identidades de producción no alcance la copia de recuperación.

▼ **¿Cuándo empieza a correr el plazo de 72 horas del RGPD?**

Cuando el responsable tiene constancia de la violación de datos personales, es decir, cuando tiene un grado razonable de certeza de que un incidente de seguridad ha comprometido datos personales, no cuando termina la investigación.

▼ **Inténtalo: La vulnerabilidad A tiene CVSS 9.8, está en una biblioteca de un servidor interno de procesos por lotes y su función vulnerable nunca se llama. La vulnerabilidad B tiene CVSS 7.5, está en tu servicio público de inicio de sesión y se sabe que se explota activamente. ¿Cuál va primero?**

B primero. La puntuación base de CVSS por sí sola es una mala guía. B es accesible desde internet y está en la lista de vulnerabilidades explotadas conocidas, así que un ataque es probable y podría afectar a un servicio crítico. Parchéalo o protégelo ahora, y revisa los logs en busca de señales de explotación pasada. La puntuación de A es mayor, pero es interno y su código no es alcanzable, así que prográmalo dentro del objetivo acordado. Anota el razonamiento y vuelve a comprobarlo si cambia la exposición.

FUENTES

- OWASP Logging Cheat Sheet: https://cheatsheetseries.owasp.org/cheatsheets/Logging_Cheat_Sheet.html
- NIST SP 800-61 Rev. 3: Incident Response Recommendations and Considerations (2025): https://csrc.nist.gov/pubs/sp/800/61/r3/final
- FIRST: Common Vulnerability Scoring System v4.0: https://www.first.org/cvss/v4-0/
- FIRST: Exploit Prediction Scoring System (EPSS): https://www.first.org/epss/
- RFC 9116: A File Format to Aid in Security Vulnerability Disclosure: https://www.rfc-editor.org/rfc/rfc9116

Lecturas recomendadas

- *Threat Modeling: Designing for Security*, Adam Shostack (2014), Wiley. El libro práctico de referencia sobre modelado de amenazas: las cuatro preguntas, los diagramas de flujo de datos, STRIDE, los árboles de ataque y cómo hacer que el modelado de amenazas forme parte de la ingeniería cotidiana.
- *Security Engineering: A Guide to Building Dependable Distributed Systems (3rd edition)*, Ross Anderson (2020), Wiley. Un relato amplio y muy documentado de cómo fallan los sistemas reales, desde los protocolos y el control de acceso hasta la banca, los pagos y la economía de la seguridad.
- *Designing Secure Software: A Guide for Developers*, Loren Kohnfelder (2021), No Starch Press. Principios de seguridad desde el diseño, modelado de amenazas y revisiones de diseño seguro de la mano de uno de los coinventores de STRIDE, escrito para desarrolladores y arquitectos en activo.
- *API Security in Action*, Neil Madden (2020), Manning. Autenticación, tokens, OAuth 2.0, seguridad basada en capacidades y protección de las llamadas entre servicios, con código funcional.
- *OAuth 2 in Action*, Justin Richer and Antonio Sanso (2017), Manning. Construye clientes, servidores de autorización y servidores de recursos paso a paso, lo que hace concretos los flujos y sus ataques.
- *The Protection of Information in Computer Systems*, Jerome H. Saltzer and Michael D. Schroeder (1975), Proceedings of the IEEE, vol. 63, no. 9. El origen del mínimo privilegio, los valores predeterminados a prueba de fallos, la mediación completa y los demás principios de diseño que todavía se citan hoy.
  https://doi.org/10.1109/PROC.1975.9939
- *SP 800-207: Zero Trust Architecture*, Scott Rose, Oliver Borchert, Stu Mitchell and Sean Connelly (NIST) (2020), National Institute of Standards and Technology. La definición de referencia de zero trust, sus principios y sus componentes lógicos (motor de políticas, administrador de políticas y punto de aplicación).
  https://csrc.nist.gov/pubs/sp/800/207/final
- *Regulation (EU) 2016/679 (General Data Protection Regulation)*, European Parliament and Council of the European Union (2016), Official Journal of the European Union (EUR-Lex). El texto principal en el que se basa la sección de privacidad; los artículos 5, 25 y 32 a 35 son los que más importan a los arquitectos.
  https://eur-lex.europa.eu/eli/reg/2016/679/oj

Glosario

Análisis de composición de software (SCA)
Herramientas y prácticas que identifican los componentes de código abierto de un software y los cotejan con bases de datos de vulnerabilidades y licencias conocidas.

Anonimización
Tratar los datos de modo que dejen de referirse a una persona identificable por cualquier medio que razonablemente pueda utilizarse; los datos verdaderamente anónimos quedan fuera del RGPD (GDPR).

Árbol de ataque
Un árbol cuya raíz es el objetivo de un atacante y cuyos hijos son las formas alternativas (OR) o combinadas (AND) de alcanzarlo, popularizado por Bruce Schneier en 1999.

Ataque a la cadena de suministro de software
Ataque que compromete algo que consume una organización, como una dependencia, una herramienta de compilación, una actualización o un registro, para llegar a esa organización y a otras situadas aguas abajo.

Autenticación
Verificar que un usuario o sistema es quien o lo que dice ser, por ejemplo, mediante una contraseña, una passkey o un certificado.

Autenticación multifactor (MFA)
Autenticación que exige dos o más tipos distintos de factor: algo que sabes, algo que tienes y algo que eres.

Autoridad de certificación (CA)
Una organización o un sistema que emite certificados digitales firmándolos, garantizando así que una clave pública pertenece a un servidor, una persona o una carga de trabajo determinados.

Autorización
Decidir si un usuario o sistema autenticado tiene permiso para realizar una acción concreta sobre un recurso concreto.

Base jurídica
Uno de los seis fundamentos del artículo 6 del RGPD (GDPR) (consentimiento, contrato, obligación legal, intereses vitales, misión de interés público, intereses legítimos) en los que debe basarse el tratamiento de datos personales.

Bug bounty (programa de recompensas)
Un programa que paga a investigadores externos por informar de vulnerabilidades de seguridad válidas mediante un proceso acordado.

Calendario de conservación
Lista documentada de cuánto tiempo se conserva cada categoría de datos, por qué y qué ocurre con ellos al final de ese periodo.

Cifrado de sobre (envelope encryption)
Cifrar los datos con una clave de datos y luego cifrar (envolver) esa clave de datos con una clave de cifrado de claves guardada en un KMS, de modo que los datos masivos nunca viajan al KMS y las claves pueden rotarse con poco costo.

Cifrado simétrico
Cifrado en el que la misma clave secreta cifra y descifra, como AES; es rápido, por lo que se usa para grandes volúmenes de datos.

Compilación reproducible (reproducible build)
Compilación que produce una salida idéntica bit a bit a partir del mismo código fuente y las mismas entradas, de modo que cualquiera pueda recompilar y comprobar que un artefacto publicado no ha sido manipulado.

Control de acceso basado en atributos (ABAC)
Decidir el acceso evaluando una política sobre atributos del usuario, el recurso, la acción y el contexto, como el país, la clasificación de los datos o la hora del día.

Control de acceso basado en relaciones (ReBAC)
Decidir el acceso a partir de las relaciones entre usuarios y objetos (propietario de, miembro de, trabaja en), almacenadas como un grafo; Zanzibar de Google (2019) es el ejemplo más conocido.

Control de acceso basado en roles (RBAC)
Conceder permisos a roles y roles a usuarios, de modo que el acceso se gestiona asignando a las personas roles como agente de soporte o gerente de restaurante.

Copia de seguridad inmutable
Una copia de seguridad que no puede modificarse ni eliminarse antes de que termine su periodo de retención, ni siquiera por un administrador, lo que la protege del ransomware y de personas internas malintencionadas.

Criptografía asimétrica (de clave pública)
Criptografía con un par de claves: una clave pública que cualquiera puede tener y una clave privada que se mantiene en secreto; se usa para el intercambio de claves, las firmas digitales y los certificados.

CVE
Common Vulnerabilities and Exposures: el programa que asigna a cada vulnerabilidad conocida públicamente un identificador único, como CVE-2021-44228.

CVSS
El Common Vulnerability Scoring System, mantenido por FIRST, que califica la gravedad de una vulnerabilidad conocida de 0 a 10; la versión 4.0 se publicó en 2023.

Datos personales
Según el RGPD (GDPR), toda información sobre una persona física (un individuo vivo) identificada o identificable, como un nombre, un número de identificación, datos de localización o un identificador en línea.

Defensa en profundidad
Superponer varios controles independientes para que un atacante que supere uno siga enfrentándose a otros, por ejemplo, una comprobación en el gateway, una comprobación en el servicio y el cifrado de los propios datos.

Dependencia transitiva
Paquete que tu software usa indirectamente porque una de tus dependencias, o una de las suyas, depende de él.

Diagrama de flujo de datos (DFD)
Un diagrama de entidades externas, procesos, almacenes de datos y los flujos de datos entre ellos, con límites de confianza trazados a través de los flujos; es el modelo habitual para el modelado de amenazas.

Encargado del tratamiento
Organización que trata datos personales por cuenta de un responsable del tratamiento, siguiendo sus instrucciones y en virtud de un contrato, como un proveedor en la nube de SMS o de correo electrónico.

EPSS
El Exploit Prediction Scoring System de FIRST, que estima la probabilidad de que una vulnerabilidad se explote en los próximos 30 días.

Evaluación de impacto relativa a la protección de datos (EIPD)
Una evaluación exigida por el artículo 35 del RGPD (GDPR) antes de un tratamiento que probablemente entrañe un alto riesgo para las personas, que describe el tratamiento, su necesidad, los riesgos y las medidas que los abordan.

Gestión de secretos
Almacenar, distribuir, rotar y auditar credenciales como contraseñas, claves de API y claves privadas en una bóveda dedicada en lugar de en el código o en archivos de configuración.

Hash de contraseñas
Guardar una contraseña como el resultado de una función deliberadamente lenta y con sal, como Argon2id, scrypt o bcrypt, para que una base de datos robada no pueda volver a convertirse en contraseñas de forma barata.

Identidad de carga de trabajo
Identidad verificable para un servicio en ejecución en lugar de para una persona, emitida por la plataforma (por ejemplo, un SPIFFE ID en un certificado de corta duración), que se usa en lugar de secretos compartidos.

Infraestructura de clave pública (PKI)
Las autoridades de certificación, los certificados, los almacenes de confianza, las políticas y los mecanismos de revocación que, en conjunto, permiten a las partes verificar las claves públicas de las demás.

JSON Web Token (JWT)
Token compacto, firmado (y opcionalmente cifrado) que transporta afirmaciones (claims) como emisor, sujeto, audiencia y caducidad, definido en el RFC 7519; muy usado para tokens de acceso y tokens de ID.

Límite de confianza
Línea de un sistema en la que los datos o el control pasan entre partes con distintos niveles de confianza o privilegio, como de internet a una API, por lo que la entrada debe comprobarse allí.

Lista de materiales de software (SBOM)
Inventario legible por máquina de los componentes y dependencias de una pieza de software, en un formato estándar como SPDX o CycloneDX.

Lockfile (archivo de bloqueo)
Archivo generado por un gestor de paquetes que registra las versiones exactas, y normalmente los hashes, de cada dependencia resuelta, de modo que cada instalación use los mismos paquetes.

Microsegmentación
Dividir una red en zonas muy pequeñas, hasta cargas de trabajo individuales, de modo que cada una acepte tráfico solo de los llamantes que necesita.

Mínimo privilegio
El principio de que cada usuario, programa y servicio recibe solo los permisos que necesita para su tarea y solo durante el tiempo que los necesita, de modo que un error o un compromiso cause daños limitados.

Modelado de amenazas
Forma estructurada de preguntarse, sobre un diseño, qué estamos construyendo, qué puede salir mal, qué vamos a hacer al respecto y si lo hemos hecho lo bastante bien.

Módulo de seguridad de hardware (HSM)
Hardware resistente a manipulaciones que genera y usa claves en su interior y nunca las libera en texto plano; los servicios KMS suelen construirse sobre HSM.

Movimiento lateral
El avance de un atacante desde la primera máquina o cuenta que compromete hacia otros sistemas dentro de la misma red u organización.

OAuth 2.0
El marco de autorización delegada del RFC 6749, en el que un servidor de autorización emite tokens de acceso que permiten a un cliente llamar a un servidor de recursos en nombre del propietario del recurso, sin ver su contraseña.

OpenID Connect (OIDC)
Capa de identidad sobre OAuth 2.0 que añade un token de ID, un JWT firmado que indica al cliente quién es el usuario y cómo y cuándo se autenticó.

Passkey (llave de acceso)
Credencial FIDO2/WebAuthn: un par de claves creado para un sitio web, cuya clave privada permanece en el dispositivo o el gestor de contraseñas del usuario, lo que hace el inicio de sesión resistente al phishing y no deja ningún secreto compartido en el servidor.

Pista de auditoría
Un registro completo, cronológico y a prueba de manipulaciones de acciones definidas, que muestra quién hizo qué y cuándo, conservado para establecer la rendición de cuentas.

PKCE
Proof Key for Code Exchange (RFC 7636): el cliente envía un hash de un secreto de un solo uso cuando inicia el flujo de código de autorización y el propio secreto cuando canjea el código, de modo que un código robado no sirve de nada.

Procedencia (provenance)
Información verificable sobre el origen de un artefacto: qué plataforma de compilación lo construyó, a partir de qué revisión del código fuente, con qué entradas y qué parámetros.

Protección de datos desde el diseño y por defecto
La obligación del artículo 25 del RGPD (GDPR) de incorporar los principios de protección de datos a los sistemas mediante medidas adecuadas y de tratar por defecto solo los datos necesarios para cada finalidad.

Punto de aplicación de políticas (PEP)
El componente situado en la ruta del tráfico, como un proxy, una pasarela o un sidecar, que permite o bloquea cada solicitud según una decisión de política.

Punto de decisión de políticas (PDP)
El componente que evalúa la política de acceso y responde permitir o denegar; un punto de aplicación de políticas (PEP) le consulta y luego aplica la respuesta.

Reglamento de Ciberresiliencia (Cyber Resilience Act, CRA)
El Reglamento (UE) 2024/2847, que establece requisitos de ciberseguridad, gestión de vulnerabilidades y obligaciones de notificación para los productos de hardware y software que se venden en la UE.

Responsable del tratamiento (controller)
La organización que decide por qué y cómo se tratan los datos personales y que tiene la responsabilidad principal de cumplir la legislación de protección de datos.

Respuesta a incidentes
La gestión organizada de un incidente de seguridad: detectarlo y analizarlo, contenerlo, eliminar la causa, recuperarse y aprender de él.

Runbook (manual de operaciones)
Procedimiento escrito y probado para gestionar paso a paso una situación operativa o de seguridad concreta, que incluye cómo verificar y revertir cada paso.

Secure Software Development Framework (SSDF)
NIST SP 800-218: conjunto de prácticas de desarrollo seguro agrupadas en preparar la organización, proteger el software, producir software bien protegido y responder a las vulnerabilidades.

Seguridad perimetral
El modelo tradicional que protege una red interna de confianza con cortafuegos y VPN y trata como amigo todo lo que ya está dentro; a veces se le llama seguridad de castillo y foso.

Seguro por defecto
Entregar un producto cuya configuración inicial es la segura, de modo que los usuarios tengan que elegir deliberadamente debilitar la protección en lugar de acordarse de activarla.

Servicio de gestión de claves (KMS)
Servicio gestionado que crea, almacena, rota y controla el uso de claves criptográficas, y realiza operaciones con ellas para que las aplicaciones nunca vean las claves maestras.

Seudonimización
Tratar datos personales de forma que ya no puedan vincularse a una persona sin información adicional guardada por separado; según el RGPD (GDPR), los datos seudonimizados siguen siendo datos personales.

Shift left
Adelantar las actividades de seguridad (o de pruebas) en el ciclo de vida del desarrollo, hacia los requisitos, el diseño y la codificación, donde corregir los problemas es más barato que en producción.

SIEM
Security information and event management (gestión de información y eventos de seguridad): sistema que recopila logs de seguridad de muchas fuentes, los correlaciona y genera alertas cuando coinciden con reglas de detección.

Sigstore
Proyecto de código abierto para firmar y verificar artefactos de software con certificados de corta duración vinculados a una identidad y un registro público de transparencia, lo que evita las claves de firma de larga duración.

SLSA
Supply-chain Levels for Software Artifacts: marco de niveles crecientes de garantía de que un artefacto se construyó a partir del código fuente esperado mediante una compilación confiable y resistente a la manipulación.

STRIDE
Acrónimo mnemotécnico de seis tipos de amenaza (suplantación de identidad o spoofing, manipulación o tampering, repudio, divulgación de información, denegación de servicio y elevación de privilegios), creado en Microsoft por Loren Kohnfelder y Praerit Garg en 1999.

Superficie de ataque
La suma de los puntos por los que un atacante puede intentar introducir datos en un sistema, extraerlos o interactuar con él de otro modo: endpoints, puertos, archivos, colas, herramientas de administración y las personas y el código en los que confía.

TLS mutuo (mTLS)
TLS en el que ambas partes presentan y verifican certificados, de modo que un servicio sabe qué servicio lo está llamando, además de que quien llama conoce al servidor.

Tokenización
Sustituir un valor sensible, como un número de tarjeta, por un token sin significado, mientras el valor real se guarda en una bóveda separada y muy protegida, a menudo gestionada por un proveedor de pagos.

Violación de datos personales
Según el artículo 4(12) del RGPD (GDPR), una violación de la seguridad que ocasiona la destrucción, pérdida o alteración accidental o ilícita de datos personales, o la comunicación o el acceso no autorizados a ellos.

Zero trust (confianza cero)
Modelo de seguridad que no concede ninguna confianza implícita basada en la ubicación en la red: cada solicitud se autentica, se autoriza y se cifra, por recurso y por sesión (NIST SP 800-207).