

DEL BIT AL PÍXEL

# Del bit al píxel

*Qué poseer y qué alquilar*

Cada pantalla que ven sus clientes la construyen muchos dueños, del silicio al píxel. Este libro recorre la pila de Windows, Linux y macOS capa por capa, cuenta cómo llegó a ser así y ofrece a un CTO una forma sencilla de decidir qué capas poseer y cuáles alquilar, según el resultado de negocio que importa.

**Juan Cardena**

Arquitecto de IA empresarial y sistemas agénticos · blog.jcardena.com

## CONTENIDO


ANTES DE EMPEZAR

Resumen para el lector: seguridad, costo, valor

PRÓLOGO

Una pulsación de tecla y una pregunta

PARTE I · LA PILA, CAPA POR CAPA

01 Silicio y arranque

02 El kernel

03 Controladores, almacenamiento y red

04 Seguridad, identidad y el píxel de confianza

05 Entornos de ejecución y lenguajes

06 Gráficos, texto y el píxel

07 El navegador, el segundo sistema operativo

08 Cadena de herramientas, distribución y los bordes de un sistema operativo

PARTE II · CÓMO LLEGAMOS AQUÍ

09 Setenta años de sistemas operativos

10 Por qué ganaron los ganadores

PARTE III · SER DUEÑO DEL BIT AL PÍXEL

11 Las costuras, en números

12 El precio de poseer

13 La IA mueve las costuras

14 Controlar el bit y la cadena de herramientas

15 Qué poseer y qué alquilar

CIERRE

Epílogo: la última capa es el ser humano

APÉNDICES

A Glosario

B Línea de tiempo

C Tabla comparativa completa

D Lecturas y fuentes

ANTES DE EMPEZAR

# Resumen para el lector: seguridad, costo, valor

Este resumen explica qué hará el libro por usted y dónde encontrarlo. Indica a qué lector se dirige y presenta las tres empresas que vuelven en cada capítulo, las preguntas que trae cada una y la única herramienta que el libro construye capa por capa. Se lee en diez minutos; después, basta seguir la ruta que corresponde a cada rol.

## Para quién es este libro

Está escrito para el **CTO, vicepresidente de ingeniería o arquitecto principal de una empresa mediana**, con unos 50 a 1000 ingenieros. Las empresas de ese tamaño alquilan casi todas las capas de la pila sobre la que entregan su producto. Son lo bastante grandes para poseer o controlar algunas capas, como la ruta de publicación y la interfaz propia del producto, y lo bastante pequeñas para que una mala apuesta sobre una capa duela.

No hace falta ser ingeniero especializado en el núcleo (kernel) del sistema operativo. Cada término técnico se explica la primera vez que aparece. Lo que sí hace falta es responder por tres cosas: la **seguridad** de lo que se ejecuta en las máquinas de sus clientes y de su personal, el **costo** de mantenerlo en marcha y el **valor** que crea para el negocio. El libro se organiza en torno a esas tres palabras.

Dos tipos de lectores sacarán menos provecho. Los grandes proveedores de nube a hiperescala ya poseen la mayor parte de su pila. Las startups en etapa temprana deberían alquilar casi todas las capas por ahora, con una excepción: la ruta que construye, firma y actualiza su propio producto, que el capítulo 14 sostiene que deben controlar desde el principio. Su ruta es la Parte I y luego el capítulo 14; el resto de la Parte III puede esperar.

## Tres empresas, lo que cada una se juega

Los consejos abstractos sobre "la pila" son fáciles de aceptar y difíciles de aplicar. Por eso cada capítulo pone a prueba sus ideas con las mismas tres empresas. Ninguna es un cliente real; cada una reúne rasgos comunes de empresas de su tipo.

> **EL FABRICANTE DE SOFTWARE**
>
> Entrega una aplicación de escritorio y una aplicación web a miles de clientes empresariales, en Windows, macOS y a veces Linux. Lo que se juega está en las capas que decide incluir: un motor de navegador dentro de su aplicación, un entorno de ejecución (runtime) de lenguaje, un instalador, un canal de actualización. Cada capa que entrega se vuelve su problema en la máquina de cada cliente, y cada capa que deja al proveedor del sistema operativo se vuelve una hoja de ruta que no controla.

> **LA EMPRESA REGULADA**
>
> Opera en finanzas, salud o gobierno, y debe demostrar lo que se ejecuta en sus máquinas, no solo creerlo. Lo que se juega es la evidencia y la respuesta: qué código puede llegar al kernel, en qué claves de arranque confían sus computadoras portátiles, qué tan rápido pasa un parche del proveedor a la flota y si puede mostrar la respuesta a un auditor en una sola página.

> **EL OPERADOR DE UNA FLOTA DE DISPOSITIVOS**
>
> Es un grupo hotelero o un minorista que opera miles de máquinas idénticas en campo: terminales de punto de venta, quioscos de registro, pantallas de autoservicio. Las máquinas viven años y están en lugares sin un ingeniero cerca. Lo que se juega es la disponibilidad y el costo de una visita a las instalaciones. Una mala actualización o una fecha de certificado olvidada afecta a todas las ubicaciones a la vez.

## El mapa de valor

Estas son las nueve preguntas que responde el libro, una por empresa y por tema, con el capítulo que responde cada una. Basta buscar la fila de la empresa correspondiente y empezar ahí.

| Empresa | Seguridad | Costo | Valor |
|---|---|---|---|
| **Fabricante de software** | ¿Cuántas copias de un motor de navegador entregamos y qué tan rápido podemos parchear cada una? **Cap. 7** | ¿Cuánto les cuesta a nuestros clientes, en memoria, disco y trabajo de actualización, incluir un runtime por aplicación? **Cap. 11** | ¿Cuáles son las pocas capas que vale la pena poseer porque diferencian el producto? **Cap. 15** |
| **Empresa regulada** | ¿Quién puede cambiar código dentro de nuestro kernel y cómo se lo demostramos a un auditor? **Cap. 2** | ¿Cuánto nos cuesta el retraso de parches de todos nuestros proveedores en exposición y en trabajo por incidentes? **Cap. 11** | ¿Hasta dónde pueden la identidad, la firma y la atestación demostrar lo que una persona realmente vio y aprobó? **Cap. 4** |
| **Operador de flota de dispositivos** | ¿Qué certificados de arranque y fechas de soporte de firmware vencerán en nuestra flota, y cuándo? **Cap. 1** | ¿Qué controladores y dispositivos podemos estandarizar para reducir la lista de actualizaciones y las visitas a las instalaciones? **Cap. 3** | ¿Debemos controlar la imagen del sistema operativo y la ruta de actualización de nuestros quioscos, y qué haría falta? **Cap. 14** |

## Lo que podrá hacer

- **Nombrar a cada dueño** entre su código y la pantalla de su cliente, capa por capa, en Windows, Linux y macOS.
- **Llenar una Lista de materiales del píxel (Pixel Bill of Materials)** para su propio producto: una fila por componente, con quién lo escribe, quién paga, quién puede enviar una actualización a sus máquinas y cuándo termina su soporte.

- **Ponerles cifra a las costuras**: la memoria, los canales de actualización y el retraso de parches que resultan de alquilar muchas capas a muchos dueños.
- **Decidir entre alquilar, financiar, bifurcar o construir** cada capa, ligando cada decisión a una métrica de negocio, y defender la decisión ante su consejo de administración.
- **Planear la respuesta a incidentes por dueño**, para que cuando una falla cruce capas ya se sepa a quién llamar y qué puede corregir cada uno.

### Rutas de lectura por rol

| Rol | Empezar con | Después | Tener a mano |
|---|---|---|---|
| **CTO o vicepresidente de ingeniería** | Prólogo, capítulo 10 (Por qué ganaron los ganadores) | Parte III: capítulos 11 a 15 | Parte I, un capítulo por capa, según surjan las preguntas |
| **Líder de plataforma o arquitecto** | Parte I en orden: capítulos 1 a 8 | Capítulos 11, 12 y 14 | Apéndice C, la tabla comparativa completa |
| **Líder de seguridad** | Capítulos 1, 2 y 4 | Capítulos 7 y 13 (La IA mueve las costuras) | El recuadro Lente de seguridad al final de cada capítulo |

La Parte II (capítulos 9 y 10) cuenta cómo nacieron los tres sistemas. Es el mejor lugar para entender por qué cada proveedor se comporta como lo hace, y se puede leer por separado.

### La lista de materiales del píxel

Una fábrica conoce cada pieza de su producto, quién la fabrica y cuánto cuesta. Los equipos de software rara vez saben lo mismo de la pantalla que miran sus clientes. La **lista de materiales del píxel** es esa lista para toda la pila, del procesador al píxel encendido. Cada capítulo de la Parte I agrega sus filas, y el capítulo 15 le entrega una en blanco. Las tablas de los capítulos tienen siete columnas.

| Columna | Qué pregunta |
|---|---|
| **Componente** | La pieza real, por su nombre: un firmware, un kernel, un controlador, una biblioteca, un motor. |
| **Quién lo escribe** | La empresa o el proyecto cuyos ingenieros cambian el código. |
| **Quién paga** | Quién financia ese trabajo: un proveedor a través de sus precios, patrocinadores o usted. |
| **Quién puede actualizarlo** | Quién puede enviar una versión nueva a sus máquinas. A menudo no es quien lo escribe, y ninguno de los dos es siempre quien lo firma; los capítulos lo señalan donde la diferencia importa. |
| **Plazo de soporte** | Cuándo deja de publicar actualizaciones el proveedor, o cuándo vence un certificado o una clave. |
| **¿Expuesto a entradas no confiables?** | ¿Puede un archivo, una página web, un paquete de red o una imagen de afuera llegar a este código? Si la respuesta es sí, sus fallas son vías de ataque. |
| **Su decisión** | **Alquilar:** usar la capa del proveedor tal como se entrega. **Financiar:** pagar o aportar personal a una capa abierta que no controla; compra influencia, no control. **Bifurcar:** mantener su propia copia modificada. **Construir:** escribir la suya. Se permiten combinaciones. |

En este libro, **poseer** una capa significa bifurcarla o construirla: el código es suyo. **Controlar** una capa significa decidir su composición, su firma y su despliegue aunque no la haya escrito, por ejemplo una imagen de sistema operativo que se compone con piezas del proveedor y se fija en una versión, o su cadena de publicación (pipeline) que se ejecuta en servidores de compilación alquilados. Configurar o fijar versiones es control, no una bifurcación. La ruta de publicación es el ejemplo principal: en cada lista su decisión dice **Controlar (construir solo las piezas críticas para la confianza)**, es decir, servidores de compilación alquilados con su propio pipeline y sus propias claves de firma. Cuando el libro habla de un proveedor, dice que el proveedor **suministra** o **controla** una capa, como en "el proveedor del sistema operativo controla el aviso de consentimiento". La lista completa del capítulo 15 agrega tres columnas a las siete: **plazo de parche si la IA lo encuentra primero**, **tiempo de recuperación probado** y **responsable**, la persona que responde por la fila.

Este es un ejemplo de fila resuelta, del capítulo 2, para el agente de seguridad de los equipos (endpoints) que una empresa regulada ejecuta en sus portátiles.

| Componente | Quién lo escribe | Quién paga | Quién puede actualizarlo | Plazo de soporte | ¿Expuesto a entradas no confiables? | Su decisión |
|---|---|---|---|---|---|---|
| Agente de seguridad de los equipos (endpoints) con un componente en el kernel | El proveedor de seguridad | Usted, por suscripción | El proveedor, a menudo muchas veces al día, sin ventana de cambios | El plazo de su contrato | Sí: por diseño inspecciona archivos y tráfico no confiables | Alquilar; exigir despliegue escalonado y un plan para salir del kernel |

Al leer la fila de izquierda a derecha aparece un riesgo que ninguna columna muestra por sí sola: una parte a la que usted paga, pero que no controla, puede cambiar código en el nivel de privilegio más alto de cada máquina que usted posee, sin esperar su autorización. Eso no es razón para abandonar el producto. Es razón para negociar cómo se actualiza. En julio de 2024, una actualización defectuosa de uno de esos agentes dejó fuera de servicio unos 8,5 millones de dispositivos Windows, según el conteo de Microsoft. El capítulo 2 cuenta esa historia.

### La idea que hay que llevarse

Cada capa tiene un dueño. Poseer más capas no hace por sí solo que un producto sea mejor o más seguro, y alquilar una capa no es un fracaso. Lo que la propiedad cambia es algo más acotado y más útil de saber.

> *Poseer una capa cambia quién puede corregirla, y qué tan rápido.*

Conviene tener esa frase junto a cada decisión de este libro. En lugar de preguntar "¿deberíamos poseer esto?", conviene preguntar: cuando esta capa falle, ¿quién la va a corregir, cuánto tardará y podemos vivir con esa respuesta?

PRÓLOGO

# Una pulsación de tecla y una pregunta




Se pulsa la letra a. Un instante después aparece en la pantalla una pequeña forma de luz. Parece un solo evento. Es una carrera de relevos a través de cada capa de la máquina, y casi todos los corredores de esa carrera trabajan para alguien más.

Este libro sigue esa carrera. Empieza en un interruptor de tecla que el controlador del teclado codifica como bits, y termina en el píxel, un punto encendido en una pantalla. Entre ambos están el firmware, un núcleo (kernel), los controladores (drivers), un sistema de entrada, un entorno de ejecución (runtime) de la aplicación, un motor de texto, un compositor y un procesador gráfico. Windows, Linux y macOS corren la carrera cada uno a su manera, y cada respuesta fue moldeada por décadas de historia y de negocio.

La razón para seguirla no es la curiosidad. Cada capa de esa carrera tiene un dueño, una hoja de ruta y un precio. Cuando una capa cambia, su producto cambia con ella, esté o no preparado. Un CTO que puede ver toda la carrera puede decidir dónde ubicarse en ella. El Resumen para el lector presenta las tres empresas que ponen a prueba esa decisión en cada capítulo.

## La misma tecla, tres caminos

Esta es la letra a en cada sistema, del teclado a la luz. Los nombres son componentes reales que se pueden consultar. La forma es la misma en los tres; los dueños no.

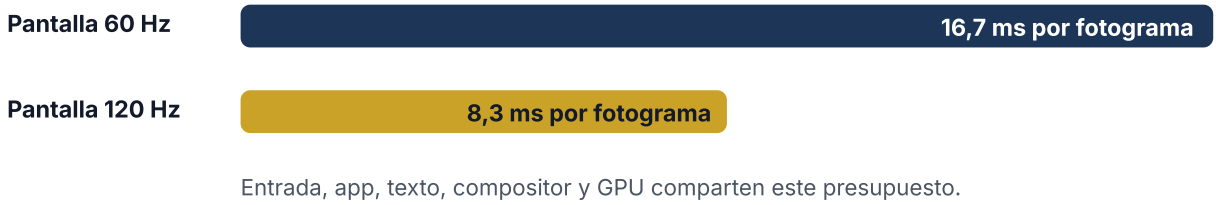
| Paso | Windows | Linux (escritorio Wayland) | macOS |
|---|---|---|---|
| **Reporte de tecla** | El teclado envía un reporte USB HID que lleva el estado de los modificadores (Shift, Ctrl, Alt) y el código de uso 0x04, la posición de tecla llamada "a and A"; luego el sistema operativo aplica la distribución del teclado para decidir en qué carácter se convierte | | |
| **Entrada en el kernel** | kbdhid.sys, kbdclass.sys | usbhid, evdev | IOHIDFamily |
| **Evento a la aplicación** | win32k, WM_KEYDOWN y WM_CHAR | libinput, xkbcommon, evento de tecla de Wayland | WindowServer, NSEvent |
| **Texto** | DirectWrite | HarfBuzz da forma al texto, FreeType dibuja | Core Text |
| **Componer la pantalla** | Desktop Window Manager | Mutter o KWin | WindowServer, Core Animation |
| **GPU y pantalla** | Direct3D, controlador WDDM | Mesa, DRM y KMS | Metal |
| **Luz** | El nuevo fotograma se muestra en el siguiente refresco de la pantalla | | |

La tabla muestra el recorrido habitual a través del compositor. No es el único: una aplicación a pantalla completa a veces puede saltarse el compositor y entregar sus fotogramas directamente al hardware de la pantalla, mediante salida directa a pantalla (direct scanout), superposiciones de hardware (overlays) o, en Windows, independent flip.

> **CÓMO LLEGÓ A SER ASÍ**
>
> Un teclado PS/2 generaba una interrupción de hardware, IRQ 1, cada vez que cambiaba una tecla. Un teclado USB funciona al revés. USB define un tipo de transferencia de "interrupción", pero es el host el que sondea al teclado (polling), con un intervalo que el dispositivo declara en su descriptor de endpoint, a menudo de 1 a 10 milisegundos. Después, el controlador USB del host interrumpe al procesador cuando llegan datos. Así, una pulsación de tecla moderna mezcla sondeo e interrupciones, y la pila conserva rastros de ambos diseños. Esa es la primera lección de este libro: la pila es historia que todavía se puede ver.

FIGURA P.1

### El presupuesto de tiempo de un fotograma




Entrada, app, texto, compositor y GPU comparten este presupuesto.

Un segundo dividido entre la frecuencia de refresco. Cada capa de este libro gasta una parte, y pasarse de ese tiempo obliga al usuario a esperar un fotograma completo más.

> *Cada capa entre el bit y el píxel tiene un dueño. Poseer una capa cambia quién puede corregirla, y qué tan rápido.*

## La lista de materiales del píxel

El libro construye una lista de cada capa de ese camino, un capítulo a la vez, y la llama **Lista de materiales del píxel (Pixel Bill of Materials)**. Esta es una vista previa simplificada; las columnas completas están en el Resumen para el lector.

| Capa | Quién la escribe | Quién la paga | Quién puede actualizarla | Su decisión |
|---|---|---|---|---|
| Modelado de texto (shaping) | Un proyecto abierto con un equipo central pequeño | Sobre todo patrocinadores | Cada aplicación o sistema operativo que incluye una copia | Alquilar o financiar |
| Motor de navegador | Proyecto original (upstream), dirigido en gran parte por un proveedor | Ese proveedor | El proveedor del runtime, o cada aplicación que lo incluye | Alquilar y planear para la costura |
| Su ruta de publicación | Su equipo, en servidores de compilación alquilados | Usted | Usted | Controlar (construir solo las piezas críticas para la confianza) |

Para cada fila, una empresa tiene cuatro opciones reales. Puede **alquilar** la capa: usar la capa del proveedor tal como se entrega. Puede **financiarla**: pagar o aportar personal a una capa abierta que no controla, lo que compra influencia, no control. Puede **bifurcarla**: mantener su propia copia modificada. O puede **construirla**: escribir la suya. Ninguna gana de antemano. Poseer una capa no hace mejor a un producto; lo que cambia es quién corrige los defectos y en qué plazos: de qué equipo son los ingenieros que corrigen un defecto, y según qué calendario. Cada capítulo de la Parte I agrega sus filas a la lista, y el capítulo 15 le entrega una en blanco para su propia empresa.

**IMPLICACIONES PARA EL NEGOCIO**

**Seguridad.** Los dueños de este camino cumplen papeles distintos: algunos escriben el código, otros lo firman y otros pueden enviar una actualización a las máquinas de sus clientes. Hay que saber quién es quién, qué tan rápido parchea cada uno y cuáles se ejecutan dentro del kernel.

**Costo.** Cada capa trae su propio canal de actualización, su ciclo de pruebas y sus copias en memoria. Ese sobrecosto de las costuras es dinero real, y pocas empresas lo suman.

**Valor.** Poseer una capa es una decisión de negocio, no una elección de lenguaje. El capítulo 15 da los criterios para elegir las pocas capas que vale la pena poseer o controlar, y para alquilar el resto en buenas condiciones.

CAPÍTULO UNO

# Silicio y arranque




Antes de que su sistema operativo ejecute una sola línea, otros tres dueños ya ejecutaron la suya: el fabricante del chip, el proveedor del firmware y la empresa que tiene las claves de firma del arranque. Usted no eligió a la mayoría, y en la práctica una flota mediana no puede parchear ese firmware ni volver a firmar esas claves de arranque por su cuenta. Sin embargo, cuando uno de ellos falla, es su flota la que deja de arrancar. Este capítulo trata del código que se ejecuta primero y del calendario que lleva en silencio.

## Anillos, modos y el mapa de memoria

Un procesador no confía por igual en todas las instrucciones. Funciona con **niveles de privilegio**, y el hardware decide qué nivel puede tocar qué parte de la máquina. En los procesadores Intel y AMD (la familia x86) estos niveles se llaman **anillos**, numerados del 0 al 3, y un número menor significa más privilegio. El anillo 0 tiene el mayor privilegio que el procesador le da a un sistema operativo, aunque un hipervisor, el System Management Mode del firmware y las protecciones de la plataforma todavía pueden limitarlo. El anillo 3 tiene el menor, y el código que se ejecuta ahí solo puede hacer lo que el código más privilegiado le permite. En la práctica, Windows, Linux y macOS en x86 usan solo dos: el anillo 0 para el kernel y el anillo 3 para las aplicaciones.

Los procesadores Arm, que equipan todas las Mac que Apple vende hoy y la mayoría de los teléfonos, usan en cambio **niveles de excepción**. EL0 ejecuta aplicaciones, EL1 ejecuta el kernel del sistema operativo, EL2 ejecuta un hipervisor (la capa que aloja máquinas virtuales) y EL3 ejecuta el firmware de mayor confianza. Aquí la numeración va al revés: un nivel más alto significa más privilegio. La idea es la misma. El código en un nivel más privilegiado en general puede alcanzar el estado de los niveles inferiores, y un hipervisor y el firmware agregan dominios de privilegio por encima del kernel.

La segunda protección es la **unidad de gestión de memoria**, o MMU. Las aplicaciones comunes trabajan con direcciones virtuales, no físicas. Cada proceso ve su propio mapa privado, llamado **memoria virtual**, y la MMU traduce cada dirección al vuelo con tablas que escribe el kernel. La documentación de controladores de Microsoft lo dice sin rodeos: como el espacio de direcciones de cada aplicación es privado, una aplicación no puede modificar los datos de otra. El código del kernel no recibe un mapa privado así, y el capítulo 2 muestra lo que eso cuesta.

## La GPU, una segunda computadora

El procesador gráfico, o GPU, no es un chip auxiliar. Es una segunda computadora con su propio conjunto de instrucciones, su propio planificador y su propio firmware. Una tarjeta gráfica dedicada también tiene su propia memoria, llamada memoria de video; una GPU integrada, dentro del encapsulado del procesador como en muchas computadoras portátiles, comparte la memoria principal con la CPU. En el camino acelerado por hardware, la CPU no dibuja sus píxeles. Escribe comandos en un búfer, y la GPU los ejecuta con su propio reloj y rasteriza el resultado. Una CPU todavía puede dibujar mediante un rasterizador por software, la alternativa que describe el capítulo 3. Su controlador, escrito por el fabricante de la GPU, es una de las piezas de código más grandes de cualquier escritorio, y los capítulos 3 y 6 vuelven a él.

En una computadora con tarjeta gráfica dedicada, la GPU trabaja más rápido con datos en su propia memoria de video, así que los programas suelen copiar los datos ahí, aunque la GPU también puede leer memoria del sistema asignada a su espacio de direcciones a través del bus, más despacio. La documentación de Direct3D 12 de Microsoft nombra los dos casos: un adaptador dedicado tiene memoria de video local y memoria del sistema no local, mientras que un adaptador integrado, o UMA, tiene un solo segmento, la memoria del sistema. Apple construyó sus propias máquinas en torno al modelo compartido. Cuando presentó el chip M1 el 10 de noviembre de 2020, Apple describió una **arquitectura de memoria unificada**: un solo conjunto de memoria compartido por la CPU, la GPU y otros motores, para que puedan usar los mismos datos sin copiarlos entre conjuntos. Es una decisión de hardware con efectos en el software. Las GPU integradas en computadoras con Windows y Linux también comparten memoria. Lo que agrega el control de Apple sobre el chip y el sistema operativo es que sus frameworks de gráficos, aprendizaje automático y video pueden dar por hecho ese único diseño en cada Mac que vende.

## Firmware: el código antes del código

El **firmware** es el programa guardado en un chip de la tarjeta madre. Despierta el hardware, revisa la memoria, encuentra un disco y entrega el control a un cargador del sistema operativo. En las computadoras y los servidores de hoy, ese firmware sigue el estándar **UEFI** (Unified Extensible Firmware Interface), mantenido por el UEFI Forum, que cuenta con más de 250 empresas miembro.

Pocos fabricantes de computadoras escriben su propio firmware UEFI. La mayoría licencia una base de código de un **proveedor independiente de BIOS**: principalmente AMI, Insyde y Phoenix. El fabricante de la computadora agrega su propio código encima y entrega el resultado. Así, una sola falla de firmware puede tener tres dueños: el código de referencia del fabricante del chip, la base del proveedor de BIOS y la compilación del fabricante de la computadora. Cada uno puede tener que publicar una corrección antes de que su máquina esté segura.

**Secure Boot** es la función de UEFI que decide qué puede ejecutarse después. El firmware guarda un pequeño conjunto de bases de datos de claves, definidas en fábrica. La guía de hardware de Microsoft las nombra: la base de datos de firmas ( `db` ) lista los firmantes de confianza; la base de datos de revocaciones ( `dbx` ) lista los firmantes e imágenes que no lo son; la base de datos Key Exchange Key ( `KEK` ) guarda las claves autorizadas para actualizar esas dos listas; y la Platform Key ( `PK` ), definida por

el fabricante de la computadora, controla la KEK. Antes de ejecutar un cargador de arranque, el firmware lo verifica contra `db` y `dbx`. Cuando Secure Boot está activado y aplicándose, un cargador solo se ejecuta si lo firmó una clave de `db` o si su hash aparece en `db`, y solo si `dbx` no lo ha revocado.

Un **TPM** (Trusted Platform Module) es un pequeño chip de seguridad, o una función protegida dentro del procesador, que guarda claves y registra mediciones de lo que arrancó. Windows 11 exige TPM 2.0 y un firmware UEFI compatible con Secure Boot. En noviembre de 2020 Microsoft anunció **Pluton**, un procesador de seguridad integrado en la CPU junto con AMD, Intel y Qualcomm. Su diseño se entregó por primera vez en la Xbox One en 2013. En las computadoras con Windows, Pluton actúa primero como TPM, y su firmware se actualiza mediante Windows Update, lo que traslada parte de la raíz de confianza del hardware al calendario de parches de Microsoft.

| Paso de arranque | Computadora con Windows | Computadora con Linux | Mac con Apple silicon |
|---|---|---|---|
| **Primera etapa del firmware del host relevante para el arranque del sistema** | Firmware UEFI del fabricante de la computadora, por lo general sobre una base de AMI, Insyde o Phoenix | El mismo firmware UEFI | Boot ROM, fijada en el chip al fabricarlo |
| **Quién firma la siguiente etapa** | Microsoft (gestor de arranque de Windows) | Microsoft firma `shim`; la distribución firma GRUB y el kernel | Apple (LLB, luego iBoot) |
| **Almacén de claves y políticas** | UEFI `db`, `dbx`, `KEK`, `PK` | Claves UEFI, más la lista de Machine Owner Keys en `shim` | LocalPolicy, firmada por el Secure Enclave |
| **Custodio de claves en hardware** | TPM 2.0 o Pluton | TPM 2.0 | Secure Enclave, con su propia boot ROM y sepOS |
| **Quién entrega las correcciones de firmware** | El fabricante de la computadora; algunas mediante Windows Update | El fabricante de la computadora; algunas mediante el Linux Vendor Firmware Service | Apple, dentro de las actualizaciones de macOS |

El firmware UEFI no es el primerísimo código de una computadora. Los procesadores de seguridad dentro del chip, como el Converged Security and Management Engine de Intel y el Platform Security Processor de AMD, arrancan antes desde su propia ROM y pueden verificar el firmware antes de que lo ejecuten los núcleos principales. Pertenecen al fabricante del chip y forman parte de su fila en la lista.

En Linux, la cadena tiene un eslabón más. Como la mayoría de las computadoras salen de fábrica confiando en las claves de Microsoft, muchas distribuciones Linux populares incluyen un pequeño cargador de primera etapa llamado **shim**, que Microsoft firma con su certificado UEFI para terceros. Shim lleva la clave propia de la distribución y la usa para verificar el siguiente cargador, por lo general GRUB, que a su vez verifica el kernel. Ese diseño deja a Microsoft fuera del ciclo de publicación de cada distribución: Microsoft firma shim pocas veces, y la distribución firma todo lo que viene después. Shim

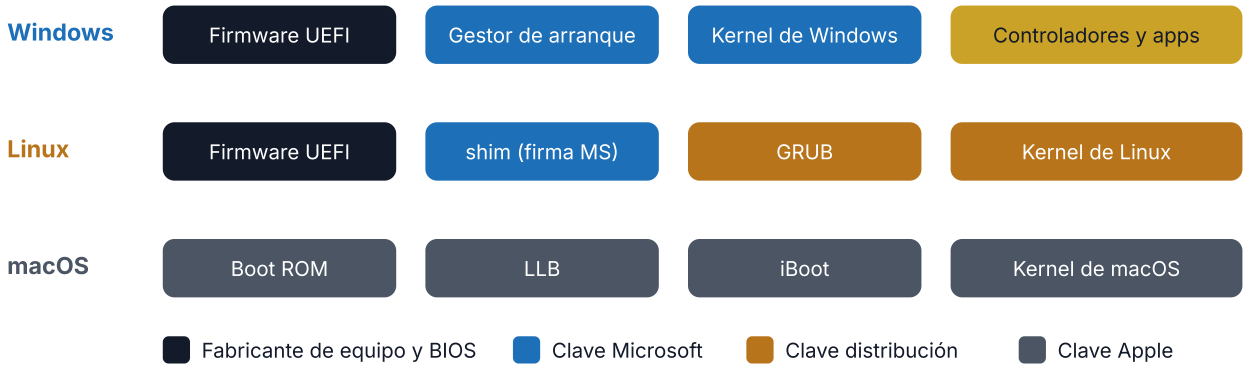
es el camino común, no el único. Un dueño puede registrar sus propias claves en el firmware, arrancar una **imagen de kernel unificada (UKI)**, un solo archivo UEFI firmado que contiene el kernel y sus datos de arranque, directamente desde un firmware que confía en su firmante, o desactivar Secure Boot.

Apple suministra y controla todo el camino en sus propios chips. La Boot ROM se graba cuando se fabrica el chip y no se puede cambiar. Apple la llama la raíz de confianza del hardware. Contiene la clave pública de la autoridad de certificación raíz de Apple (la clave privada nunca sale de Apple) y verifica la siguiente etapa, el Low-Level Bootloader (LLB). En una Mac con Apple silicon, LLB carga la LocalPolicy y verifica iBoot, que verifica el kernel de macOS. El **Secure Enclave**, un procesador de seguridad separado en el mismo chip, arranca su propio sistema operativo pequeño, sepOS, desde su propia boot ROM, y firma el archivo LocalPolicy que registra qué modo de seguridad eligió el dueño.

FIGURA 1.1

## La cadena de confianza y quién controla cada eslabón

**Cada caja verifica la firma de la caja a su derecha antes de ejecutarla.**




En el camino común con shim, una computadora con Linux confía en una firma de Microsoft antes de confiar en su propia distribución; las claves registradas por el dueño, una UKI firmada directamente o Secure Boot desactivado lo evitan. Una Mac en Seguridad total, el modo predeterminado, solo confía en Apple. Una computadora con Windows confía primero en su fabricante y después en Microsoft.

**CÓMO LLEGÓ A SER ASÍ**

El firmware original de las computadoras personales, el BIOS, llegó con la IBM PC en 1981 y se diseñó para un procesador de 16 bits. Sobrevivió décadas gracias a extensiones ingeniosas: las tarjetas de red, por ejemplo, traían ROM opcionales que agregaban el arranque por red, estandarizado por la especificación Preboot Execution Environment (PXE) de Intel, versión 2.1 de 1999. Pero no tenía una forma estándar de verificar que el código que iniciaba estuviera firmado. En 1998 Intel lanzó la Intel Boot Initiative para diseñar un nuevo modelo de arranque para sus servidores Itanium. El resultado se publicó como EFI, que terminó en la versión 1.10. En 2005 la industria formó el Unified EFI Forum, Intel aportó EFI 1.10 y la especificación se convirtió en UEFI, abierta a cualquier procesador. Secure Boot llegó con UEFI 2.3.1, y Microsoft lo volvió un requisito de hardware para las computadoras certificadas para Windows 8 en 2012. Esa decisión explica por qué los certificados de Microsoft están dentro de casi todas las computadoras vendidas desde entonces, y por qué Linux tuvo que inventar shim para arrancar en ellas.

## Microcódigo: el procesador que se puede parchear

Los procesadores modernos son en parte programables. Muchas instrucciones complejas se ejecutan como pequeños programas internos llamados **microcódigo**, y el fabricante del chip puede enviar microcódigo nuevo para cambiar cómo se comporta el procesador después de venderlo. Una actualización de microcódigo no se guarda de forma permanente en el procesador; se carga en cada arranque, ya sea desde el firmware o desde el sistema operativo.

Cada sistema lo recibe de forma distinta. En Linux, el kernel carga el microcódigo temprano desde el disco RAM inicial, con archivos que las distribuciones empaquetan a partir de Intel y AMD; la propia documentación del kernel advierte que cargarlo tarde, con el sistema en marcha, es riesgoso, y está desactivado de forma predeterminada desde el kernel 5.19. En Windows, el microcódigo llega sobre todo dentro de las actualizaciones de firmware del fabricante de la computadora, y Microsoft también ha publicado por su cuenta paquetes de microcódigo de Intel, empezando por KB4090007 en marzo de 2018. En una Mac, Apple entrega el firmware dentro de las actualizaciones de macOS, así que el dueño nunca ve un paso aparte.

## Cuando una falla cruza cuatro dueños

El 3 de enero de 2018, Google Project Zero y varios equipos académicos publicaron **Meltdown** y **Spectre**. Project Zero los había revelado a Intel, AMD y Arm el 1 de junio de 2017. Las fallas no eran errores de ningún sistema operativo. Vivían en la **ejecución especulativa**, la costumbre del procesador de adelantarse para ganar tiempo, que dejaba rastros que un atacante podía medir para leer memoria que nunca debía ver.

Ningún dueño podía corregirlo solo. Los fabricantes de chips publicaron microcódigo. Los equipos de kernel cambiaron la forma en que se asigna la memoria al espacio de direcciones: Linux agregó el aislamiento de tablas de páginas (KPTI), y Windows y macOS publicaron sus propios cambios. Los equipos de compiladores agregaron mitigaciones. Los proveedores de navegadores redujeron la precisión de los temporizadores de JavaScript; Mozilla llevó `performance.now()` a 20 microsegundos y desactivó `SharedArrayBuffer` en Firefox 57.0.4. El primer microcódigo de Intel para Spectre

provocó reinicios inesperados y, el 22 de enero de 2018, Intel pidió a sus clientes que dejaran de desplegarlo hasta que llegaran versiones corregidas. Un CTO que quería saber "¿ya estamos parcheados?" tenía que preguntar a cuatro dueños distintos, con cuatro calendarios distintos.

Las fallas de firmware viajan igual. En julio de 2020 Eclypsium reveló **BootHole** (CVE-2020-10713), un desbordamiento de búfer en la forma en que GRUB2 lee su archivo de configuración. Como el GRUB vulnerable estaba firmado y era de confianza, un atacante podía usarlo para saltarse Secure Boot tanto en máquinas Windows como Linux. La corrección exigió nuevas compilaciones de GRUB y shim de cada distribución, además de actualizaciones de revocación de `dbx` en cada máquina. En diciembre de 2023 Binarly presentó **LogoFAIL**: fallas en los analizadores de imágenes que usa el firmware UEFI para mostrar el logotipo del fabricante al arrancar. Afectaban al firmware de los tres principales proveedores de BIOS, AMI, Insyde y Phoenix. Una imagen de logotipo manipulada se ejecuta demasiado pronto para que Secure Boot la detenga. La única corrección es una nueva compilación de firmware de cada fabricante de computadoras, y cada uno la publicó con su propio calendario.

> *Una falla de firmware tiene tres dueños y una corrección tiene tres fechas de publicación. Su flota queda parcheada en la última.*

### El calendario de vencimientos

Los certificados vencen. Los certificados de Microsoft en los que Secure Boot confía desde 2011 se emitieron con fechas de fin fijas, y esas fechas ya llegaron. No es una falla. Es un calendario, y una flota que no lo lleva lo descubre por las malas.

**RECUADRO FECHADO: CERTIFICADOS DE SECURE BOOT DE 2011 (A NOVIEMBRE DE 2026)**

El artículo de soporte de Microsoft "Windows Secure Boot certificate expiration and CA updates" lista tres certificados que vencen y sus cuatro reemplazos:

- **Microsoft Corporation KEK CA 2011**, que autoriza las actualizaciones de `db` y `dbx`: venció en junio de 2026; lo reemplaza Microsoft Corporation KEK 2K CA 2023.
- **Microsoft UEFI CA 2011**, que firma cargadores de arranque de terceros (incluido shim de Linux) y ROM opcionales: venció en junio de 2026; lo reemplazan Microsoft UEFI CA 2023 y Microsoft Option ROM UEFI CA 2023.
- **Microsoft Windows Production PCA 2011**, que firma el cargador de arranque de Windows: venció en octubre de 2026; lo reemplaza Windows UEFI CA 2023.

Microsoft afirma que los dispositivos sin los certificados de 2023 seguirán arrancando y funcionando, y seguirán recibiendo las actualizaciones normales de Windows, pero ya no podrán recibir nuevas protecciones de seguridad para el proceso de arranque temprano.

Nada se rompe ese día. La máquina sigue arrancando. Lo que se detiene es la capacidad de recibir futuras correcciones a nivel de arranque, como nuevas revocaciones en `dbx` después del próximo BootHole. Una flota que no hace el cambio parece sana y queda congelada en silencio con su vieja lista

de confianza.

**EJEMPLO: EL CALENDARIO DE VENCIMIENTOS DEL OPERADOR DE FLOTA**

Un grupo hotelero opera 4000 quioscos de registro y terminales de punto de venta de tres fabricantes de hardware, comprados a lo largo de seis años. Su equipo de plataforma lleva un calendario compartido con una fila por elemento: los certificados de Secure Boot en el firmware de cada modelo, la fecha de fin de soporte de firmware de cada fabricante, la versión del firmware del TPM, el shim de Linux o el cargador de arranque de Windows de cada imagen, y los certificados de firma de código de sus propias aplicaciones. Cada fila tiene una fecha, un responsable y una prueba: un quiosco de cada modelo en un laboratorio, actualizado con un trimestre de anticipación. Cuando un fabricante deja de publicar firmware para un modelo antiguo, la fila lo dice, y ese modelo pasa al presupuesto de reemplazo. El calendario cuesta unos días al año. Una fila olvidada cuesta una visita técnica a cada ubicación.

## Esta capa en la lista de materiales del píxel

| Componente | Quién lo escribe | Quién paga | Quién puede actualizarlo | Plazo de soporte | ¿Expuesto a entradas no confiables? | Su decisión |
|---|---|---|---|---|---|---|
| Microcódigo de la CPU | Intel, AMD (Apple para sus propios chips) | Incluido en el precio del chip | El fabricante de la computadora en el firmware, o el proveedor del sistema operativo al arrancar; el fabricante del chip lo escribe y lo firma | Lista de soporte del fabricante del chip por familia de procesadores | Indirectamente: Spectre mostró que las páginas web podían sondearlo | Alquilar; darle seguimiento por modelo |
| Firmware UEFI | Base de AMI, Insyde o Phoenix, más código del fabricante de la computadora | Incluido en el precio del dispositivo | El fabricante de la computadora, directamente o mediante Windows Update o LVFS | Plazo de soporte del fabricante de la computadora para ese modelo | Sí: LogoFAIL usó una imagen de logotipo de arranque | Alquilar; comprar modelos con soporte de firmware largo |
| Certificados de Secure Boot | Microsoft (y el fabricante de la computadora para la PK) | Microsoft | Microsoft mediante Windows Update, o el fabricante de la computadora mediante el firmware; Microsoft firma las claves nuevas | El conjunto de 2011 venció en junio y octubre de 2026 | Indirectamente: el firmware analiza los paquetes firmados de actualización de db y dbx, y dbx es la forma de revocar errores de cargador del tipo BootHole | Alquilar; llevar el calendario |
| `shim` y GRUB de Linux | Proyectos de código abierto, compilados por cada distribución | Las distribuciones y sus patrocinadores | Su distribución; Microsoft firma shim | El soporte de la versión de la distribución | Sí: BootHole usó un archivo de configuración de GRUB | Tomarlo de una distribución (alquilar); financiar si depende de ella |

| | | | | | | |
|---|---|---|---|---|---|---|
| Boot ROM e iBoot de Apple | Apple | Incluido en el precio de la Mac | Solo Apple (la Boot ROM no se puede actualizar) | El soporte de macOS de Apple para esa Mac | Limitado, y dentro del control de Apple | Alquilar |

**LENTE DE SEGURIDAD**

**¿Quién puede cambiarlo?** El fabricante del chip, el proveedor de BIOS, el fabricante de la computadora y quien tiene las claves de arranque, por lo general Microsoft. En una Mac, solo Apple. Es posible agregar confianza sin reemplazar la Platform Key: en Linux, shim permite que el dueño de una máquina registre una Machine Owner Key (MOK) y arranque kernels y módulos firmados con ella. Lo que una flota mediana no puede hacer en la práctica es parchear el firmware o volver a firmar las claves de los proveedores.

**¿Qué tan rápido se puede corregir?** Despacio, y al ritmo del dueño más lento. Una corrección de firmware debe pasar del proveedor de BIOS al fabricante de la computadora y luego a sus máquinas, modelo por modelo. Algunas piezas, como la Boot ROM de Apple, no se pueden parchear nunca; el sistema está diseñado para que las etapas posteriores cubran sus límites.

**¿Qué cambia la IA aquí?** El análisis de binarios asistido por IA abarata la búsqueda en el código de firmware compartido, como los analizadores de logotipos que afectó LogoFAIL, pero una corrección todavía viaja del proveedor de BIOS al fabricante de la computadora y luego a sus máquinas, modelo por modelo. Hay que planear para fallas que se encuentran más rápido de lo que esa cadena puede publicar correcciones.

**IMPLICACIONES PARA EL NEGOCIO**

**Seguridad.** La empresa regulada debe poder demostrar qué arrancó. Eso significa mediciones del TPM, Secure Boot activado y un `dbx` actualizado en cada máquina. Un auditor que pregunta "¿cómo saben que sus portátiles confían en los certificados de 2023?" merece un informe, no una suposición.

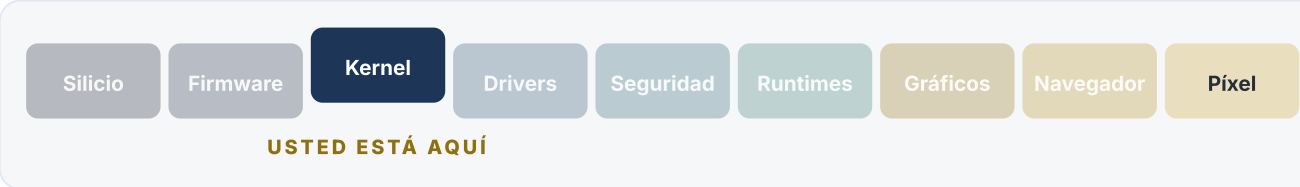
**Costo.** Para el operador de una flota de dispositivos, el precio de un quiosco incluye su plazo de soporte de firmware. Un modelo un poco más barato que pierde las actualizaciones de firmware tres años antes cuesta más en una vida útil de siete años, porque termina en un viaje de un técnico o en un reemplazo anticipado.

**Valor.** El fabricante de software no puede poseer esta capa y no debería intentarlo. Su valor está en conocerla: probar su instalador y sus controladores en máquinas con los certificados nuevos, y avisar pronto a sus clientes cuando un cambio de arranque los vaya a afectar.

**Este trimestre:** reunir en un solo calendario cada certificado, fecha de soporte de firmware y clave de arranque de la que dependen sus máquinas, con un responsable por fila.

CAPÍTULO DOS

# El kernel




El 19 de julio de 2024, un archivo de configuración de un producto de seguridad llegó a máquinas Windows de todo el mundo, y unos 8,5 millones de ellas se detuvieron en una pantalla azul. Aeropuertos, hospitales, hoteles y tiendas se apagaron esa misma mañana. El archivo no era parte de Windows, y ninguna de las empresas afectadas lo escribió. Era código alquilado que se ejecutaba en el único lugar donde un solo error lo detiene todo: el kernel.

## Qué hace un kernel

El **kernel** es el corazón del sistema operativo: el programa que se ejecuta con privilegio total sobre el hardware, en el anillo 0 en x86 o en EL1 en Arm, como explicó el capítulo 1. Todo lo demás le pide ayuda. Cumple cuatro funciones principales.

- **Procesos e hilos.** Un **proceso** es un programa en ejecución con su propia memoria privada. Un **hilo** es una línea de trabajo dentro de él. El kernel los crea, les da seguimiento y los termina.
- **Memoria virtual.** El kernel escribe las tablas de páginas que usa la unidad de gestión de memoria, para que cada proceso vea su propio mapa privado y no pueda leer los datos de otro.
- **Planificación.** Una máquina tiene unos pocos núcleos de procesador y miles de hilos. El **planificador** decide qué hilo se ejecuta en qué núcleo, y por cuánto tiempo, muchas veces por segundo.
- **Llamadas al sistema y controladores.** Los programas llegan a los archivos, la red, la pantalla y cada dispositivo mediante **llamadas al sistema**, la puerta de entrada del kernel. Detrás de esa puerta están los **controladores (drivers)**, el código que habla con cada pieza de hardware.

El kernel también es un muro. La documentación de controladores de Microsoft enuncia la regla que rige este capítulo: todo el código en modo kernel comparte un solo espacio de direcciones, así que un controlador de kernel no está aislado de otros controladores ni del sistema operativo, y si falla, falla todo el sistema operativo. Una aplicación por lo general falla sola. El código del kernel no tiene ese aislamiento de procesos: un error en él puede detener o corromper todo el sistema. Linux a veces puede sobrevivir a una falla del kernel, llamada oops, deteniendo solo la tarea que la provocó, pero entonces el kernel se marca como contaminado (tainted), y su estado puede dejar de ser confiable.

## Tres kernels, tres diseños

Los diseñadores de kernels discuten cuánto código debe quedar detrás del muro. Un kernel **monolítico** pone la base del sistema y todos sus servicios, incluidos sistemas de archivos, red y controladores, en un solo programa en modo kernel. Un **microkernel** deja solo un núcleo pequeño en modo kernel y ejecuta los servicios como programas separados fuera de él. Eso permite aislar la falla de un servicio, a cambio de cruzar el muro más veces; cuánto cuestan esos cruces depende del diseño, así que ningún modelo es más seguro ni más lento por definición. Los tres sistemas de este libro eligieron distinto.

**Linux** es monolítico, con **módulos cargables**: los controladores y sistemas de archivos se pueden cargar en el kernel en ejecución cuando hacen falta, pero una vez cargados se ejecutan con privilegio total como el resto. **XNU**, el kernel de macOS, es en palabras de Apple un kernel híbrido que combina el kernel Mach desarrollado en Carnegie Mellon University con componentes de FreeBSD y una API de controladores en C++ llamada IOKit. La parte Mach aporta ideas de microkernel como el paso de mensajes; la parte BSD aporta el modelo de procesos de Unix y la red; todo se ejecuta junto en modo kernel. **Windows NT** también suele llamarse híbrido: un kernel pequeño en la base, un "ejecutivo" de servicios encima y controladores al lado, todo en modo kernel, con el núcleo de ventanas y gráficos (`win32k`) también en el kernel.

| Tema | Windows (kernel NT) | Linux | macOS (XNU) |
|---|---|---|---|
| **Diseño** | Híbrido: kernel más ejecutivo, controladores en modo kernel | Monolítico con módulos cargables | Híbrido: Mach más BSD más IOKit |
| **Quién controla el código fuente** | Microsoft (cerrado) | Mantenedores upstream, encabezados por Linus Torvalds (GPLv2) | Apple (código publicado para partes de Darwin) |
| **Modelo de planificación** | Apropiativo, 32 niveles de prioridad (0 a 31), por turnos dentro de cada nivel | EEVDF desde Linux 6.6 (octubre de 2023), en reemplazo de CFS | Hilos Mach, guiados por clases de calidad de servicio de las aplicaciones |
| **Código de terceros en el kernel** | Común: controladores, agentes de seguridad, anti-cheat | Común: módulos de fabricantes de hardware y proveedores de seguridad | Restringido: extensiones de kernel declaradas obsoletas (deprecated) desde 2019 |
| **Hacia dónde se mueve el código externo** | Modo usuario, bajo la Windows Resiliency Initiative | No hay una política única; upstream prefiere controladores integrados en el árbol | System Extensions y DriverKit, en espacio de usuario |
| **Rust en el kernel** | Partes de `win32k` (regiones GDI) desde compilaciones de 2023 | Soportado desde 6.1 (diciembre de 2022); controlador Rust Binder en 6.18 (noviembre de 2025); experimento declarado concluido en diciembre de 2025 | No anunciado |

## Planificación: quién recibe el procesador

Cada kernel debe responder la misma pregunta miles de veces por segundo: ¿qué hilo se ejecuta después? Las respuestas definen cómo se siente un sistema.

Windows usa prioridades. Microsoft documenta 32 niveles, del 0 al 31. El planificador reparte porciones de tiempo por turnos entre todos los hilos listos del nivel más alto, y un hilo de mayor prioridad que queda listo detiene de inmediato a uno de menor prioridad. Linux usa un modelo de equidad desde 2007. En la versión 6.6, publicada el 29 de octubre de 2023, reemplazó el Completely Fair Scheduler por **EEVDF** (Earliest Eligible Virtual Deadline First), que lleva la cuenta del tiempo de procesador que se le debe a cada tarea y da el siguiente turno a la que tiene el plazo más cercano entre las elegibles. macOS pide a las aplicaciones que etiqueten su trabajo con **clases de calidad de servicio**, desde trabajo interactivo con el usuario, como las animaciones, hasta trabajo en segundo plano, como la indexación, y el sistema usa esas etiquetas para fijar la planificación, el rendimiento del procesador y del disco y el comportamiento de los temporizadores.

Un CTO rara vez ajusta un planificador directamente. Pero el planificador decide si la respuesta táctil del quiosco sigue fluida mientras se ejecuta un análisis antivirus, y si la fuente de datos de mercado de una mesa de operaciones espera detrás de una tarea de respaldo. Saber qué modelo usa su plataforma le dice qué ajustes existen.

## Código alquilado en el anillo 0

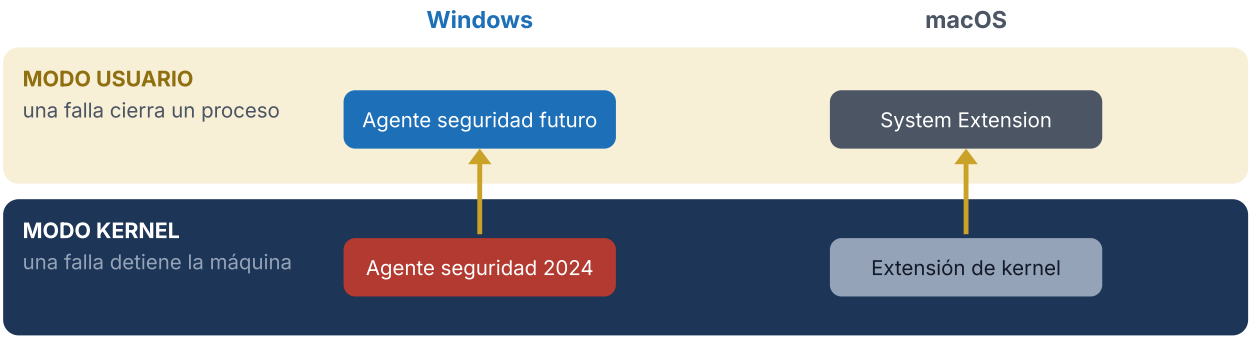
Un kernel en ejecución también aloja código que su dueño no escribió. Los controladores gráficos, de almacenamiento y de red, las herramientas de virtualización, los agentes de respaldo, el software anti-cheat y los productos de seguridad de los equipos (endpoints) cargan componentes en el kernel. Cada uno es código alquilado con privilegio total. Cuando falla, falla para toda la máquina.

La caída de julio de 2024 es el caso más claro. El resumen técnico de la propia CrowdStrike dice que a las 04:09 UTC del 19 de julio de 2024 publicó una actualización de configuración del sensor para sistemas Windows, un archivo conocido como Channel File 291. La actualización tenía un error de lógica. El componente de kernel del sensor Falcon la leyó y falló, y con él el sistema operativo. CrowdStrike revirtió el archivo a las 05:27 UTC, 78 minutos después. Para entonces, las máquinas que lo habían recibido estaban atascadas: fallaban en cada arranque antes de poder descargar la corrección, y muchas necesitaron a una persona frente al teclado. Microsoft estimó que resultaron afectados 8,5 millones de dispositivos Windows, menos del uno por ciento de todas las máquinas Windows.

Se desprenden tres lecciones. Primera, el contenido que provocó la falla no era un controlador nuevo; era un archivo de datos leído por un controlador que ya tenía la confianza del sistema. Un proceso de revisión solo de código no lo habría detectado. Segunda, la velocidad del canal de actualización del proveedor, normalmente una ventaja, llevó la falla a todas partes a la vez. Tercera, la recuperación fue manual, así que su costo creció con el número de máquinas y de ubicaciones.

FIGURA 2.1

**Dónde se ejecuta el código de seguridad de terceros, antes y después**




Anunciado nov. 2024; GA en Windows 11, nov. 2026  Declarado obsoleto desde macOS 10.15 (2019)

En Linux nadie puede hacer este cambio por todos los proveedores.

Apple sacó primero el código externo de su kernel, porque controla toda la plataforma. Microsoft sigue sus pasos, con socios que necesitan tiempo para rediseñar. Linux deja la decisión a cada proveedor y a cada distribución.

## Sacar el código alquilado del kernel

Apple se movió primero. En su conferencia de desarrolladores de 2019, con macOS Catalina 10.15, presentó **System Extensions** y **DriverKit**, que permiten crear controladores y herramientas de seguridad que se ejecutan en espacio de usuario en lugar del kernel. La página para desarrolladores de Apple indica que las extensiones de kernel fueron declaradas obsoletas (deprecated) y que, a partir de macOS Big Sur (2020), macOS ya no carga de forma predeterminada las extensiones de kernel que usan interfaces declaradas obsoletas. Las que todavía cargan no pueden hacerlo bajo demanda: necesitan la aprobación de un administrador y un reinicio, y en una Mac con Apple silicon el dueño primero debe bajar la política de seguridad de arranque a Seguridad reducida en recoveryOS y permitir extensiones de kernel de desarrolladores identificados. Los productos de seguridad pasaron a la interfaz **Endpoint Security** y los filtros de red a **Network Extension**. Los desarrolladores necesitan un permiso (entitlement) de Apple para usarlas. Apple pudo hacerlo porque controla la lista de hardware, el kernel y el programa de desarrolladores.

El camino de Microsoft es más difícil, porque Windows se ejecuta en hardware de miles de fabricantes y una gran industria de seguridad construyó sus productos en el kernel. En noviembre de 2024, cuatro meses después de la caída, Microsoft anunció la **Windows Resiliency Initiative**. Su plan declarado incluye nuevas capacidades de Windows "que permitirán a los desarrolladores de productos de seguridad construirlos fuera del modo kernel", para que el antivirus y la protección de los equipos (endpoints) puedan ejecutarse en modo usuario, como las aplicaciones. En junio de 2025 Microsoft dijo que entregaría al mes siguiente una vista previa privada de la plataforma de seguridad de los equipos (endpoints) de Windows a los socios de su Microsoft Virus Initiative. En Ignite, en noviembre de 2025, dijo que la API Windows Endpoint Security Platform estaba en vista previa con esos socios, que ayudaban a que estuviera disponible de forma general en 2026. El 17 de noviembre de 2026, en Ignite, Microsoft anunció que la plataforma "está alcanzando la disponibilidad general en Windows 11". La

disponibilidad general de la plataforma no equivale a que el producto de cada proveedor se ejecute en ella: cada proveedor de seguridad todavía tiene que entregar una versión que funcione fuera del kernel. Conviene tratarlo como una migración en curso: preguntar a cada proveedor con componentes en el kernel si su producto ya se ejecuta en la nueva plataforma, y cuándo lo hará.

Linux no tiene un movimiento único equivalente. Sus mantenedores upstream prefieren controladores integrados en el árbol principal del kernel, donde se revisan y se mantienen al día; los módulos fuera del árbol corren por cuenta del proveedor. Las herramientas de seguridad usan cada vez más interfaces del kernel que limitan lo que puede hacer el código externo, pero cada proveedor elige su propio diseño, y cada distribución su propia política.

**CÓMO LLEGÓ A SER ASÍ**

Los tres kernels descienden de ideas de los años sesenta a los ochenta. Unix nació en Bell Labs en 1969, cuando Ken Thompson y Dennis Ritchie empezaron a trabajar en una computadora PDP-7 poco usada. Su modelo de procesos y de archivos moldeó tanto a BSD como a Linux. En Carnegie Mellon University, el Proyecto Mach funcionó de 1985 a 1994 y exploró un microkernel capaz de alojar Unix como un servicio. NeXT construyó su sistema operativo sobre Mach y BSD, y cuando Apple compró NeXT, ese diseño se convirtió en Darwin y XNU; Mac OS X 10.0 salió el 24 de marzo de 2001. En Microsoft, un equipo dirigido por Dave Cutler, contratado de Digital Equipment Corporation en 1988, empezó un nuevo kernel portable que salió como Windows NT 3.1 en julio de 1993. NT 4.0 y Windows 2000 se basaron en esa línea, y también todas las versiones de Windows para consumidores desde Windows XP. Linux empezó el 25 de agosto de 1991, cuando Linus Torvalds le contó a un grupo de noticias que estaba escribiendo un sistema operativo libre que era "solo un pasatiempo". Décadas después, esas tres decisiones todavía deciden quién puede cambiar su kernel.

## Rust llega al kernel

La mayor parte del código de kernel está escrito en C, un lenguaje que deja la seguridad de memoria en manos del programador. Una gran parte de las fallas de seguridad graves son errores de memoria que C permite y que los lenguajes más seguros evitan. **Rust** verifica el uso de memoria cuando se compila el código. No tiene recolector de basura ni máquina virtual, así que puede trabajar a nivel de kernel.

Linux integró el soporte inicial de Rust en la versión 6.1, publicada el 11 de diciembre de 2022. Era infraestructura, no controladores: el andamiaje que permitió escribir controladores posteriores en Rust. En diciembre de 2025, en la cumbre de mantenedores del kernel, el proyecto declaró concluido y exitoso el experimento de Rust; el parche que quitó la etiqueta de "experimental" decía, en palabras de su autor Miguel Ojeda, que Rust llegó para quedarse. Agregó que eso no significa que todo funcione en cada configuración, arquitectura y cadena de herramientas (toolchain). El primer controlador grande en Rust ya había llegado: Linux 6.18, publicado el 30 de noviembre de 2025, agregó una versión en Rust de Binder, el controlador de comunicación entre procesos que usa Android.

Microsoft tomó una ruta más acotada. En julio de 2023, la build 25905 de Windows Insider incluyó `win32kbase_rs.sys`, una nueva implementación en Rust de las regiones GDI, parte del núcleo gráfico dentro del kernel de Windows. Microsoft dijo que seguiría aumentando el uso de Rust en el kernel. El

componente llegó después a Windows 11 versión 24H2, y dejó una lección instructiva. En 2025 Check Point Research reveló que un metarchivo EMF+ malformado, un formato gráfico de Windows, podía hacer caer equipos 24H2 a través de ese código en Rust. Rust hizo su trabajo: detectó un acceso fuera de límites y se detuvo en lugar de corromper la memoria. Pero en el kernel, detenerse significa una caída del sistema. Microsoft lo calificó como un problema moderado de denegación de servicio y lo corrigió en 2025. La seguridad de memoria bloquea clases completas de corrupción de memoria, así que una falla que pudo haber sido una toma de control se convirtió en una caída. No detiene los errores de lógica, y no vuelve inofensivo el código de kernel.

> *La seguridad de memoria puede convertir una toma de control en una caída. En el kernel, una caída sigue siendo una interrupción del servicio.*

### Por qué la mayoría de las empresas alquila el kernel, y lo ajusta

Casi ninguna empresa debería escribir su propio kernel. Los tres de este libro acumulan décadas de soporte de hardware, respuesta de seguridad y compatibilidad que ningún equipo mediano puede igualar. La pregunta útil no es "¿poseer o alquilar?", sino "¿cuánto ajustamos y cuánto código externo dejamos entrar?"

Ajustar no es poseer en el sentido de este libro, que significa bifurcar o construir, pero es control real a bajo costo. Un equipo puede elegir una línea de kernel de soporte a largo plazo en lugar de la más nueva, definir opciones de planificación y de memoria para su carga de trabajo, quitar los módulos que no necesita y decidir qué controladores y agentes de terceros pueden cargarse. En Windows, las herramientas de políticas pueden bloquear controladores sin firma o no deseados. En macOS, el perfil de administración de dispositivos decide qué System Extensions se permiten. En Linux, un equipo puede compilar el código fuente del kernel de la distribución con su propia configuración, que es el mayor control de los tres y también el que más trabajo cuesta mantener al día. Solo se convierte en una bifurcación cuando el equipo empieza a mantener sus propios parches.

**EJEMPLO: UN INVENTARIO DEL KERNEL**

Un banco regulado lista cada componente que se carga en el kernel de sus imágenes estándar de portátiles y servidores: los controladores de cada fabricante de hardware, su agente de seguridad de los equipos (endpoints), su cifrado de disco, sus herramientas de virtualización. Para cada uno registra el proveedor, cómo llegan las actualizaciones (y si las actualizaciones de contenido se escalonan o se envían a todas las máquinas a la vez), si el proveedor tiene en camino una versión en modo usuario y cómo se recupera una máquina que no arranca sin una visita. La lista tiene menos de veinte filas. Se convierte en la respuesta del banco cuando un auditor pregunta: "¿quién puede cambiar código en su kernel, y con qué calendario?"

## Esta capa en la lista de materiales del píxel

| Componente | Quién lo escribe | Quién paga | Quién puede actualizarlo | Plazo de soporte | ¿Expuesto a entradas no confiables? | Su decisión |
|---|---|---|---|---|---|---|
| Kernel de Windows NT | Microsoft | Microsoft, mediante licencias | Solo Microsoft | Las fechas de servicio de la versión de Windows | Sí: archivos, red, fuentes y gráficos llegan al código del kernel | Alquilar; controlar qué controladores se cargan |
| Kernel de Linux | Mantenedores upstream y miles de colaboradores, muchos pagados por empresas | Las empresas que contribuyen; las distribuciones | Su distribución, o usted si compila su propio kernel | La línea de soporte a largo plazo que elija | Sí | Tomarlo de una distribución (alquilar); ajustar; financiar si es crítico |
| XNU | Apple | Incluido en el precio de la Mac | Solo Apple | El soporte de macOS de Apple para esa Mac | Sí | Alquilar |
| Agente de seguridad de los equipos (endpoints) en el kernel | El proveedor de seguridad | Usted, por suscripción | El proveedor, a menudo muchas veces al día | Su contrato | Sí: por diseño inspecciona archivos y tráfico no confiables | Alquilar; exigir despliegue escalonado y un plan de modo usuario |
| Controladores de hardware (GPU, almacenamiento, red) | Fabricantes de hardware | Incluido en el precio del hardware | El fabricante, a través del proveedor del sistema operativo o de su propia herramienta | El soporte del fabricante para ese dispositivo | A menudo: los controladores de red y de gráficos procesan (parsing) datos externos | Alquilar; estandarizar modelos para reducir la lista |

LENTE DE SEGURIDAD

**¿Quién puede cambiarlo?** El proveedor del sistema operativo y cada proveedor cuyo controlador o agente se permita cargar. En Windows y Linux esa lista suele ser más larga de lo que nadie ha puesto por escrito. En macOS es corta por diseño.

**¿Qué tan rápido se puede corregir?** El proveedor del sistema operativo corrige su kernel con un calendario mensual o por versión. Un proveedor con código en el kernel puede enviar cambios en minutos, lo que es bueno ante amenazas nuevas y peligroso ante errores nuevos. Si aplicar una corrección exige arrancar un equipo que no arranca, el plazo depende de cuánto tarde alguien en llegar hasta él.

**¿Qué cambia la IA aquí?** Las herramientas de IA ayudan a traducir el C del kernel a Rust y a buscar errores de memoria en controladores de terceros, pero la caída de 24H2 muestra que el código más seguro todavía necesita revisión y un responsable de guardia. La ganancia mayor es tener menos código externo en el kernel, y eso no lo da ninguna herramienta.

IMPLICACIONES PARA EL NEGOCIO

**Seguridad.** Para la empresa regulada, cada componente de su kernel es una pregunta de auditoría. Hay que mantener el inventario, exigir a cada proveedor con código en el kernel que escalone sus actualizaciones y pedir un plan escrito para pasar a modo usuario a medida que Windows y macOS lo hagan posible.

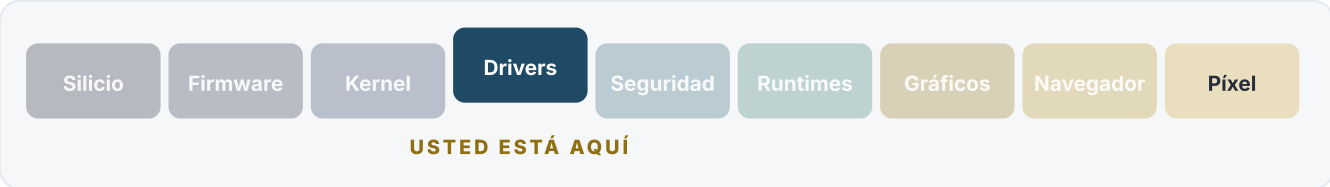
**Costo.** El operador de una flota de dispositivos pagó julio de 2024 en visitas a las instalaciones, no en licencias: una máquina que no puede arrancar no puede recuperarse sola. Conviene presupuestar la recuperación remota (administración fuera de banda, imágenes de recuperación, claves de cifrado accesibles) antes de necesitarla. Cuesta mucho menos que una mañana con los mostradores de registro apagados.

**Valor.** El fabricante de software gana al mantener su propio código fuera del kernel. Un producto que se ejecuta en espacio de usuario, mediante interfaces soportadas, es más fácil de entregar en los tres sistemas, sobrevive a las actualizaciones del sistema operativo y no supone un riesgo en los equipos de sus clientes.

**Primer paso:** listar todo lo que se carga en su kernel, y en cada fila anotar quién puede enviar un cambio y cómo vuelve a funcionar una máquina que no arranca.

CAPÍTULO TRES

# Controladores, almacenamiento y red




El kernel decide qué se ejecuta. Los controladores deciden lo que el hardware realmente hace. Cada fotograma que ve su cliente, cada registro que guarda su sistema y cada paquete que envía pasa por código escrito por una empresa que no es usted y, muy a menudo, tampoco el proveedor del sistema operativo. Este capítulo trata de esos dueños, y de la única ruta de datos que algunas empresas deciden poseer por completo.

Un **controlador (driver)** es el código que traduce entre el kernel y una pieza de hardware: un controlador de disco, una tarjeta de red, un procesador gráfico, un lector de huellas. Como toca el hardware directamente, un controlador se ha ejecutado tradicionalmente dentro del kernel con privilegios totales, así que un solo controlador defectuoso puede tumbar toda la máquina. Los tres sistemas respondieron a ese riesgo de tres maneras muy distintas, y cada respuesta indica qué ingenieros reparan un error de controlador, y según qué calendario.

## Tres modelos de controladores, tres acuerdos

**Windows** se ejecuta en hardware de miles de fabricantes, así que Microsoft tuvo que dar a terceros una entrada al kernel estable y documentada. El antiguo **Windows Driver Model (WDM)** es potente, pero es famoso por lo difícil que resulta usarlo bien. Microsoft agregó encima los **Windows Driver Frameworks (WDF)**: el Kernel-Mode Driver Framework (KMDF) para controladores que deben ejecutarse en el kernel, y el User-Mode Driver Framework (UMDF) para los que pueden ejecutarse como procesos comunes. Microsoft publicó el código fuente de WDF en GitHub en marzo de 2015, unos meses antes de que saliera Windows 10.

La estabilidad vino con una barrera. A partir de Windows 10, versión 1607, una instalación nueva con Secure Boot activado no carga un controlador nuevo en modo kernel a menos que lo haya firmado el propio portal para desarrolladores de Microsoft y, como la documentación de controladores de Microsoft todavía indica a noviembre de 2026, la cuenta del portal debe tener registrado al menos un certificado de firma de código de validación extendida (EV). En otras palabras, un tercero escribe el controlador, pero Microsoft decide si puede ejecutarse en el kernel.

**Linux** hizo el acuerdo contrario. El kernel promete a las aplicaciones una interfaz de llamadas al sistema estable, pero no promete a los controladores ninguna interfaz estable dentro del kernel. El documento del kernel `stable-api-nonsense`, escrito por Greg Kroah-Hartman, dice sin rodeos que Linux "no tiene una interfaz binaria del kernel, ni tiene una interfaz estable del kernel". Las interfaces internas cambian cada vez que los desarrolladores corrigen errores o mejoran diseños. El consejo a los fabricantes de hardware es directo: "lleve su controlador de kernel al árbol principal del kernel". Un

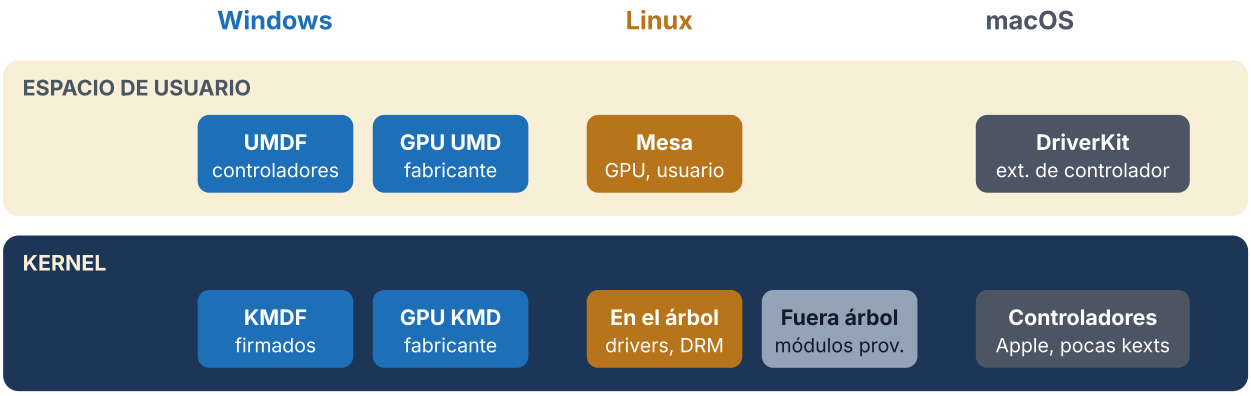
controlador que vive **dentro del árbol** lo actualiza quien cambia la interfaz de la que depende. Un controlador que se mantiene **fuera del árbol** se rompe según el calendario de su dueño, no según el de usted.

**macOS** tomó un tercer camino, posible porque Apple controla la mayor parte de su propio hardware. **DriverKit** saca muchos controladores del kernel al espacio de usuario como "extensiones de controlador", y el capítulo 2 cuenta cómo Apple retiró las extensiones de kernel a las que reemplazan. Una extensión de controlador que falla se puede reiniciar como cualquier otro proceso.

| Pregunta | Windows | Linux | macOS |
|---|---|---|---|
| **Interfaz de controladores** | WDM, con KMDF y UMDF encima; estable entre versiones | Sin interfaz estable dentro del kernel; los controladores van en el árbol del kernel | DriverKit en espacio de usuario; extensiones de kernel declaradas obsoletas (deprecated) |
| **Quién escribe la mayoría de los controladores** | Los fabricantes de hardware | Fabricantes y desarrolladores del kernel, upstream | Apple, más los fabricantes de dispositivos adicionales |
| **Quién permite que se ejecute un controlador** | Microsoft firma los controladores de kernel (Windows 10, 1607 y posteriores) | La distribución que compila el kernel, y el dueño de la máquina | Apple, mediante la firma de código y la aprobación de extensiones del sistema |
| **Ruta de corrección de un error de controlador** | El fabricante la publica; Windows Update puede distribuirla | Parche upstream, luego el kernel de cada distribución | La actualización del sistema de Apple, o la extensión del fabricante |

FIGURA 3.1

## Dónde se ejecuta un controlador




El código en la banda oscura puede detener la máquina. El de la banda clara se puede reiniciar.

El mismo trabajo, ubicado de tres maneras. Windows divide los controladores por framework, Linux mantiene la mayoría en el árbol del kernel y macOS los saca al espacio de usuario donde puede.

## El controlador de la GPU: un dueño aparte

El controlador gráfico merece su propia línea en cualquier inventario, porque en dos de los tres sistemas pertenece a una empresa que no es usted ni el proveedor del sistema operativo.

En Windows, Microsoft presentó el **Windows Display Driver Model (WDDM)** con Windows Vista. WDDM divide el controlador gráfico en dos: un controlador de pantalla en modo usuario que hace la mayor parte del trabajo pesado, y un controlador simplificado en modo kernel. También trajo la **detección y recuperación de tiempos de espera** (timeout detection and recovery), que permite a Windows reiniciar un controlador de pantalla colgado sin reiniciar la máquina. El fabricante de la GPU escribe ambas mitades y las publica con su propio calendario.

En Linux, la parte de kernel de la mayoría de los controladores gráficos vive en el árbol del kernel bajo el Direct Rendering Manager (DRM), y la parte de usuario vive en **Mesa**, un proyecto abierto que implementa OpenGL y Vulkan para muchas GPU. La gran excepción ha sido NVIDIA, cuyo controlador fue cerrado durante mucho tiempo. En mayo de 2022 NVIDIA publicó módulos de kernel de GPU de código abierto, con licencia doble GPL y MIT, y sus controladores posteriores pasaron por completo a ellos para las GPU soportadas. Todavía se distribuyen fuera del árbol del kernel.

En macOS con Apple silicon, Apple diseña la GPU y escribe su controlador. Un dueño, un canal de actualización. Es más simple, y también significa que ninguna otra parte puede enviarle una corrección.

> **RECUADRO: LA ALTERNATIVA SILENCIOSA**
>
> Windows y Linux mantienen una ruta de dibujo por software documentada para cuando falla la ruta de la GPU. Windows tiene WARP, un rasterizador por software que Microsoft documenta para usar "cuando no hay un controlador de video disponible o no funciona correctamente". Linux tiene llvmpipe de Mesa, un rasterizador por software que usa la CPU. En macOS, Apple documenta un renderizador por software solo para su framework OpenGL, que recurre a él salvo que una aplicación pida únicamente hardware; OpenGL está declarado obsoleto (deprecated) desde macOS 10.14, y Apple no documenta una alternativa así para Metal.
>
> El riesgo es que la alternativa es silenciosa. Después de una actualización de controlador, una GPU nueva en una imagen de nube o un cambio en la plantilla de una máquina virtual, una aplicación puede terminar dibujando en la CPU. La pantalla sigue funcionando. Nada falla, así que el monitoreo de errores no ve nada. Lo que cambia es el costo: sube el uso de CPU, los fotogramas tardan más, la batería se agota más rápido y una flota de escritorios en la nube o de quioscos necesita máquinas más grandes.
>
> La solución es barata. Registrar qué renderizador y qué ruta de datos usa realmente una máquina, como métrica y no como línea de registro, y alertar cuando cambie después de una actualización.

## Sistemas de archivos: donde se da forma a sus datos

Un **sistema de archivos** decide cómo los bytes se convierten en archivos en un disco: cómo se organizan, cómo se sobrevive a una caída y si el sistema puede saber cuándo los datos se han degradado en silencio.

**Windows** usa **NTFS** para casi todo, incluido el volumen de arranque. Microsoft anunció el **Resilient File System (ReFS)** en enero de 2012 y lo entregó con Windows Server 2012. ReFS guarda sumas de verificación de sus metadatos y, de forma opcional, de los datos de los archivos, así que puede detectar corrupción y, con almacenamiento en espejo, repararla mientras el volumen sigue en línea. A noviembre de 2026, la documentación de ReFS de Microsoft todavía indica que no es de arranque y que no se admite en medios extraíbles, y ninguna versión publicada de Windows arranca desde él, así que NTFS sigue siendo el predeterminado de uso general.

**Linux** ofrece una elección, que es a la vez una fortaleza y una decisión que hay que tomar. **ext4** se declaró estable en Linux 2.6.28, publicado el 24 de diciembre de 2008, y sigue siendo un predeterminado común. **XFS**, que nació en los sistemas IRIX de Silicon Graphics, es un sistema de archivos con registro por diario (journaling) y múltiples hilos, diseñado para archivos grandes y sistemas de archivos grandes. **Btrfs** se integró en el kernel principal en enero de 2009, durante el ciclo 2.6.29, y agrega copia en escritura (copy-on-write), sumas de verificación para datos y metadatos, e instantáneas. Cada distribución elige un predeterminado, así que dos máquinas Linux de la misma flota pueden guardar datos de maneras distintas.

**macOS** pasó al **Apple File System (APFS)**, el predeterminado en las Mac desde macOS High Sierra (10.13) en 2017. Apple lo diseñó para almacenamiento flash, con cifrado nativo como capacidad del sistema de archivos, metadatos con copia en escritura, clones de archivos baratos e **instantáneas** de solo lectura de un volumen completo. En una Mac con Apple silicon o con chip T2, el volumen interno sigue cifrado incluso con FileVault desactivado, pero las credenciales del usuario no lo protegen: Apple dice que en ese caso la clave solo está protegida por un identificador de hardware en el Secure Enclave. **FileVault** es el ajuste que vincula la clave a la contraseña del usuario, y por eso decide si los datos están protegidos en reposo frente a alguien que tiene la Mac en sus manos.

| Función | Windows | Linux | macOS |
|---|---|---|---|
| **Sistema de archivos predeterminado** | NTFS | ext4 o XFS, según la distribución; Btrfs en algunas | APFS |
| **Instantáneas** | Volume Shadow Copy; instantáneas a nivel de archivo de ReFS en Server 2022 y posteriores | Btrfs, o el gestor de volúmenes | Instantáneas de APFS |
| **Cifrado en reposo** | BitLocker, debajo del sistema de archivos | dm-crypt, debajo del sistema de archivos | Integrado en APFS; FileVault vincula la clave a la contraseña del usuario |

## La ruta de red: sockets, anillos y programas en el kernel

Casi todos los programas que se comunican por red todavía usan una idea de 1983. La interfaz de **sockets** llegó con 4.2BSD Unix y trata una conexión de red casi como un archivo que se puede leer y escribir. Linux y macOS usan directamente los sockets BSD. Windows adoptó la misma idea con

**Windows Sockets**, o Winsock, cuya especificación versión 1.1 tiene fecha del 20 de enero de 1993 y sigue el modelo de la interfaz de Berkeley.

Los sockets tienen un costo: cada lectura o escritura es una llamada al sistema, un viaje al kernel y de regreso. En servidores que manejan millones de operaciones por segundo, esos viajes se acumulan. En Linux 5.1, publicado en mayo de 2019, **io_uring** de Jens Axboe agregó otro modelo: la aplicación y el kernel comparten en memoria dos búferes en anillo, uno para solicitudes y otro para resultados, de modo que un programa puede enviar muchas operaciones con pocas llamadas al sistema. Empezó con el almacenamiento y creció hasta cubrir la red. También agregó una gran superficie nueva dentro del kernel, lo que importa para la lente de seguridad más adelante.

Apple fue en la dirección contraria para los desarrolladores de aplicaciones. En la WWDC 2018 presentó **Network.framework** como "una alternativa moderna a los sockets": una interfaz de más alto nivel que maneja por defecto el establecimiento de conexiones, el cifrado TLS y los cambios de red, y sobre la que está construido el propio sistema de carga de URL de Apple. Se cede algo de control y se gana un comportamiento que Apple mantiene al día.

La tercera idea es **eBPF**: pequeños programas que se cargan en el kernel de Linux en tiempo de ejecución, que el kernel revisa con un verificador antes de ejecutarlos ante eventos como la llegada de un paquete. Los equipos usan eBPF para filtrado rápido de paquetes, balanceo de carga y observabilidad sin escribir un módulo de kernel. Microsoft anunció el proyecto **eBPF for Windows** en mayo de 2021, un esfuerzo de código abierto que reutiliza herramientas eBPF existentes en Windows. A noviembre de 2026 su propio repositorio todavía lo describe como un trabajo en curso, así que aún no reemplaza el conjunto de mecanismos de Linux.

### Quién controla la ruta de datos

La **ruta de datos** es el recorrido de una solicitud desde la tarjeta de red hasta su código y de regreso, y desde su código hasta el disco. En un sistema predeterminado, cada paso pertenece a otro: el controlador del fabricante de la tarjeta de red, la pila de red del kernel, la biblioteca TLS del proveedor del sistema operativo, el sistema de archivos, el controlador de almacenamiento. Para la mayoría de las empresas eso es lo correcto: son capas maduras, parcheadas por equipos grandes.

Poseerla empieza a rendir cuando la ruta de datos es el producto, cuando es su mayor costo o cuando necesita una corrección más rápido de lo que cualquier proveedor la publicará. Cloudflare es un ejemplo público: construyó **Pingora**, su propio proxy en Rust, para reemplazar el software de proxy anterior frente a su red. El capítulo 14 cuenta esa historia completa.

Cloudflare construyó el proxy, donde se concentran su lógica de negocio y sus costos. No escribió su propio kernel, ni sus controladores, ni su sistema de archivos. Poseyó esa capa estrecha y alquiló las capas amplias que están debajo.

> *Un controlador es un contrato que nunca firmó. Hay que saber quién lo escribió, quién permite que se ejecute y quién publicará la corrección.*

**CÓMO LLEGÓ A SER ASÍ**

Windows Vista sacó del kernel la mayor parte del código de los controladores gráficos con WDDM, porque los controladores de pantalla habían sido una de las principales causas de caídas. Linux nunca ofreció a los controladores una interfaz estable, y en cambio pidió a los fabricantes que se unieran al árbol. La barrera de firma de Windows se describe arriba, y el abandono de las extensiones de kernel por parte de Apple se cuenta en el capítulo 2. Tres sistemas, un problema, tres dueños: Microsoft como guardián, la comunidad del kernel como mantenedora y Apple como proveedor de casi todo.

## Esta capa en la lista de materiales del píxel

| Componente | Quién lo escribe | Quién paga | Quién puede actualizarlo | Plazo de soporte | ¿Expuesto a entradas no confiables? | Su decisión |
|---|---|---|---|---|---|---|
| Controlador de GPU (WDDM, Mesa, Apple) | El fabricante de la GPU, los desarrolladores de Mesa o Apple | El fabricante, con lo que obtiene de la venta de hardware | El fabricante; en Linux, también la distribución | El fabricante deja de dar soporte a las GPU antiguas con su propio calendario | Sí: los shaders y el contenido de páginas web y documentos llegan a él | Alquilar; fijar y probar versiones |
| Controlador y pila de la tarjeta de red | El fabricante de la tarjeta de red y el proveedor del sistema operativo | Los proveedores | El proveedor del sistema operativo o la distribución | La vida de soporte del sistema operativo | Sí: cada paquete que llega de la red | Alquilar |
| Sistema de archivos (NTFS, ext4, APFS) | El proveedor del sistema operativo o la comunidad del kernel | El proveedor del sistema operativo; en Linux, las empresas que contribuyen | El proveedor del sistema operativo o la distribución | La vida de soporte del sistema operativo | Sí, mediante medios extraíbles e imágenes de disco | Alquilar; en Linux, elegir a conciencia |
| Proxy o balanceador de carga frente a su servicio | Un proyecto abierto, un proveedor o usted | Usted, mediante licencia, factura de nube o personal | A quien haya elegido | El que usted defina, si lo construye | Sí: es lo primero que toca internet | Alquilar, financiar o construir, según la escala |

LENTE DE SEGURIDAD

**¿Quién puede cambiarlo?** El código de controladores en modo kernel se ejecuta con el privilegio más alto del sistema operativo, y la mayor parte lo escriben fabricantes de hardware; los controladores en modo usuario (UMDF, extensiones de DriverKit, la mitad en modo usuario de un controlador de GPU) por lo general solo detienen su propio proceso cuando fallan. En Windows, Microsoft controla qué controladores de kernel pueden cargarse. En Linux, la distribución decide qué controladores incluye en su kernel. En macOS, Apple escribe la mayoría de los controladores y aprueba el resto. Su empresa puede elegir el hardware y las versiones, y poco más.

**¿Qué tan rápido se puede corregir?** Tan rápido como el dueño más lento de la cadena. Una falla en un controlador de red en Linux se corrige upstream, luego la toma cada distribución y luego la instala la empresa. Una falla en un controlador de GPU en Windows espera al fabricante de la GPU. Las flotas que fijan versiones de controladores por estabilidad también deben hacerse cargo de mover esa versión fijada.

**¿Qué cambia la IA aquí?** El fuzzing asistido por IA abarata la búsqueda en controladores y en interfaces internas del kernel como io_uring y eBPF, que procesan (parsing) entradas complejas con privilegio total, así que saldrán a la luz más fallas, tanto para los proveedores como para los atacantes. Para una flota que fija versiones de controladores, eso pone el peso en mover la versión fijada: hay que tener una forma probada de llevar un controlador nuevo a cada modelo en días.

IMPLICACIONES PARA EL NEGOCIO

**Seguridad.** Un **operador de una flota de dispositivos** con miles de quioscos tiene más dueños de controladores de los que cree: GPU, pantalla táctil, lector de tarjetas, impresora de recibos, red. La mayoría se ejecuta al menos en parte en el kernel, y cada uno viene de una empresa distinta, con un ritmo de parches distinto. Hay que listarlos, por modelo y versión, antes de que un incidente obligue a hacerlo.

**Costo.** Un **fabricante de software** que entrega una aplicación de escritorio con gráficos intensivos paga la diversidad de controladores de GPU en tickets de soporte y en alternativas silenciosas por software que hacen que la aplicación se sienta lenta en algunas máquinas. Una **empresa regulada** que elige un sistema de archivos en Linux paga después si elige uno sin sumas de verificación para registros cuya integridad debe demostrar. Ambos costos se evitan con una matriz de pruebas y una métrica del renderizador.

**Valor.** Poseer la ruta de datos solo rinde donde es el producto o la factura más grande, como lo fue el proxy para Cloudflare. Para la mayoría de las empresas, lo valioso es más acotado: elegir controladores y sistemas de archivos a propósito, medir la ruta que realmente sigue el hardware y mantener corta la ruta de corrección.

**Antes de la próxima actualización de controladores:** listar cada controlador de nivel kernel de su flota con su dueño y su versión, y alertar cuando una máquina recurra a una ruta por software.

CAPÍTULO CUATRO

# Seguridad, identidad y el píxel de confianza




Cada aprobación en una empresa termina con una persona que mira una pantalla y pulsa un botón. Se libera un pago, se firma un contrato, un administrador concede un privilegio. Toda la cadena supone que la pantalla mostró la verdad. Este capítulo explica cómo decide cada sistema operativo quién puede hacer qué, y luego plantea la pregunta que casi nadie incluye en un registro de riesgos: ¿quién responde por el píxel que la persona realmente vio?

Una **frontera de confianza** es una línea del sistema donde cambian las reglas: de un lado el código puede hacer algo, del otro no. El kernel y el espacio de usuario forman la primera frontera, tratada en el capítulo 2. Este capítulo trata las fronteras construidas encima: entre usuarios, entre aplicaciones, entre un servicio en segundo plano y la persona frente al teclado. Cada sistema las traza de otra manera, y su proveedor controla el trazado.

## Fronteras de confianza en cada sistema

**Windows** asigna a cada proceso un **token de acceso** (access token), un objeto del kernel que indica con qué usuario se ejecuta, a qué grupos pertenece y qué privilegios tiene. Cada archivo, clave del registro y demás objetos llevan una **lista de control de acceso (ACL)** que indica qué usuarios y grupos pueden hacer qué. Cuando un programa abre un archivo, el kernel compara el token con la lista. El **Control de cuentas de usuario (UAC)**, introducido con Windows Vista, da a un administrador dos tokens: uno filtrado para el trabajo diario y uno completo que solo se usa tras una aprobación. De forma predeterminada, la solicitud de aprobación aparece en el **escritorio seguro**, un escritorio aparte al que, según Microsoft, solo pueden acceder procesos de Windows. Los programas comunes no pueden dibujar en él ni enviarle entradas.

**Linux** parte del modelo Unix: un ID de usuario, un ID de grupo y bits de permiso en cada archivo, con el superusuario, root, autorizado para todo. Desde Linux 2.2 el kernel divide el poder de root en **capacidades** (capabilities) separadas, como el derecho a usar un puerto de red bajo o a cargar un módulo del kernel, de modo que un servicio puede tener solo las que necesita. Encima, los **Linux Security Modules** añaden reglas obligatorias que incluso root debe respetar. **SELinux**, publicado por la Agencia de Seguridad Nacional de EE. UU. en diciembre de 2000, etiqueta cada proceso y cada archivo y compara cada acceso con una política central. **AppArmor**, incorporado al kernel principal en Linux 2.6.36 en octubre de 2010, confina los programas por ruta de archivo y es más sencillo de escribir.

**macOS** conserva el modelo Unix por debajo y añade capas propias de Apple. **System Integrity Protection (SIP)**, introducido con OS X El Capitan en 2015, impide incluso al usuario root modificar archivos y carpetas protegidos del sistema. El **App Sandbox** confina una aplicación a los recursos que declara. Esas declaraciones son **entitlements**: pares clave-valor firmados dentro de la firma de código de una app que otorgan derechos concretos. Por último, **Transparency, Consent and Control (TCC)** pregunta al usuario, o a un administrador mediante la gestión de dispositivos, antes de que una app pueda usar la cámara, el micrófono, carpetas protegidas o la grabación de pantalla.

| Frontera | Windows | Linux | macOS |
|---|---|---|---|
| **Identidad de un proceso** | Token de acceso | IDs de usuario y de grupo, más capacidades | ID de usuario, más firma de código y entitlements |
| **Quién puede acceder a un objeto** | ACL en cada objeto | Bits de permiso; política de SELinux o AppArmor | Bits de permiso; perfil de sandbox |
| **Reglas que ni el administrador puede saltar** | Procesos protegidos, firma de controladores | SELinux o AppArmor en modo de cumplimiento (enforcing) | System Integrity Protection |
| **Preguntar a la persona** | UAC en el escritorio seguro | Solicitudes propias de cada escritorio (polkit, portales) | Solicitudes de TCC |
| **Gestor de servicios** | Service Control Manager | systemd en la mayoría de las distribuciones | launchd |
| **Marco de inicio de sesión** | Windows Hello y proveedores de credenciales | PAM | Touch ID, verificado en el Secure Enclave, mediante Authorization Services y LocalAuthentication; PAM para `sudo` y algunos servicios |

## Servicios e identidad

La mayor parte del código de una máquina se ejecuta sin que nadie lo vigile. Un **gestor de servicios** inicia ese código, lo reinicia y decide con qué identidad se ejecuta. En Windows es el **Service Control Manager (SCM)**, que ejecuta cada servicio con una cuenta como LocalSystem o una cuenta virtual dedicada. En la mayoría de las distribuciones Linux es **systemd**, que Lennart Poettering presentó en abril de 2010 y que puede aplicar capacidades, sistemas de archivos privados y otros límites a cada servicio desde un solo archivo de unidad. En macOS es **launchd**, que el propio Poettering citó como modelo para la idea de systemd de iniciar servicios bajo demanda. El gestor de servicios es donde el mínimo privilegio se configura o se olvida.

El inicio de sesión es la otra mitad de la identidad. En Linux y otros sistemas Unix, los **Pluggable Authentication Modules (PAM)**, propuestos por primera vez por ingenieros de Sun en el RFC 86.0 de la Open Software Foundation en octubre de 1995, permiten a los administradores cambiar la forma en que los usuarios demuestran quiénes son sin modificar cada programa. Microsoft anunció **Windows Hello** para Windows 10 en una entrada del blog Windows Experience el 17 de marzo de 2015, con sensores de rostro, iris o huella dactilar. En los Mac, la verificación de **Touch ID** se ejecuta en el **Secure Enclave**, un procesador aparte; Apple afirma que los datos de la huella se cifran y se protegen con una clave disponible solo para el Secure Enclave. En los tres casos el proveedor del sistema operativo controla la pantalla de inicio de sesión, la ruta del sensor y el mensaje que lee la persona.

## El píxel de confianza

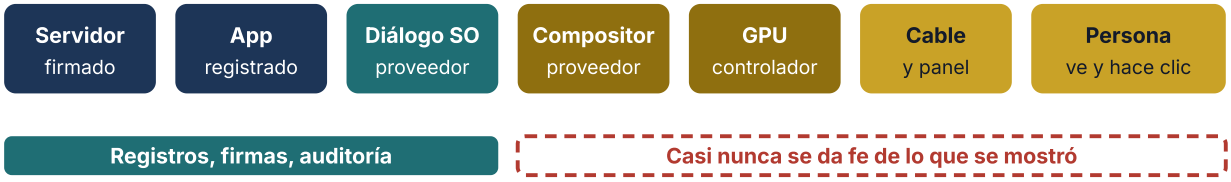
El principio no es nuevo. Quienes diseñan firmas lo llaman **WYSIWYS**, lo que se ve es lo que se firma ("what you see is what you sign"), título de un artículo de 1998 de Peter Landrock y Torben Pedersen. Este libro lo aplica a toda aprobación, no solo a las firmas: **lo que la persona vio debe ser lo que el sistema hizo**. Cada diálogo de consentimiento, confirmación de pago, solicitud de inicio de sesión y firma electrónica depende de ello. El sistema prueba mucho sobre la transacción. Firma la solicitud, registra al usuario, pone fecha y hora al clic. En la mayoría de los sistemas prueba mucho menos sobre la pantalla.

Pero menos no es cero. **Android Protected Confirmation**, añadido en Android 9 en 2018 como función opcional en dispositivos compatibles, cede el control a una interfaz de usuario protegida por hardware que muestra una declaración breve a pantalla completa. Si la persona aprueba, la app firma esa declaración con una clave que no puede firmar nada más. La documentación de Google dice que una respuesta positiva indica "con alta confianza que el usuario ha visto el texto dado, incluso si el framework de Android (incluido el kernel) estuviera comprometido". No conviene apoyarse solo en ella. Requiere hardware dedicado y solo funciona donde el fabricante del dispositivo la admite. En enero de 2024 un cambio en Android declaró obsoleto (deprecated) el código interno que la sostiene, citando su costo para los fabricantes de dispositivos y su baja adopción entre desarrolladores; en mayo de 2024 ese cambio se revirtió como una "obsolescencia parcial involuntaria". A noviembre de 2026 la referencia pública de `ConfirmationPrompt` de Google no lleva ningún aviso de obsolescencia, pero una función con ese historial solo cabe en un diseño como un extra donde el dispositivo la ofrezca. La brecha real es más estrecha y sigue siendo amplia. A noviembre de 2026, los escritorios de Windows, macOS y Linux no ofrecen a las apps de terceros ningún mecanismo general equivalente: el escritorio seguro de Windows protege solicitudes del sistema como UAC, no su pantalla de pagos. Donde no se usa un mecanismo así, los registros prueban la transacción, no lo que vio la persona.

"Píxel de confianza" abarca además cinco amenazas distintas, y cada una necesita un control distinto. Mezclarlas es la forma en que los equipos compran un control para una amenaza y creen que cubre las demás.

FIGURA 4.1

## La ruta del consentimiento y dónde se acaba la evidencia







El clic vuelve con pruebas. La imagen viajó hacia adelante sin ellas.

Superposiciones, capturas, agentes y deepfakes actúan en la mitad derecha.

Un consentimiento deja evidencia sólida en la parte izquierda de esta ruta y, salvo que se use una pantalla de confirmación protegida, poca en la derecha. La mitad derecha pertenece al proveedor del sistema operativo, al proveedor de la GPU y al fabricante de la pantalla.

| Modelo de amenaza | Qué falla | Quién suministra el control | Control |
|---|---|---|---|
| 1. Integridad de la transacción | Los datos firmados no son los que la persona quería | Usted, dentro de las reglas de su regulador | Vinculación dinámica: vincular el monto y el beneficiario a la solicitud mediante la firma |
| 2. Interfaz de confianza | Otro programa dibuja encima de la pantalla de aprobación o la imita | El proveedor del sistema operativo | Superficies protegidas donde existan (escritorio seguro, confirmación protegida, filtrado de superposiciones); para apps de escritorio de terceros, que no tienen ninguna, una confirmación por un segundo canal |
| 3. Identidad y deepfakes | El rostro o la voz en pantalla no es quien parece | Usted, mediante procesos | Devolución de llamada por otro canal; aprobaciones que ningún canal por sí solo puede completar |
| 4. Captura y privacidad | Lo que muestra la pantalla se copia, se guarda o se exporta | El proveedor del sistema operativo lo construye; usted fija la política | Política de captura: display affinity, política de Recall, reglas de exportación |
| 5. Agentes delegados | Un software que actúa con la autoridad del usuario lee y acepta la aprobación | Usted, mediante identidad y permisos | Identidades separadas para agentes, permisos acotados, ningún agente puede aprobar por sí solo una operación con valor económico |

**Modelo de amenaza 1: integridad de la transacción.** La pregunta es si los datos que firma el sistema son los que la persona quería aprobar. En la Unión Europea, las reglas de autenticación reforzada de clientes de la segunda Directiva de Servicios de Pago (PSD2), establecidas en el Reglamento

Delegado (UE) 2018/389 de la Comisión, exigen la **vinculación dinámica**. El artículo 5 exige que "el ordenante sea consciente del importe de la operación de pago y del beneficiario", que el código de autenticación sea específico de ese monto y ese beneficiario, y que cualquier cambio en uno de ellos invalide el código. El artículo 5, apartado 2, va más allá: los proveedores deben adoptar medidas de seguridad que garanticen "la confidencialidad, autenticidad e integridad" del monto y del beneficiario y de "la información mostrada al ordenante en todas las fases de la autenticación". Un malware que muestra un beneficiario y envía otro es, por tanto, una falla de controles que la norma ya exige. Queda la brecha más estrecha: la norma fija el deber, pero ningún mecanismo estándar permite probar después lo que mostró la pantalla. El código de autenticación prueba qué monto y qué beneficiario se autenticaron, no lo que vio la persona. La identidad digital sigue el mismo patrón. El Reglamento (UE) 2024/1183, conocido como eIDAS 2.0, entró en vigor el 20 de mayo de 2024 y crea la **Cartera Europea de Identidad Digital** (European Digital Identity Wallet). La Comisión Europea indica que los Estados miembros deben ofrecer carteras a sus ciudadanos antes de que termine 2026, semanas después de esta edición de noviembre de 2026, así que conviene comprobar el estado de la cartera de cada Estado miembro el día en que se diseñe para ella. Una cartera muestra atributos y solicitudes de firma en la pantalla de un teléfono, y la aprobación de la persona en ella es lo que la parte que se basa en esa aprobación da por bueno. **Control:** vincular el monto, el beneficiario y la identidad a la solicitud mediante la firma, verificar esa vinculación en el servidor y, para acciones de alto valor, volver a mostrar los datos vinculados en un canal que el dispositivo solicitante no controle.

**Modelo de amenaza 2: interfaz de confianza.** La pregunta es si otro programa puede dibujar encima de la aprobación o falsificarla. Si puede, la persona ve una cosa y pulsa otra. En Android este ataque es tan común que Google lo documenta como **tapjacking** y da a los desarrolladores un indicador, `filterTouchesWhenObscured`, para ignorar los toques mientras otra ventana cubre la vista. En Windows la defensa para el consentimiento de administrador es el escritorio seguro, donde otros programas no pueden dibujar. En dispositivos Android compatibles, Protected Confirmation va más allá y muestra el texto de aprobación mediante una interfaz protegida por hardware que un sistema operativo comprometido no puede alterar. El proveedor del sistema operativo suministra cada uno de estos controles. Una aplicación puede pedirlos; no puede construirlos. En los escritorios de Windows, macOS y Linux, una app de terceros no tiene una superficie protegida para su propia pantalla de aprobación. **Control:** usar la superficie protegida que ofrezca la plataforma donde exista; para aprobaciones de terceros en el escritorio, enviar la confirmación final a un segundo canal, como un dispositivo aparte, una aprobación firmada por otro canal en su propia app en un segundo dispositivo o una devolución de llamada, y tratar Protected Confirmation como un extra donde un dispositivo lo admita, no como una dependencia.

**Modelo de amenaza 3: identidad y deepfakes.** La pregunta es si el rostro o la voz en la pantalla pertenecen a quien parece. En enero de 2024 un empleado de finanzas en Hong Kong se unió a una videollamada en la que el director financiero y los demás colegas eran deepfakes, creados, según la policía, a partir de video y audio públicos. El empleado hizo 15 transferencias por un total de 200 millones de dólares de Hong Kong, unos 25,6 millones de dólares estadounidenses. En mayo de 2024 la firma de ingeniería Arup confirmó que era la víctima. El software de llamadas funcionó exactamente

como estaba diseñado. **Control:** devolver la llamada a un número conocido antes de cualquier transferencia por encima de un umbral, y exigir aprobaciones que ningún canal por sí solo pueda completar.

**Modelo de amenaza 4: captura y privacidad.** La pregunta es quién puede copiar, guardar o transmitir lo que muestra la pantalla. Windows ofrece **display affinity**: `SetWindowDisplayAffinity` con `WDA_EXCLUDEFROMCAPTURE`, disponible desde Windows 10 versión 2004, mantiene una ventana fuera de las capturas de pantalla. Microsoft deja claro que no es una garantía de seguridad, ya que nada impide apuntar una cámara a la pantalla. Microsoft anunció **Recall** el 20 de mayo de 2024, con sus primeros equipos Copilot+, una función de Windows que guarda instantáneas periódicas de la pantalla para que el usuario pueda buscar lo que ha visto. Después de las críticas de investigadores de seguridad y del público, Microsoft la retrasó y la modificó. Su entrada del 27 de septiembre de 2024 describe el nuevo diseño: Recall siempre requiere activación expresa, las instantáneas siempre se cifran, los servicios de almacenamiento se ejecutan en un enclave de seguridad basada en virtualización y el acceso requiere Windows Hello. La documentación de gestión de Microsoft indica que Recall está deshabilitado y eliminado de forma predeterminada en los dispositivos gestionados, y que los administradores deciden si los usuarios pueden habilitarlo y, en el Espacio Económico Europeo, si pueden exportar sus instantáneas. El proveedor del sistema operativo puede decidir que lo que muestra una máquina también se guarde; la protección de ese almacén es diseño del proveedor, y la política que lo rodea es suya. El recuadro siguiente trata el límite regulatorio: para algunas empresas, mostrar una pantalla puede ser una exportación. **Control:** fijar la política de captura de forma deliberada (Recall, grabación de pantalla, herramientas de soporte remoto), marcar las ventanas sensibles con display affinity y tratar el uso compartido de pantalla con tecnología controlada como una decisión de cumplimiento de exportaciones.

**Modelo de amenaza 5: agentes que actúan con autoridad delegada.** La pregunta ya no es quién está en la pantalla sino qué está en el teclado. El 22 de octubre de 2024 Anthropic lanzó **computer use** (uso de la computadora por agentes) en beta pública: un modelo que mira capturas de pantalla, mueve el cursor, hace clic y escribe. OpenAI lanzó **Operator**, un agente de navegador, como versión preliminar de investigación el 23 de enero de 2025. Estos agentes actúan con los privilegios del usuario y leen los mismos píxeles que lee una persona, así que un diálogo de consentimiento pensado para un humano puede ser aceptado por software en nombre del humano. Nadie suplanta a nadie: el agente tiene autoridad real y delegada, y un prompt inyectado en una página o un documento puede orientar cómo la usa. **Control:** dar a los agentes identidades propias y permisos estrechamente acotados, registrar sus acciones con esas identidades y mantener los pagos y las firmas fuera del alcance de cualquier agente que actúe solo.

> *El proveedor del sistema operativo o el proyecto de escritorio, no usted, controla el píxel del consentimiento. Es una decisión de poseer o alquilar, la haya tomado alguien a propósito o no.*

Para la mayoría de las empresas, lo correcto sigue siendo alquilar esa capa, porque el escritorio seguro, la pantalla de inicio de sesión y la confirmación protegida los construye el proveedor del sistema operativo, no la app. Alquilar bien significa saber que está alquilado. Conviene incluir en el inventario las pantallas de consentimiento de las que depende el negocio, indicar a cuál de los cinco modelos de amenaza se enfrenta cada una y darle el control que le corresponde.

**RECUADRO PARA EMPRESAS REGULADAS: COMPARTIR PANTALLA PUEDE SER UNA EXPORTACIÓN**

Según las Export Administration Regulations (EAR) de EE. UU., una tecnología puede exportarse sin que nada salga del edificio. 15 CFR 734.13 define una **exportación presunta** (deemed export): entregar tecnología o código fuente controlados a una persona extranjera dentro de Estados Unidos. 15 CFR 734.15(a)(1) dice que la tecnología se "entrega" mediante la "inspección visual u otra inspección por una persona extranjera de artículos que revele 'tecnología' o código fuente sujetos a las EAR". Una pantalla compartida, una sesión de soporte remoto o un escritorio virtual compartido pueden ser esa inspección.

Esto se aplica solo a la tecnología y el código fuente sujetos a las EAR que requieren una licencia para esa persona. No convierte cada pantalla compartida en una exportación. Pero para un proveedor de defensa, un diseñador de chips o un grupo de investigación, el píxel es un cruce de frontera, y la política de pantalla compartida corresponde al equipo de cumplimiento de exportaciones.

**CÓMO LLEGÓ A SER ASÍ**

La NSA entregó SELinux a la comunidad de código abierto en diciembre de 2000, y el debate sobre si aceptarlo dio lugar a Linux Security Modules, un punto de enganche general que permitió conectar al kernel SELinux y, más tarde, AppArmor en 2010. Windows Vista, lanzado al público el 30 de enero de 2007, añadió UAC y el escritorio seguro: los administradores pasaron a trabajar a diario con un token filtrado y a aprobar su propia elevación en una pantalla que otros programas no podían tocar. Apple añadió SIP en 2015, para que ni siquiera root pudiera alterar el sistema operativo. Cada paso movió la línea entre código de confianza y código no confiable. Ninguno da fe de lo que muestra la pantalla.

## Esta capa en la lista de materiales del píxel

| Componente | Quién lo escribe | Quién paga | Quién puede actualizarlo | Plazo de soporte | ¿Expuesto a entradas no confiables? | Su decisión |
|---|---|---|---|---|---|---|
| Solicitudes de consentimiento y elevación (UAC, TCC, polkit) | Proveedor del sistema operativo o proyecto de escritorio | Proveedor del sistema operativo | Proveedor del sistema operativo o distribución | Vida de soporte del sistema operativo | Indirectamente: superposiciones, interfaces falsas y agentes la atacan; no procesa archivos no confiables | Alquilar; confirmar las acciones de alto valor por un segundo canal |
| Inicio de sesión y biometría (Windows Hello, PAM, Touch ID) | Proveedor del sistema operativo y fabricante del sensor | Proveedores del sistema operativo y del hardware | Proveedor del sistema operativo | Vida de soporte del sistema operativo y del dispositivo | En parte: ataques de presentación a los sensores | Alquilar |
| Política de acceso obligatorio (SELinux, AppArmor, perfiles de sandbox) | Distribución o proveedor del sistema operativo; usted para reglas propias | Proveedor, más el tiempo de su personal | Usted, para su propia política | El suyo, para la política propia | Indirectamente: limita lo que puede alcanzar un proceso comprometido | Alquilar el marco; construir su propia política |
| Sus pantallas de aprobación de pagos y firmas | Usted | Usted | Usted | El suyo | Sí: superposiciones, captura, agentes | Construir; vincular el monto y el beneficiario a la solicitud mediante la firma y confirmar en una pantalla protegida o por un segundo canal |

LENTE DE SEGURIDAD

**¿Quién puede cambiarlo?** El proveedor del sistema operativo controla los tokens, las capacidades, SIP, el escritorio seguro y cada solicitud del sistema. Los administradores controlan la política que se ejecuta encima. Los equipos de aplicaciones solo son dueños de sus propias pantallas, y esas pantallas las dibuja el compositor del proveedor.

**¿Qué tan rápido se puede corregir?** Una falla en una solicitud del sistema o en un sandbox espera a la publicación del proveedor del sistema operativo. Un diseño de aprobación débil en su propio producto se puede corregir en su próxima versión, y por eso la vinculación del monto, el beneficiario y la identidad a la solicitud mediante la firma es la parte que conviene poseer.

**¿Qué cambia la IA aquí?** Los deepfakes abaratan la falsificación de un rostro o una voz en una videollamada, y los agentes de computer use pueden leer y aceptar una pantalla de consentimiento con los privilegios del usuario. Ambos rompen el supuesto detrás de los modelos de amenaza 3 y 5, de que un humano conocido miró el píxel, así que las aprobaciones de operaciones con valor económico necesitan un paso que ni un rostro falsificado ni un agente puedan completar por sí solos.

IMPLICACIONES PARA EL NEGOCIO

**Seguridad.** Una **empresa regulada** que libera pagos basándose en una videollamada o una aprobación en pantalla confía en una capa que no posee. Conviene exigir la devolución de llamada a un número conocido para cualquier transferencia por encima de un umbral y asegurarse de que los agentes que se ejecutan en los equipos del personal no puedan aprobar pagos ni firmas por sí solos.

**Costo.** Un **fabricante de software** que construye su propio flujo de firma o aprobación paga por diseñarlo dos veces: una para las personas y otra para los agentes que lo operarán. Un **operador de una flota de dispositivos** con quioscos de autoservicio debe decidir si se permiten la captura de pantalla, las superposiciones y las herramientas de soporte remoto en las máquinas que aceptan pagos, y hacerlo cumplir mediante políticas, no por costumbre.

**Valor.** El valor es una confianza que resiste una disputa. Una empresa que puede demostrar qué monto y qué beneficiario se autenticaron, y que la persona los confirmó por un segundo canal, tiene una evidencia que una captura de pantalla no puede dar. Aun así no puede probar lo que mostró la pantalla salvo que la confirmación se haya ejecutado en una pantalla protegida, así que no conviene prometer más que eso en sus términos y condiciones.

**Primera acción:** listar cada aprobación de la que depende el negocio y, para cada una, anotar quién controla la pantalla en la que aparece y a cuál de las cinco amenazas se enfrenta.

CAPÍTULO CINCO

# Entornos de ejecución y lenguajes




Sus desarrolladores no escriben para el kernel. Escriben para un entorno de ejecución (runtime): una biblioteca, una máquina virtual o un intérprete que se sitúa entre su código y el sistema operativo. Ese runtime decide cómo se maneja la memoria, con qué rapidez llega una corrección de seguridad a sus clientes y quién está autorizado a publicarla. La mayoría de las empresas eligieron sus runtimes hace años, proyecto a proyecto, y nunca dejaron la decisión por escrito.

## Qué es realmente un runtime

Un **runtime** es el código que debe estar presente en la máquina para que su programa se ejecute, aunque su equipo no lo haya escrito. Inicia el programa, reparte la memoria, informa de los errores, habla con el sistema operativo en nombre del programa y a menudo limpia lo que este deja.

Los runtimes tienen tres tamaños aproximados. El más pequeño es una **biblioteca del lenguaje**, como el runtime de C: código de arranque, asignación de memoria, funciones de archivos y de cadenas. En el medio está un **runtime gestionado**, como la Java Virtual Machine o el Common Language Runtime (CLR) de .NET. Añade un **recolector de basura**, una parte que encuentra la memoria que el programa ya no usa y la libera, y normalmente un **compilador just-in-time**, que convierte bytecode portable en código máquina mientras el programa se ejecuta. El más grande es una **plataforma completa** que lleva un motor de lenguaje, una biblioteca estándar y a menudo un motor de navegador, como Node.js o Electron.

El tamaño importa menos que quién lo suministra. Todo runtime tiene un proveedor o una comunidad que lo escribe, un plazo de soporte que se acaba y un canal de actualización. Un defecto en el runtime es un defecto en su producto, y solo se puede corregir tan rápido como ese canal lo permita.

## Las plataformas nativas: tres sistemas operativos, tres contratos

Cada sistema operativo ofrece un contrato de runtime nativo, el conjunto de bibliotecas con las que una aplicación puede contar.

**Windows** ofrece Win32, la interfaz en C con el sistema que tiene décadas de antigüedad, y el runtime de C debajo. Desde Windows 10, Microsoft distribuye el **Universal C Runtime** (UCRT) como parte del sistema operativo. La documentación de Microsoft indica que en Windows 10 y 11 siempre se usa la

copia del sistema, incluso cuando una aplicación lleva su propia copia local. Encima está .NET: el antiguo .NET Framework, que es un componente de Windows, y el .NET moderno multiplataforma, que se publica con su propio calendario.

**Linux** ofrece las llamadas al sistema del kernel y, casi siempre, la **GNU C Library** (glibc), que implementa el estándar de C y POSIX. **POSIX** es el estándar de interfaz portable de sistemas operativos; su primera edición, IEEE 1003.1, se publicó en 1988. En Linux, la distribución controla glibc: Red Hat, Canonical, Debian o su propio equipo de compilación decide qué versión recibe y cuándo se parchea.

**macOS** ofrece libSystem, la capa de C y POSIX de Apple, y encima los frameworks Cocoa y los runtimes de Objective-C y Swift. Apple anunció Swift el 2 de junio de 2014. Con Swift 5, en 2019, el lenguaje alcanzó la **estabilidad de ABI**, lo que significa que el código compilado con distintas versiones del compilador puede funcionar en conjunto, y el runtime de Swift empezó a incluirse en el sistema operativo a partir de macOS 10.14.4. Desde entonces lo actualiza Apple, no la app.

| Pregunta | Windows | Linux | macOS |
|---|---|---|---|
| **Runtime de C base** | Universal CRT, parte de Windows 10 y posteriores | glibc (musl en algunas distribuciones pequeñas) | libSystem |
| **Framework nativo de aplicaciones** | Win32, con .NET encima | Ninguno del kernel; GTK o Qt según la app | Cocoa, con Swift y Objective-C |
| **Runtime gestionado del proveedor** | .NET CLR | Ninguno por defecto | Ninguno con recolector de basura; Swift usa conteo de referencias |
| **Quién parchea el runtime base** | Microsoft, mediante Windows Update | El proveedor de la distribución | Apple, mediante actualizaciones del sistema operativo |
| **¿Puede una app reemplazarlo?** | No el UCRT en Windows 10 y 11 | Sí, con paquetes o contenedores | Rara vez, y solo para destinos antiguos de Swift |

## Runtimes que viajan: la JVM, Node.js y Python

Tres runtimes prometen el mismo código en todos los sistemas operativos. Cada uno cumple la promesa llevando consigo su propio mundo.

La **Java Virtual Machine** ejecuta bytecode en cualquier máquina que tenga una JVM. Hoy muchos proveedores construyen su propia JVM a partir del código abierto de OpenJDK, de modo que una empresa elige un proveedor de JVM además de una versión de Java. **Node.js** envuelve un motor de JavaScript con acceso al sistema; el proyecto indica que sus líneas de soporte a largo plazo (LTS) suelen recibir correcciones críticas durante un total de 30 meses. **Python** es un intérprete con un enorme ecosistema de bibliotecas; según su política de versiones (PEP 602), cada versión tiene soporte durante unos cinco años.

Frente a esos plazos está .NET, donde, a noviembre de 2026, la política de soporte de Microsoft da a las versiones LTS tres años de soporte gratuito y a las versiones de soporte estándar (STS) dos años, es decir, 24 meses. Un equipo que usa Java, Node.js, Python y .NET mantiene cuatro calendarios de soporte, cuatro fuentes de avisos de seguridad y cuatro proyectos de actualización que nunca coinciden.

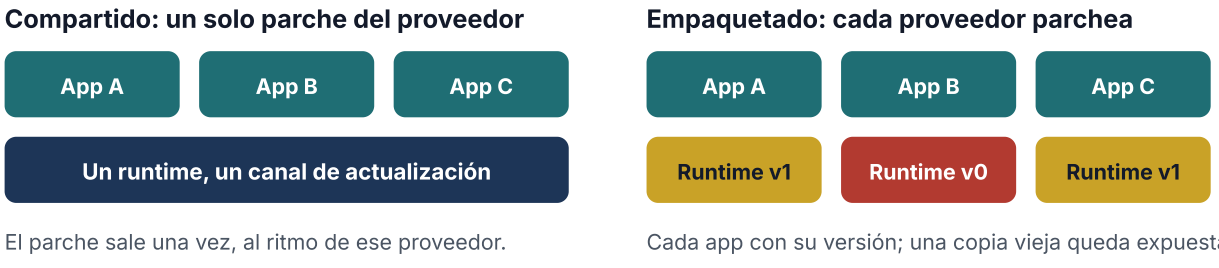
### Por qué cada app trae ahora su propio mundo

Empaquetar resuelve un problema real. Un equipo que distribuye su runtime dentro de la app sabe exactamente qué versión de ese runtime usan sus clientes, y una actualización del sistema no puede cambiarla. Empaquetar fija la dependencia, no la plataforma: las interfaces del sistema operativo, los controladores y la política de seguridad que están debajo del paquete siguen cambiando con cada actualización del sistema. Electron lo dice en su propia documentación: al integrar Chromium y Node.js en su binario, permite que una sola base de código JavaScript se ejecute en Windows, macOS y Linux. Las apps .NET autocontenidas, las apps Java con una JVM privada y las herramientas Python con un intérprete congelado siguen la misma idea.

El precio es la duplicación. Un portátil con cinco apps Electron lleva cinco motores de navegador y cinco runtimes de Node.js, cada uno en su carpeta, cada uno actualizado por su proveedor, cada uno con su calendario. Cuando se encuentra una falla en ese código compartido, el sistema operativo no puede parchearlo por usted. Cada proveedor de apps debe publicar una nueva compilación y cada cliente debe instalarla.

FIGURA 5.1

**Un runtime compartido o un runtime por app**




**Plazo de parche = proveedor más lento + instalación más lenta del cliente**

Con un runtime compartido es un proveedor. Con runtimes empaquetados, todos los de la máquina.

El rojo marca la copia cuyo proveedor aún no ha publicado la corrección.

Ningún modelo gana. Los runtimes compartidos centralizan los parches: una corrección llega a todas las apps, pero la rapidez depende del ritmo de ese único proveedor, y un cambio puede romper apps. Los runtimes empaquetados fijan la versión del runtime pero multiplican los proveedores de los que se depende.

**CÓMO LLEGÓ A SER ASÍ**

Los primeros runtimes eran pequeños. Un programa en C llevaba una rutina de arranque y una biblioteca de funciones, y el sistema operativo hacía el resto. POSIX, publicado por primera vez en 1988, intentó que esa biblioteca fuera la misma en todos los Unix. Luego Sun lanzó Java 1.0 el 23 de enero de 1996 con una promesa audaz: compilar una vez y ejecutar en cualquier máquina que tenga una máquina virtual. Microsoft respondió el 13 de febrero de 2002, cuando lanzó .NET Framework y Visual Studio .NET. El runtime gestionado llevó la recolección de basura y una memoria más segura al código empresarial de todos los días. Veinte años después, los gobiernos empezaron a preguntar por qué tanta parte del sistema bajo esos runtimes seguía escrita en C y C++.

### La seguridad de memoria se vuelve política

Un **error de seguridad de memoria** ocurre cuando un programa lee o escribe memoria que no debería tocar: más allá del final de un búfer, después de que la memoria se liberó o mediante un puntero del tipo equivocado. C y C++ permiten que estos errores compilen. Lenguajes como Rust, Go, Java, C#, Swift y Python están diseñados para impedir la mayoría de ellos.

Dos cifras de proveedores llevaron este tema al consejo de administración. En julio de 2019 el Microsoft Security Response Center escribió que alrededor del 70 % de las vulnerabilidades a las que Microsoft asigna un CVE cada año son problemas de seguridad de memoria. En septiembre de 2024 Google informó que los problemas de seguridad de memoria bajaron del 76 % de las vulnerabilidades de Android en 2019 al 24 % en 2024, después de que Android empezara a escribir código nuevo en lenguajes con seguridad de memoria. El argumento de Google era que la mayor parte del antiguo código C y C++ seguía ahí; el cambio vino de escribir el código nuevo de otra manera.

Los gobiernos lo convirtieron en orientación. Tres documentos marcan el camino:

- **Noviembre de 2022:** la Agencia de Seguridad Nacional de EE. UU. (NSA) publicó una hoja de información de ciberseguridad, "Software Memory Safety", que pedía un cambio estratégico hacia lenguajes con seguridad de memoria.
- **6 de diciembre de 2023:** CISA, la agencia de ciberseguridad de EE. UU., junto con la NSA, el FBI y agencias de Australia, Canadá, el Reino Unido y Nueva Zelanda, publicó "The Case for Memory Safe Roadmaps", que pide a los fabricantes de software publicar su plan.
- **24 de junio de 2025:** CISA y la NSA publicaron "Memory Safe Languages: Reducing Vulnerabilities in Modern Software Development".

Ninguno de estos documentos es una ley que prohíba C. Son orientaciones. Pero algunos compradores regulados y sus auditores ya piden a sus proveedores de software una hoja de ruta de seguridad de memoria, y "nunca lo pensamos" es una respuesta débil.

> *La IA abarata escribir y portar código. No abarata responder por él.*

### Rust y el costo de migrar

Rust es el lenguaje que más se menciona en estas discusiones porque ofrece seguridad de memoria sin recolector de basura y sin máquina virtual. Su compilador comprueba quién es dueño de cada porción de memoria, y cuándo puede liberarse, antes de que el programa se ejecute. Eso permite usarlo en kernels, controladores y otros lugares donde un recolector de basura no cabría.

Rust sigue teniendo un runtime, en el sentido llano de este capítulo. Su biblioteca estándar contiene un módulo de servicios de runtime que, en palabras de su propio código fuente, incluye "el heap global" y "el soporte de desenrollado y backtrace", y prepara el programa antes de que empiece su función `main`. Es pequeño y le toca a usted distribuirlo, lo que significa que también le toca actualizarlo. El código Rust además llama a la biblioteca de C de la plataforma en la mayoría de los sistemas, así que glibc, el UCRT y libSystem siguen en la lista.

Migrar el código existente es la parte difícil. El 31 de julio de 2024 DARPA, la agencia de investigación de defensa de EE. UU., anunció **TRACTOR** (Translating All C to Rust), un programa para automatizar la traducción de código C heredado a Rust seguro e idiomático. DARPA no exigió IA, aunque el director del programa dijo que los grandes modelos de lenguaje casi con seguridad formarán parte de las respuestas. MIT Lincoln Laboratory dirige las pruebas y publica los bancos de pruebas para los equipos. Es un programa de investigación. A noviembre de 2026 demuestra que el problema merece financiación; no demuestra que la migración sea barata.

> **EJEMPLO**
>
> Un operador de una flota de dispositivos tiene un quiosco de pagos cuya biblioteca del lector de tarjetas está escrita en C. Un asistente de IA puede proponer una versión en Rust en días. Alguien todavía tiene que demostrar que se comporta igual en cada modelo de lector, volver a pasar la certificación de pagos y estar de guardia cuando falle a las 2 de la madrugada en una tienda.

## Esta capa en la lista de materiales del píxel

| Componente | Quién lo escribe | Quién paga | Quién puede actualizarlo | Plazo de soporte | ¿Expuesto a entradas no confiables? | Su decisión |
|---|---|---|---|---|---|---|
| `glibc` o el UCRT | Proyecto GNU; Microsoft | Proveedores de distribuciones; Microsoft | El proveedor del sistema operativo o de la distribución | La versión del sistema operativo | Sí, a través de cada analizador que lo llama | Alquilar |
| Runtime de .NET | Microsoft y la comunidad .NET | Microsoft | Microsoft, o usted si es autocontenido | 3 años (LTS), 2 años (STS), a noviembre de 2026 | Sí, en apps web y de escritorio | Alquilar |
| Node.js dentro de apps Electron | Proyectos Node.js y Electron | Patrocinadores y miembros de la fundación | Cada proveedor de apps que lo empaqueta | El fin de vida de la versión mayor de Electron (se mantienen las tres últimas), luego la publicación del proveedor de la app; no el plazo LTS de Node.js | Sí, todo lo que la app carga | Alquilar, y rastrear cada copia |
| Intérprete de Python en herramientas | Desarrolladores principales de Python | Python Software Foundation y patrocinadores | Quien lo congeló dentro de la herramienta | Unos 5 años por versión | A menudo, mediante analizadores de archivos y datos | Alquilar o financiar |
| Sus módulos centrales en C o C++ | Su equipo | Usted | Usted | Mientras los distribuya | Depende de lo que analicen | Construir, con una hoja de ruta de seguridad de memoria |

**LENTE DE SEGURIDAD**

**¿Quién puede cambiarlo?** Para el runtime base, el proveedor del sistema operativo. Para los runtimes empaquetados, cada proveedor de apps que distribuye uno. Un equipo rara vez ve la lista completa hasta que escanea sus máquinas en busca de copias.

**¿Qué tan rápido se puede corregir?** Tan rápido como el proveedor más lento de la copia más lenta. Un runtime compartido se parchea una vez; uno empaquetado se parchea app por app, y la compilación más antigua en uso fija su exposición real.

**¿Qué cambia la IA aquí?** Las herramientas de IA abaratan encontrar errores en los runtimes y portar código a lenguajes con seguridad de memoria. No cambian quién firma la publicación, quién carga con el plazo de soporte ni quién responde cuando falla un módulo traducido.

**IMPLICACIONES PARA EL NEGOCIO**

**Seguridad.** Una empresa regulada debería poder listar cada runtime de sus máquinas, su dueño y su fecha de fin de soporte, y mostrar una hoja de ruta de seguridad de memoria para el código que escribe en C o C++. Los auditores tienen orientaciones gubernamentales a las que remitirse; conviene tener la respuesta preparada.

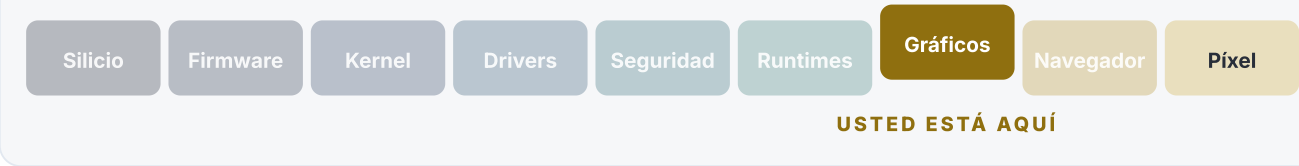
**Costo.** Un fabricante de software que usa Java, Node.js, Python y .NET paga cuatro calendarios de soporte y cuatro proyectos de actualización al año. Elegir uno o dos runtimes por línea de producto es una decisión de costo, no una cuestión de gusto.

**Valor.** Un operador de una flota de dispositivos gana más con una lista corta de runtimes compartidos y parcheados por el proveedor: un solo canal de actualización que probar, una sola corrección que enviar a cada quiosco. Solo conviene empaquetar un runtime donde controlar la versión exacta compense el proveedor adicional.

**Esta semana:** anotar cada runtime que distribuye, quién lo suministra y su fecha de fin, y tratar cada copia empaquetada como un producto que ahora mantiene la empresa.

CAPÍTULO SEIS

# Gráficos, texto y el píxel




Cada fotograma que ve su cliente lo ensambla un compositor que no le pertenece, lo dibuja un controlador de GPU cuyo código fuente normalmente no se puede leer (las pilas abiertas como Mesa y muchos controladores DRM del kernel de Linux son la excepción) y un motor de texto que mantiene un puñado de personas se encarga de la tipografía. Esta capa decide si su producto se ve bien, si se puede vender en un país determinado y si una sola imagen o fuente manipulada puede tomar el control de la máquina. También es la capa que la mayoría de los equipos nunca incluye en un registro de riesgos.

## Tres pipelines, un fotograma

El prólogo siguió una pulsación de tecla hasta la pantalla. Este capítulo abre el último tramo de esa carrera: cómo una ventana terminada se convierte en luz. En los tres sistemas el recorrido habitual a través del compositor tiene la misma forma. La app dibuja en su propio búfer fuera de pantalla. Un **compositor**, un programa del sistema dueño de toda la pantalla, combina el búfer de cada ventana en una sola imagen de escritorio. Un **controlador gráfico** entrega esa imagen a la GPU y al hardware de pantalla, que la muestran en el siguiente ciclo de refresco de la pantalla. Ese es el caso habitual, no el único: una app a pantalla completa o de un tamaño adecuado puede saltarse la composición. La documentación del modelo flip de DXGI de Microsoft describe "Direct Flip" e "Independent Flip", donde los fotogramas de la aplicación llegan a la pantalla sin composición del escritorio, y las superposiciones multiplano, donde el hardware de pantalla muestra el búfer de una app en su propio plano. Linux KMS y macOS tienen rutas comparables de salida directa a pantalla (direct scanout).

**Windows.** Las apps dibujan con Direct3D, Direct2D o interfaces más antiguas. La documentación de Microsoft dice que, con la composición del escritorio, las ventanas "ya no dibujan directamente en la pantalla"; su dibujo se redirige a superficies fuera de pantalla en la memoria de video, que el Desktop Window Manager (DWM) convierte en una imagen de escritorio. La composición llegó con Windows Vista, y desde Windows 8 DWM está siempre activo y ni los usuarios ni las apps pueden desactivarlo. Debajo está el **Windows Display Driver Model** (WDDM), también introducido con Vista, que divide cada controlador de GPU en una parte de modo usuario y otra de modo kernel y permite a Windows reiniciar un controlador de pantalla colgado sin reiniciar el equipo.

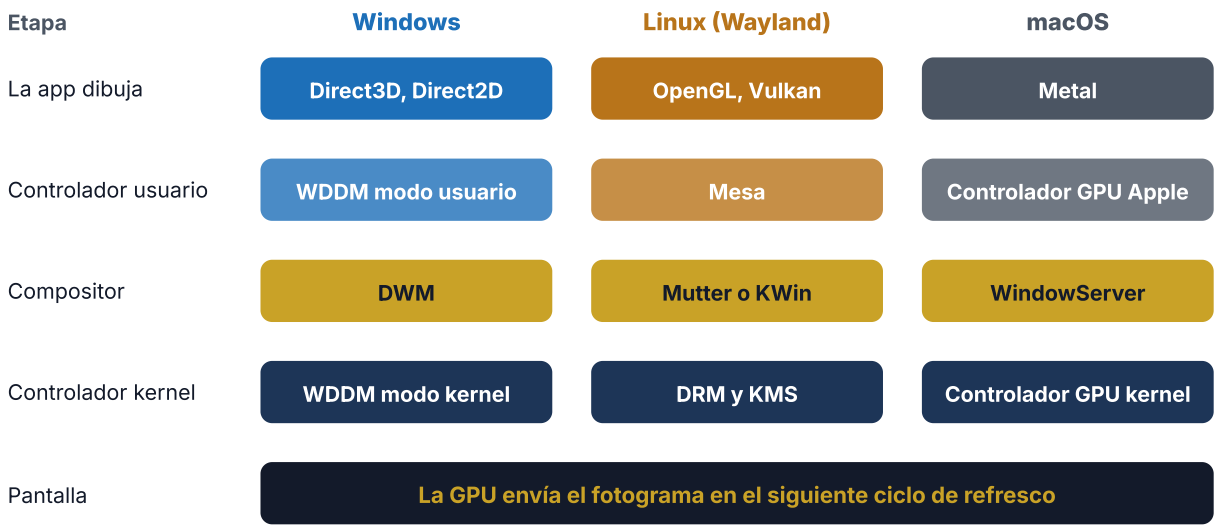
**Linux.** Las apps dibujan con OpenGL o Vulkan, normalmente a través de **Mesa**, la biblioteca gráfica de código abierto en espacio de usuario. El **Direct Rendering Manager** (DRM) del kernel gestiona la memoria y los trabajos de la GPU, y **Kernel Mode Setting** (KMS) fija el modo de pantalla y coloca en

ella los fotogramas terminados. En un escritorio moderno el compositor también es el servidor de pantalla: bajo Wayland, Mutter de GNOME o KWin de KDE reciben el búfer de cada app y construyen el fotograma.

**macOS.** Las apps dibujan con Metal, la interfaz gráfica de Apple anunciada con iOS 8 el 2 de junio de 2014, o mediante frameworks de más alto nivel construidos sobre ella. **Core Animation** describe cada ventana como un árbol de capas, y **WindowServer**, el compositor del sistema, combina esas capas en el fotograma que muestra la pantalla.

FIGURA 6.1

### El pipeline gráfico en cada sistema




Dorado: el compositor, el único programa que ve cada ventana. Siempre del proveedor del SO o del escritorio.

La misma forma, distintos proveedores. En macOS con Apple silicon, Apple suministra todas las cajas salvo su app. En Windows, Microsoft suministra las interfaces gráficas, DWM y el modelo de controladores, pero el fabricante de la GPU escribe las dos mitades del controlador, como mostró el capítulo tres. En Linux las cajas pertenecen a Mesa, al kernel, al proyecto de escritorio y al fabricante de la GPU.

Todo debe caber en un fotograma. A 60 Hz un fotograma dura 16,7 milisegundos; a 120 Hz dura 8,3. Si una app no llega a tiempo, el compositor reutiliza la última superficie terminada de esa app, así que su ventana muestra un fotograma viejo mientras el resto de la pantalla puede seguir actualizándose. Si quien no llega es el compositor o la GPU, la pantalla puede seguir mostrando el fotograma compuesto anterior, salvo que una ventana tenga su propia ruta de salida directa a pantalla o de superposición. Por eso los compositores y los controladores se ajustan con tanto cuidado, y por eso un cambio de renderizado "pequeño" de un proveedor puede costarle a una app de quiosco su desplazamiento fluido.

| Parte | Windows | Linux | macOS |
|---|---|---|---|
| **Interfaz gráfica** | Direct3D 11 y 12 | OpenGL, Vulkan (Mesa) | Metal |
| **Compositor** | DWM, siempre activo desde Windows 8 | Mutter, KWin y otros | WindowServer con Core Animation |
| **Modelo de controladores** | WDDM, desde Vista | DRM y KMS en el kernel | Controladores propios de Apple |
| **Motor de texto** | DirectWrite | HarfBuzz da forma al texto, FreeType rasteriza | Core Text da forma al texto y lo maqueta; Core Graphics rasteriza |
| **Quién publica las correcciones** | Microsoft y el fabricante de la GPU | Kernel, Mesa, proyecto de escritorio, distribución | Apple |

CÓMO LLEGÓ A SER ASÍ

El MIT publicó la versión 11 del X Window System el 15 de septiembre de 1987. X11 se construyó para un mundo de terminales en red: un servidor de pantalla dibujaba en nombre de programas que podían ejecutarse en otra máquina. Décadas después casi todo el dibujo ocurría en la GPU, y el servidor X quedaba en medio sin hacer mucho más que copiar. Wayland tomó el otro camino: el compositor es el servidor de pantalla y las apps le entregan búferes terminados. Wayland 1.0 se anunció el 22 de octubre de 2012. Fedora 25 hizo de Wayland su sesión de escritorio predeterminada en noviembre de 2016, y GNOME 49 desactivó por defecto su sesión X11 dedicada en septiembre de 2025, mientras las apps X11 antiguas siguen funcionando mediante XWayland.

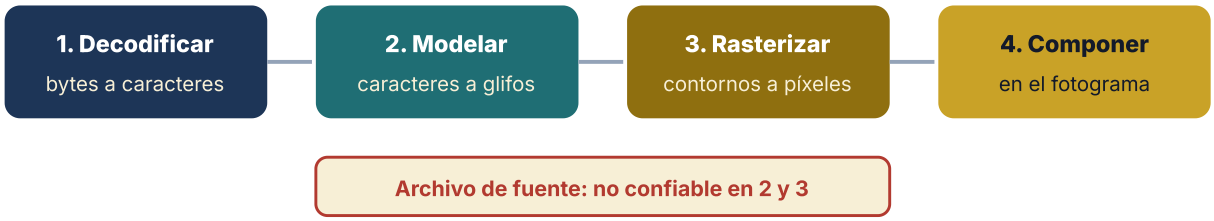
## El texto son cuatro problemas, no uno

Los equipos dicen "fuentes" cuando se refieren a cuatro cosas distintas. Separarlas es el primer paso para gestionar el riesgo.

- La **codificación** convierte bytes en caracteres. Unicode, normalmente almacenado como UTF-8, es la respuesta común; los estándares nacionales siguen importando.
- El **modelado de texto (shaping)** convierte caracteres en los glifos correctos, en el orden y la posición correctos: unir las letras árabes, reordenar las vocales índicas, colocar los diacríticos. HarfBuzz es el motor abierto de modelado de texto que usan muchos navegadores y escritorios Linux.
- Las **fuentes** son los archivos que contienen los contornos de los glifos y las reglas que sigue el modelado de texto. Un archivo de fuente es un pequeño programa hecho de tablas, y el motor confía en que esté bien formado.
- La **rasterización** convierte los contornos de los glifos en píxeles a un tamaño dado. FreeType lo hace en Linux y Android. En Windows, DirectWrite hace tanto el modelado de texto como la rasterización. En macOS, Core Text da forma al texto, lo maqueta y entrega los glifos y sus posiciones a Core Graphics (Quartz), que los dibuja.

FIGURA 6.2

### La ruta del texto: de los bytes a los píxeles encendidos




Archivo de fuente: no confiable en 2 y 3

Windows: DirectWrite hace 2 y 3. Linux: HarfBuzz hace 2, FreeType hace 3.
macOS: Core Text hace 2, Core Graphics (Quartz) hace 3.
El compositor de la etapa 4 es el mismo de la figura 6.1.

Cuatro etapas, a menudo cuatro dueños. Una fuente web o una fuente incrustada en un documento llega a las etapas 2 y 3 directamente desde la red.

## El texto es acceso al mercado

El texto decide dónde se puede vender. El estándar chino de juegos de caracteres **GB 18030-2022** pasó a ser obligatorio el 1 de agosto de 2023. Define tres niveles de implementación, cada uno con más caracteres, y cambió la forma en que algunos caracteres se asignan a Unicode respecto a la edición de 2005. Todo producto dentro del alcance del estándar debe cumplir el primer nivel; que se aplique un nivel superior depende de la categoría del producto. Un fabricante de software que vende en China necesita saber en qué categoría entra su producto y qué nivel cumple su pila de texto, porque un estándar obligatorio se verifica, no se presupone.

Myanmar muestra la misma lección desde el otro lado. Durante años la mayoría de los teléfonos del país usaron Zawgyi, una fuente que reutilizaba puntos de código Unicode para letras distintas. El gobierno fijó el 1 de octubre de 2019 como el día en que la comunicación electrónica oficial pasó a Unicode. El texto escrito en una codificación aparece como letras rotas en la otra, así que cada app tuvo que detectar, convertir o elegir.

El texto también es una superficie de ataque. En marzo de 2025 Meta divulgó **CVE-2025-27363**, una escritura fuera de límites en FreeType 2.13.0 y anteriores, que se activa al procesar archivos de fuentes variables, y señaló que pudo haber sido explotada en ataques reales. CISA la añadió a su catálogo Known Exploited Vulnerabilities el 6 de mayo de 2025. Las versiones más recientes de FreeType no estaban afectadas. El riesgo vivía en cada producto que aún llevaba la 2.13.0 o una copia más antigua.

> *Un SBOM lista los paquetes que se distribuyen. La pregunta que importa es a qué decodificadores pueden llegar los bytes no confiables.*

## Exposición de los decodificadores (reachability)

Un **decodificador** es código que convierte un formato de archivo, como una imagen, una fuente o un video, en píxeles o glifos. Los decodificadores procesan (parsing) datos que llegan de desconocidos, así que es ahí donde apuntan los atacantes.

La lección de la falla explotada en 2023 en **libwebp**, la biblioteca de imágenes WebP (capítulos siete y once), no es que un único decodificador esté en todas las rutas. No lo está. La lección es que una lista de materiales de software (SBOM), la lista de paquetes de un producto, indica qué está instalado. No indica a cuáles de esos decodificadores pueden llegar los bytes no confiables: un adjunto de chat, una imagen web, la miniatura de un documento. Ese mapa de exposición es lo que indica la urgencia de una corrección. Un proveedor publica su respuesta como una declaración **VEX** (Vulnerability Exploitability eXchange), una afirmación legible por máquina de "afectado" o "no afectado" para cada vulnerabilidad (capítulo 8).

## Las API se pueden retirar

Alquilar una interfaz gráfica significa alquilar su futuro. En las notas de la versión de macOS Mojave 10.14, en 2018, Apple indicó que los frameworks OpenGL y OpenCL "están declarados obsoletos y se mantienen por motivos de compatibilidad", y pidió a los desarrolladores pasar a Metal. La declaración de obsolescencia marcó una dirección, no una fecha de eliminación. Aun así, una empresa cuyo producto de ingeniería o de imagen médica se basaba en OpenGL supo que el futuro de su capa de renderizado en el Mac lo decidía ahora otro, aunque el código siguiera funcionando.

**RECUADRO: EL COLOR EN INTERFACES REGULADAS Y DE ALARMA**

Las pantallas de alto rango dinámico (HDR) y de gama amplia rompen un viejo supuesto: que un valor de color siempre se ve igual. La Recomendación UIT-R BT.2100, aprobada por primera vez en julio de 2016, define las señales de televisión HDR con dos métodos de transferencia, PQ y HLG. El mismo rojo puede mostrarse de forma distinta según la pantalla, la gestión de color del sistema operativo y el modo HDR. Una empresa regulada con pantallas de alarma o imágenes clínicas debería probar los colores críticos en el hardware real de su flota y no dar por hecho que el valor del píxel es su apariencia.

**RECUADRO: RENDERIZADORES PROPIOS Y ACCESIBILIDAD**

Algunos productos lo dibujan todo por su cuenta, en un canvas HTML o directamente en la GPU, por velocidad y coherencia. Los píxeles de un canvas no tienen significado para un lector de pantalla, así que el producto debe reconstruir su capa de accesibilidad: una descripción paralela de títulos, texto y controles. Google Docs empezó a pasar del renderizado HTML al renderizado basado en canvas en mayo de 2021 e indicó que las tecnologías de asistencia compatibles, como los lectores de pantalla, no se verían afectadas. Cumplir esa promesa es un trabajo de ingeniería continuo. La Ley Europea de Accesibilidad (European Accessibility Act) se aplica desde el 28 de junio de 2025, así que poseer el renderizador significa poseer ese trabajo y su exposición legal.

## Esta capa en la lista de materiales del píxel

| Componente | Quién lo escribe | Quién paga | Quién puede actualizarlo | Plazo de soporte | ¿Expuesto a entradas no confiables? | Su decisión |
|---|---|---|---|---|---|---|
| Compositor del sistema (DWM, Mutter o KWin, WindowServer) | Proveedor del sistema operativo o proyecto de escritorio | Proveedor del sistema operativo; proveedores de distribuciones | El proveedor del sistema operativo o de la distribución | La versión del sistema operativo | Indirectamente, a través de cada ventana | Alquilar |
| Controlador de GPU (controlador WDDM, Mesa y DRM, controlador de Apple) | Fabricante de la GPU, comunidad Mesa, Apple | Fabricantes de GPU y proveedores de sistemas operativos | Fabricante de la GPU o proveedor del sistema operativo | La política de controladores del proveedor | Sí, mediante shaders y contenido WebGPU o WebGL | Alquilar, y fijar versiones probadas en las flotas |
| Modelado y rasterización de texto (HarfBuzz, FreeType) | Pequeños equipos abiertos | Sobre todo patrocinadores | Cada producto que distribuye una copia | Versiones del proyecto; la antigüedad de su copia | Sí, fuentes web y de documentos | Alquilar o financiar |
| Decodificadores de imágenes (libwebp, libpng y otros) | Proyectos abiertos, algunos liderados por un proveedor | Ese proveedor o patrocinadores | Cada producto que los empaqueta | Versiones del proyecto | Sí, cualquier imagen que abra un usuario | Alquilar; documentar la exposición y publicar VEX |
| Su propio renderizador en canvas o GPU | Su equipo | Usted | Usted | Mientras lo distribuya | Sí, todo lo que dibuja | Construir solo con una capa de accesibilidad |

LENTE DE SEGURIDAD

**¿Quién puede cambiarlo?** El proveedor del sistema operativo controla el compositor, el fabricante de la GPU o el proveedor del sistema operativo controla el controlador, y cada app que empaqueta una biblioteca de texto o de imágenes controla su propia copia. Los mantenedores de la biblioteca escriben la corrección; instalarla en los equipos sigue siendo responsabilidad de cada empresa.

**¿Qué tan rápido se puede corregir?** Para los componentes del sistema, tan rápido como la actualización del sistema operativo llegue a sus máquinas. Para los decodificadores y motores de fuentes empaquetados, tan rápido como el proveedor más lento cuya app lleve una copia antigua.

**¿Qué cambia la IA aquí?** Los decodificadores de imágenes y los motores de fuentes son pequeños analizadores de archivos no confiables, el tipo de código que OSS-Fuzz de Google sondea ahora con objetivos de fuzzing escritos por IA (en noviembre de 2024 informó de 26 vulnerabilidades encontradas así), así que cabe esperar más avisos contra las copias que se empaquetan. Los agentes que leen la pantalla también convierten el píxel renderizado en una entrada para el software, de modo que lo que dibuja esta capa, y quién puede capturarlo, importa para la seguridad además de para la apariencia.

IMPLICACIONES PARA EL NEGOCIO

**Seguridad.** Un fabricante de software debería identificar a qué decodificadores y motores de fuentes pueden llegar los archivos no confiables en cada producto, y saber qué copias están empaquetadas. Cuando llegue el próximo aviso de libwebp o FreeType, ese mapa convierte una semana de búsqueda en una hora de parcheo.

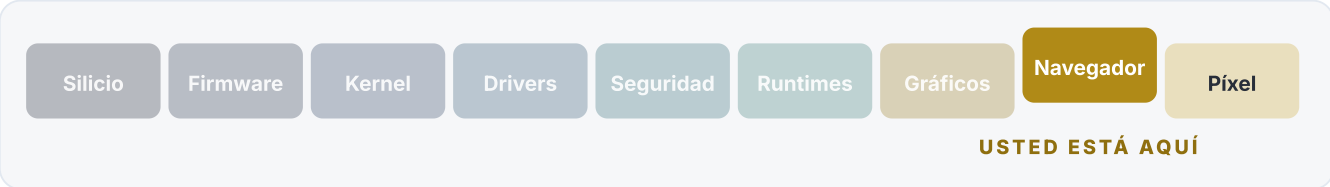
**Costo.** Un operador de una flota de dispositivos debería fijar y probar las versiones de los controladores de GPU de sus quioscos y pantallas de punto de venta, porque una actualización de controlador que rompe el renderizado cuesta visitas de campo. Conviene presupuestar la adaptación a las API que el proveedor del sistema operativo declare obsoletas como trabajo planificado, no como sorpresas.

**Valor.** El soporte de texto abre mercados: el nivel de GB 18030, la escritura de derecha a izquierda y un modelado de texto correcto son requisitos de venta para una empresa regulada o un fabricante de software que vende en el extranjero. Poseer un renderizador propio puede aportar velocidad y coherencia, pero solo si también se financia el trabajo de accesibilidad que conlleva.

**Próximo sprint:** para cada producto, listar los decodificadores a los que pueden llegar los bytes no confiables y quién parchea cada uno.

CAPÍTULO SIETE

# El navegador, el segundo sistema operativo




Basta abrir el administrador de tareas de cualquier computadora portátil de oficina y contar las copias de un motor de navegador. Está el navegador en sí, luego la app de chat, la app de reuniones, el gestor de contraseñas y el panel de ayuda dentro de su propio producto. Cada uno ejecuta un motor que procesa entradas no confiables de internet, y cada uno se parchea según el calendario de otro. Para la mayor parte del software empresarial, el motor de navegador es hoy el sistema operativo con el que el código realmente habla.

## Tres motores, tres dueños

Un **motor de navegador** es el código que convierte HTML, CSS y JavaScript en píxeles y comportamiento. Procesa la página, calcula el diseño, ejecuta los scripts, decodifica imágenes y fuentes y entrega una imagen terminada al compositor descrito en el capítulo seis. Hoy importan tres motores, y cada uno tiene un dueño distinto.

**Blink y V8.** Blink es el motor de renderizado del proyecto de código abierto Chromium. Google lo presentó el 3 de abril de 2013 como "un nuevo motor de renderizado de código abierto basado en WebKit", y dijo que el diseño multiproceso de Chromium había hecho demasiado complejo compartir un mismo motor con otros navegadores. V8, el motor de JavaScript que lo acompaña, se describe a sí mismo como "el motor de JavaScript y WebAssembly de alto rendimiento y código abierto de Google, escrito en C++". Chromium es abierto, y empresas como Microsoft contribuyen a él, pero las nuevas características web se publican mediante el proceso "Intent to Ship" propio de Blink y sus responsables de API. En la práctica, Google marca el ritmo.

**WebKit.** Apple anunció Safari el 7 de enero de 2003 con un motor "basado en KHTML, del proyecto de código abierto Konqueror de KDE". Ese motor se convirtió en WebKit. Es de código abierto con licencias de tipo BSD y LGPL, y el nombre y el logotipo son marcas registradas de Apple. Apple fija su dirección, y otras empresas mantienen adaptaciones, como WebKitGTK en Linux.

**Gecko.** Gecko es el motor de Firefox, mantenido por Mozilla. Nació del proyecto Mozilla, que Netscape inició cuando abrió el código fuente de su navegador en 1998. Es el único de los tres cuyo dueño principal no es también dueño de una plataforma.

En cada sistema de escritorio, el navegador predeterminado y el motor que las apps pueden tomar prestado vienen de lugares distintos. La tabla compara elementos equivalentes.

| Pregunta | Windows | Linux | macOS |
|---|---|---|---|
| **Navegador y motor del proveedor** | Edge: Chromium (Blink, V8) | No hay un único navegador del proveedor; la distribución elige | Safari: WebKit |
| **Web view que una app puede tomar prestada** | WebView2 (Chromium) | WebKitGTK (WebKit) o Qt WebEngine (Chromium) | WKWebView (WebKit del sistema) |
| **Quién parchea el motor prestado** | Microsoft, mediante el runtime Evergreen; o la app, con Fixed Version | Los mantenedores de paquetes de la distribución | Apple, mediante actualizaciones del sistema y de Safari |
| **Motor empaquetado en la app en su lugar** | Electron y frameworks similares: una copia de Chromium por app, parcheada por cada fabricante de la app | | |
| **Quién fija la hoja de ruta del motor** | Google, con Microsoft como gran contribuidor | El proyecto original del motor que corresponda | Apple |

La parte de la web que renderiza cada motor cambia cada mes, así que este libro no publica cifras de cuota de mercado. El resumen honesto es que la mayor parte de la navegación de escritorio actual se hace en navegadores basados en Chromium, y que WebKit y Gecko siguen siendo las dos alternativas independientes. Si una decisión depende de la cuota exacta, conviene obtener una cifra con fecha el día en que se decida.

## Microsoft decidió alquilar el motor

La lección más clara de este capítulo viene de la empresa que más poseía. Microsoft tenía su propio motor, EdgeHTML, dentro del primer navegador Edge. El 6 de diciembre de 2018 anunció que "[tiene] la intención de adoptar el proyecto de código abierto Chromium en el desarrollo de Microsoft Edge para escritorio". Prometió una "plataforma web compatible con Chromium" y dijo que contribuiría con mejoras a Chromium. El nuevo Edge, "reconstruido sobre el proyecto de código abierto Chromium", salió de la versión preliminar el 15 de enero de 2020.

Nadie lo obligó. Microsoft tenía el dinero y los ingenieros para mantener EdgeHTML. Decidió que un motor aparte costaba tiempo de pruebas a los desarrolladores web y compatibilidad a sus usuarios, y que el costo de poseerlo superaba su valor. Así que pasó de **construir** a **financiar y alquilar**: alquila la hoja de ruta de Blink y paga por tener voz en ella con sus propios ingenieros.

> *A Microsoft no la echaron; decidió dejar de mantener su propio motor y compró una voz en el que alquila.*

Esa es la tesis del libro en un solo movimiento. Poseer una capa no hizo mejor a Edge. Cambió quién podía corregir un defecto de compatibilidad web, y la respuesta había pasado a ser "solo nosotros, y despacio". Después del cambio, una corrección de un error de seguridad de Chromium llega a Edge a

través del proyecto original compartido. Las notas de seguridad de las versiones de Edge muestran ese ritmo: las correcciones de Chromium llegan como actualizaciones de Edge en cuestión de días.

### Integración: cuatro web views, cuatro modelos de actualización

Las apps rara vez construyen un navegador. Toman uno prestado mediante una **web view**: un control que muestra contenido web dentro de una ventana nativa. La web view que se elija decide quién parchea el motor dentro de su producto.

- **WebView2 en Windows.** Disponible de forma general para apps Win32 en C y C++ el 19 de octubre de 2020, y para .NET el 20 de noviembre de 2020. Tiene dos modos. En **Evergreen**, todas las apps comparten una copia del runtime, que "se actualiza automáticamente sin que usted tenga que hacer nada" y recibe las mismas actualizaciones que el navegador Edge. En **Fixed Version**, "usted controla el momento de las actualizaciones": se empaqueta un runtime concreto de más de 250 MB con la app, y nunca se actualiza salvo que se distribuya uno nuevo.
- **Electron.** Electron se describe como un framework que integra "el motor de navegador Chromium y el runtime de Node.js en su binario". Cada app Electron lleva su propio motor. El equipo de Electron da soporte a las tres últimas versiones mayores estables y publica una nueva versión mayor aproximadamente cada ocho semanas. El fabricante de la app decide cuándo reciben los usuarios el nuevo motor.
- **WKWebView en macOS e iOS.** Las apps usan el WebKit del sistema. Apple lo parchea con las actualizaciones del sistema y de Safari. El fabricante de la app no distribuye el motor y no puede parchearlo.
- **Android WebView.** La WebView de Android está basada en Chromium. Desde Android 5.0, según las notas de versión de Google, "la capa de Chromium ahora se puede actualizar desde Google Play", al margen del sistema operativo. Esto se aplica a los dispositivos con Google Play. En las compilaciones de Android sin él, el fabricante del dispositivo debe distribuir las actualizaciones de WebView; la propia guía de Chromium advierte que la WebView precompilada de AOSP "actualmente no se actualiza con un calendario regular".

Ninguna de ellas es la correcta para todos los productos. Un fabricante de quioscos puede querer Fixed Version para que una actualización del navegador no cambie una pantalla certificada. Un fabricante de software de escritorio suele querer Evergreen para que un zero-day sea la emergencia de otro. Lo importante es elegir a propósito y dejar la elección por escrito.

### La topología de actualización es arquitectura

La **topología de actualización** es la forma de la ruta de parches: cuántas copias de un componente hay en una máquina y quién debe actuar antes de que cada copia quede corregida. La decide una elección de dependencias al principio de un proyecto, y aparece años después, en medio de un incidente.

Septiembre de 2023 lo hizo visible: una falla explotada en libwebp, el decodificador de imágenes WebP empaquetado en Chromium y en muchos otros productos (CVE-2023-4863), tuvo que corregirse por separado en cada copia empaquetada. El capítulo once cuenta esa historia completa y mide el retraso de cada dueño.

En ese incidente la topología decidió el resultado. Una app con WebView2 Evergreen recibió la corrección del motor con la siguiente actualización del runtime y un reinicio. Cada app Electron tuvo que adoptar la nueva versión de Electron, recompilar, probar, firmar y distribuir su propia actualización, con su propio calendario, y luego esperar a que los usuarios la instalaran. Una app con WebView2 Fixed Version estaba en la misma situación que Electron. Las apps con WKWebView siguieron una tercera ruta. En los sistemas de Apple, WebKit decodifica las imágenes mediante el framework ImageIO de Apple, y la falla de imagen explotada de Apple esa semana estaba en ImageIO (CVE-2023-41064, parte de la cadena de ataque BLASTPASS que informó Citizen Lab). Apple no ha dicho qué código falló, pero los análisis de entonces, como el de Ben Hawkes del 21 de septiembre de 2023, concluyeron que muy probablemente era el mismo error de WebP, ya que ImageIO había añadido hacía poco soporte para WebP y Apple había informado de la falla de libwebp a Google el día anterior a su propia corrección. Esa era la copia de Apple de la ruta, parcheada una sola vez por la actualización del sistema operativo de Apple para las apps que usan los decodificadores del sistema: iOS 16.6.1 y macOS Ventura 13.5.2, el 7 de septiembre de 2023, lo corrigieron para esas apps una vez instalada la actualización.

FIGURA 7.1

### Una falla del motor, tres rutas de parche




Tres topologías para una misma clase de falla. En el incidente de WebP de 2023, las rutas de Electron y WebView2 llevaban libwebp; la ruta de Apple pasaba por el propio decodificador de imágenes del sistema de Apple, ImageIO, muy probablemente con el mismo error, y una sola actualización del sistema operativo de Apple lo corrigió para las apps que usan ese decodificador. Empaquetar da a una app el control de la versión de su motor y la hace responsable de cada parche. Compartir deja el calendario de parches en manos del dueño del runtime.

## El navegador está construido como un sistema operativo

Un navegador moderno hace los trabajos de un kernel, un nivel más arriba. Ejecuta programas no confiables, los separa, les da almacenamiento, planifica su trabajo y les permite llegar a la GPU.

**Modelo de procesos.** Chromium ejecuta un **proceso del navegador** para la interfaz de usuario y **procesos de renderizado** separados para el contenido web. Los procesos de renderizado están aislados entre sí, así que una falla en uno normalmente deja a los demás funcionando. Se comunican con el proceso del navegador mediante un sistema interno de mensajes, igual que las apps hablan con un kernel mediante llamadas al sistema.

**Sandbox.** Como los procesos de renderizado se ejecutan en sus propios procesos, el sistema operativo puede restringirlos. En Chromium un proceso de renderizado solo llega a la red a través de un servicio de red, y el sistema operativo limita su acceso a archivos, pantalla y entradas. El navegador se apoya en las mecanismos de sandbox de Windows, Linux y macOS; no las reemplaza.

**Aislamiento de sitios.** Chrome activó el **aislamiento de sitios** (site isolation) para todos los sitios en escritorio en Chrome 67. Busca mantener las páginas de sitios distintos en procesos de renderizado distintos, lo que "ayuda a defenderse de ataques especulativos de canal lateral (por ejemplo, Spectre)". Una falla de hardware del capítulo uno se respondió, en parte, con un cambio de arquitectura del navegador.

**Almacenamiento.** El navegador da a cada sitio su propio almacenamiento: cookies, almacenes locales de clave-valor y bases de datos, delimitados por el origen del sitio. Es un sistema de archivos con una regla de control de acceso, gestionado por el navegador en lugar del kernel.

**Cómputo y gráficos. WebAssembly** es un formato de código compacto y de bajo nivel que los navegadores ejecutan a una velocidad cercana a la nativa. El W3C anunció el 5 de diciembre de 2019 que la WebAssembly Core Specification se había convertido en un estándar web oficial. **WebGPU** da a las páginas web un acceso moderno a la GPU; Chrome 113 lo activó por defecto en ChromeOS, en Windows con Direct3D 12 y en macOS. Cada nueva capacidad es también una nueva superficie de ataque que llega al controlador de la GPU desde una página web.

Así que un equipo que distribuye una web view distribuye un segundo sistema operativo dentro de su producto. Hereda el modelo de procesos de ese sistema, su sandbox, su ritmo de parches y sus errores.

### Cuando el dueño de la plataforma elige el motor

En iPhone y iPad, la directriz 2.5.6 de App Review de Apple dice: "Las apps que navegan por la web deben usar el framework WebKit adecuado y WebKit JavaScript". Durante años eso significó que todo navegador de iOS, fuera cual fuera su marca, era por dentro un navegador WebKit. Una empresa podía elegir Chrome o Firefox en iOS y seguir ejecutando el motor de Apple.

La regulación cambió esto en dos mercados. El 25 de enero de 2024, Apple anunció cambios para cumplir la Ley de Mercados Digitales de la UE (Digital Markets Act), entre ellos "motores de navegador distintos de WebKit para apps de navegador y apps con experiencias de navegación integradas". Los cambios llegaron a los usuarios de los 27 países de la UE en marzo de 2024 con iOS 17.4. A noviembre de 2026, las páginas para desarrolladores de Apple ofrecen dos entitlements, el Web Browser Engine Entitlement y el Embedded Browser Engine Entitlement: en la UE para dispositivos con "como mínimo iOS 17.4 o iPadOS 18", y en Japón "en iOS 26.2 y posteriores" (la numeración por año que Apple usa desde 2025), donde el entitlement integrado solo está abierto a un "administrador de motor de navegador" (browser engine steward), la entidad con la responsabilidad principal de un motor propio.

Ambas páginas exigen el compromiso de "mitigar las vulnerabilidades que se estén explotando … en un plazo oportuno (por ejemplo, 30 días para las clases más sencillas de vulnerabilidades explotadas activamente)". Una app con un motor integrado "debe enviar una actualización de su app con esa nueva versión en un plazo de quince (15) días naturales" desde una nueva versión del motor.

Para un CTO este es el caso más claro del libro de una capa que no se puede elegir. En iOS, fuera de esos mercados, el dueño de la plataforma toma la decisión del motor. Su contenido web, su navegador integrado en la app y su app híbrida se ejecutan sobre el WebKit de Apple, con el calendario de parches de Apple.

**CÓMO LLEGÓ A SER ASÍ**

Mosaic, escrito en el National Center for Supercomputing Applications por Marc Andreessen, Eric Bina y otros, se publicó el 23 de enero de 1993 e hizo visual la web. Le siguió Netscape, y Microsoft respondió con Internet Explorer, empaquetado con Windows. Esa pelea fue la primera guerra de los navegadores. El 22 de enero de 1998 Netscape anunció que publicaría el código fuente de su navegador, y de ahí nació el proyecto Mozilla; Firefox 1.0 salió el 9 de noviembre de 2004. El 1 de septiembre de 2008 Google anunció Chrome, un navegador de código abierto construido sobre WebKit con su propio diseño multiproceso, y lanzó la beta al día siguiente en más de 100 países. Los pasos posteriores, Blink en 2013 y Edge sobre Chromium en 2020, se cuentan más arriba.

## Esta capa en la lista de materiales del píxel

| Componente | Quién lo escribe | Quién paga | Quién puede actualizarlo | Plazo de soporte | ¿Expuesto a entradas no confiables? | Su decisión |
|---|---|---|---|---|---|---|
| Motor Chromium (Blink, V8) en el navegador | Proyecto Chromium, liderado por Google | Google, más contribuidores como Microsoft | El proveedor del navegador | Ciclo de versiones del navegador | Sí, cada página web | Alquilar |
| Runtime de WebView2, Evergreen | Microsoft, sobre Chromium | Microsoft | Microsoft, automáticamente | Sigue a Edge estable | Sí, si la app carga contenido externo | Alquilar |
| Chromium empaquetado por Electron en su app | Proyecto Chromium y equipo de Electron | Google, patrocinadores de Electron, usted | Usted, al publicar una versión | Las tres últimas versiones mayores de Electron | Sí, si la app carga contenido externo | Alquilar el motor; construir y contar con personal para su ruta de parches |
| WebKit del sistema (WKWebView) | Apple y contribuidores de WebKit | Apple | Solo Apple | Ventana de soporte del sistema operativo de Apple | Sí | Alquilar; obligatorio en iOS fuera de la UE y Japón |
| Decodificador de imágenes compartido (libwebp) | Un pequeño equipo del proyecto original | Sobre todo Google | Cada producto que empaqueta una copia | Versiones del proyecto original | Sí, cualquier imagen | Alquilar y financiar; rastrear cada copia |

LENTE DE SEGURIDAD

**¿Quién puede cambiarlo?** El mantenedor del proyecto original del motor, el proveedor de la web view y, si el motor va empaquetado, el propio equipo de publicación. En iOS, fuera de la UE y Japón, solo Apple puede cambiar el motor sobre el que se ejecuta su contenido web.

**¿Qué tan rápido se puede corregir?** Tan rápido como la copia más lenta de la máquina. Un runtime compartido queda corregido cuando su dueño publica y la app se reinicia. Un motor empaquetado solo queda corregido cuando cada fabricante de apps publica y cada usuario instala.

**¿Qué cambia la IA aquí?** El código de los motores es grande, público y lleno de analizadores, y la IA ya encuentra fallas en él: la versión estable de Chrome del 28 de octubre de 2025 (Chrome 142) atribuye a Google Big Sleep, un agente de IA, cuatro vulnerabilidades de V8. Cada hallazgo así se convierte en una versión del motor que cada copia empaquetada debe repetir, así que el equipo con menos copias empaquetadas tiene menos trabajo.

IMPLICACIONES PARA EL NEGOCIO

**Seguridad.** El **fabricante de software** que distribuye una app Electron controla una ruta de parches de navegador, lo haya planeado o no. Conviene poner la actualización de Electron con los mismos plazos de incidente que sus propias correcciones críticas y medir los días que pasan desde una versión de seguridad de Chromium en el proyecto original hasta su actualización publicada.

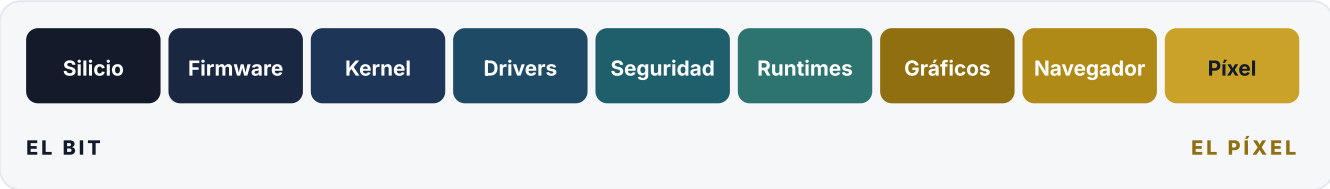
**Costo.** Cada motor empaquetado es una copia en memoria, en disco y en su plan de pruebas. La **empresa regulada** que debe demostrar qué se ejecuta en sus escritorios debería contar las copias del motor por máquina y preferir un runtime compartido y parcheado de forma centralizada, salvo que tenga una razón escrita para fijar la versión.

**Valor.** El **operador de una flota de dispositivos** con quioscos de registro puede necesitar Fixed Version para mantener estable una pantalla certificada, pero entonces cada parche es trabajo suyo. Fijar la versión cambia estabilidad por trabajo de parches, así que necesita un responsable y una fecha de revisión. La propia decisión de Microsoft muestra que alquilar bien un motor puede ser el mejor negocio.

**Para empezar:** listar cada motor de navegador que su producto distribuye o toma prestado, e indicar quién parchea cada uno y cuántos días tarda.

CAPÍTULO OCHO

# Cadena de herramientas, distribución y los bordes de un sistema operativo




Cada capa de los siete capítulos anteriores llegó a la máquina de su cliente por el mismo camino: alguien la compiló, la empaquetó, la firmó y la envió por una ruta de actualización. Esa ruta también es una capa, y toca a todas las demás. Decide en qué código confía una máquina, qué tan rápido llega una corrección y si es posible recuperarse cuando un proveedor falla. Es también la capa que una empresa mediana puede controlar con más posibilidades reales.

## Compiladores y compilaciones: donde nace cada bit

Una **cadena de herramientas (toolchain)** es el conjunto de programas que convierte el código fuente en algo que una máquina ejecuta: el compilador, el enlazador, el sistema de compilación y las herramientas que empaquetan y firman el resultado. Cada uno de los tres sistemas tiene una cadena de herramientas nativa principal, y cada una tiene un dueño distinto.

**Windows.** El compilador de C y C++ de Microsoft, MSVC, se distribuye con Visual Studio. Microsoft lo escribe, Microsoft lo paga y Microsoft decide cuándo cambia.

**Linux.** La GNU Compiler Collection, GCC, ha compilado la mayor parte del mundo Linux desde que GCC 1.0 se publicó el 23 de mayo de 1987. A su lado está **LLVM**, que se describe a sí mismo como "una colección de tecnologías modulares y reutilizables de compiladores y toolchains". LLVM nació como proyecto de investigación en la Universidad de Illinois. Su front end de C y C++, **Clang**, ofrece una interfaz de línea de comandos (driver) compatible con GCC y otro compatible con MSVC, de modo que la misma familia de compiladores aparece hoy en los tres sistemas.

**macOS.** Apple distribuye su propia versión de Clang, llamada Apple Clang, dentro de Xcode, y detalla sus cambios en las notas de versión de Xcode. Apple decide qué características del lenguaje y qué advertencias llegan a los desarrolladores de Apple, y cuándo.

El compilador es la primera herramienta que toca cada bit de su producto. Decide qué endurecimiento de seguridad viene activado por defecto y qué reglas del lenguaje se aplican. También decide si dos compilaciones del mismo código fuente producen el mismo binario.

### Empaquetado y tiendas: la ruta de actualización es una función del producto

Un programa compilado todavía tiene que llegar a una máquina y mantenerse al día. El **formato de empaquetado** y el **canal de distribución** deciden quién firma el código, quién puede enviar una actualización y quién puede detenerla.

**Windows.** MSIX es el formato de empaquetado moderno de Windows. "Todos los paquetes MSIX deben firmarse antes de la instalación", dice la documentación de Microsoft, y al actualizar solo se descargan los bloques de 64 KB que cambiaron, gracias al mapa de bloques del paquete. **WinGet**, el Windows Package Manager, es un cliente de línea de comandos de código abierto que instala y actualiza aplicaciones desde un repositorio comunitario y otras fuentes. A noviembre de 2026, Microsoft lo admite en Windows 10 versión 1809 y posteriores, Windows 11 y Windows Server 2025.

**Linux.** Cada distribución tiene su propio sistema de paquetes: `apt` y archivos `.deb` en Debian y Ubuntu, paquetes `rpm` en Fedora y Red Hat Enterprise Linux. Dos formatos más recientes funcionan en varias distribuciones. **Flatpak** se define como "un sistema para compilar, distribuir y ejecutar aplicaciones de escritorio aisladas en Linux", con runtimes compartidos por debajo. **Snap**, gestionado por Canonical a través de la Snap Store, busca actualizaciones cuatro veces al día por defecto y confina la mayoría de las aplicaciones con mecanismos del kernel como AppArmor, seccomp y namespaces.

**macOS.** Las aplicaciones llegan a los usuarios por la Mac App Store o de forma directa. La **notarización** es la verificación automática que Apple hace de una aplicación firmada antes de su publicación. Las noticias para desarrolladores de Apple del 3 de septiembre de 2019 indican que, a partir de macOS Catalina, "el software para Mac distribuido fuera de la Mac App Store debe estar notarizado por Apple". En la práctica, la verificación la aplica **Gatekeeper**, que por defecto espera que el software descargado esté firmado por un desarrollador registrado y notarizado. La guía de seguridad de Apple también dice que los usuarios pueden saltarse esa política para abrir otro software, salvo que un servicio de gestión de dispositivos lo restrinja. Así que, por defecto y para usuarios comunes, Apple está en la ruta de publicación tanto si se usa su tienda como si no.

| Pregunta | Windows | Linux | macOS |
|---|---|---|---|
| **Compilador nativo principal** | MSVC (Clang disponible) | GCC y LLVM/Clang | Apple Clang en Xcode |
| **Formatos de paquete de aplicaciones** | MSIX, firmado; instaladores MSI y EXE más antiguos | deb, rpm; Flatpak y Snap entre distribuciones | Paquetes de aplicación (bundles) y paquetes de instalación |
| **Quién está en la ruta de publicación** | La Microsoft Store, si se usa; la firma en MSIX | Los mantenedores de la distribución, o la tienda de Flatpak o Snap | Apple: la App Store, o la notarización fuera de ella |
| **Quién arma el sistema operativo a partir de piezas** | Microsoft | La distribución, con el kernel de la comunidad del kernel y miles de proyectos | Apple |

Conviene ver la ruta de actualización como una **capacidad del producto**, no como simple infraestructura. ¿Se puede enviar una corrección a todos los clientes en un día? ¿Se puede detener una actualización defectuosa a mitad del despliegue? ¿Se puede revertir? Un equipo que responde que sí ha construido algo valioso, aunque alquile todas las demás capas de este libro. El capítulo 11 calcula el costo del caso contrario: la falla de libwebp de 2023, que obligó a cada aplicación con su propio motor de navegador a sacar su propia versión de emergencia.

### Dónde termina un sistema operativo: el kernel y la distribución

En Windows y macOS, la pregunta "¿quién hizo el sistema operativo?" tiene una sola respuesta. En Linux tiene dos. El **núcleo (kernel)** lo escribe una comunidad grande, con una nueva versión principal "cada dos o tres meses", según la propia guía de proceso del kernel, y el equipo de la rama estable, que retroporta (backport) las correcciones importantes a versiones anteriores. La **distribución**, como Debian, Fedora, Ubuntu o Red Hat Enterprise Linux, elige un kernel, añade miles de otros proyectos, los parchea y compila, firma los paquetes y fija el plazo de soporte. Quien "usa Linux" alquila una distribución, y la distribución alquila el kernel.

En esa costura estuvo a punto de colarse un ataque grave a la cadena de suministro. Durante años, `xz`, una biblioteca de compresión usada en las distribuciones Linux, la mantuvo un solo voluntario no remunerado, Lasse Collin. En 2022, varias cuentas en la lista de correo del proyecto lo presionaron para que cediera el mantenimiento; el 8 de junio de 2022 respondió que su "capacidad de dedicarle tiempo ha sido bastante limitada, sobre todo por problemas de salud mental de largo plazo" y que un colaborador llamado Jia Tan podría asumir un papel mayor, y a finales de ese año Jia Tan figuraba como comantenedor. El 29 de marzo de 2024, un ingeniero llamado Andres Freund informó en una lista pública de seguridad que xz tenía una puerta trasera. El aviso del propio proyecto dice que los tarballs de las versiones 5.6.0 y 5.6.1 que contienen la puerta trasera fueron "creados y firmados por Jia Tan". Freund lo detectó en la rama inestable de Debian por lo que llamó "inicios de sesión con ssh que consumían mucha CPU". Incluso una ejecución de prueba del programa `sshd`, sin arrancarlo como servidor, tardaba "unos 0,5 s" en su sistema más antiguo con la biblioteca con puerta trasera, frente a "unos 0,01 s" sin ella. El código llegaba a sshd solo a través de un parche de la distribución que enlazaba OpenSSH con `libsystemd`, que depende de liblzma. Red Hat advirtió a los usuarios de las ramas de desarrollo de Fedora, y CISA recomendó volver a una versión limpia como la 5.4.6. Según el primer informe, las versiones maliciosas habían llegado a las distribuciones casi solo como paquetes previos a la publicación, así que se detectó antes de las versiones estables que usan la mayoría de las empresas.

> *La puerta trasera de xz no rompió el kernel ni el compilador. Entró por un proyecto pequeño y un tarball de publicación, y casi llegó al punto donde una distribución se convierte en sistema operativo.*

El borde de un sistema operativo también se desplazó hacia afuera. Para una empresa que opera servicios, el **plano de control de la nube** es hoy el sistema operativo del centro de datos: planifica el trabajo, reparte recursos y aísla a los inquilinos. Kubernetes llegó a la versión 1.0 en julio de 2015, cuando Google anunció que estaba listo para producción y lo donó a la nueva Cloud Native Computing

Foundation, bajo la Linux Foundation. Un clúster de Kubernetes cumple las funciones de un sistema operativo, como la planificación, la resolución de nombres, el almacenamiento y el control de acceso, y tiene un dueño, un ciclo de versiones y un plazo de soporte como cualquier otra capa.

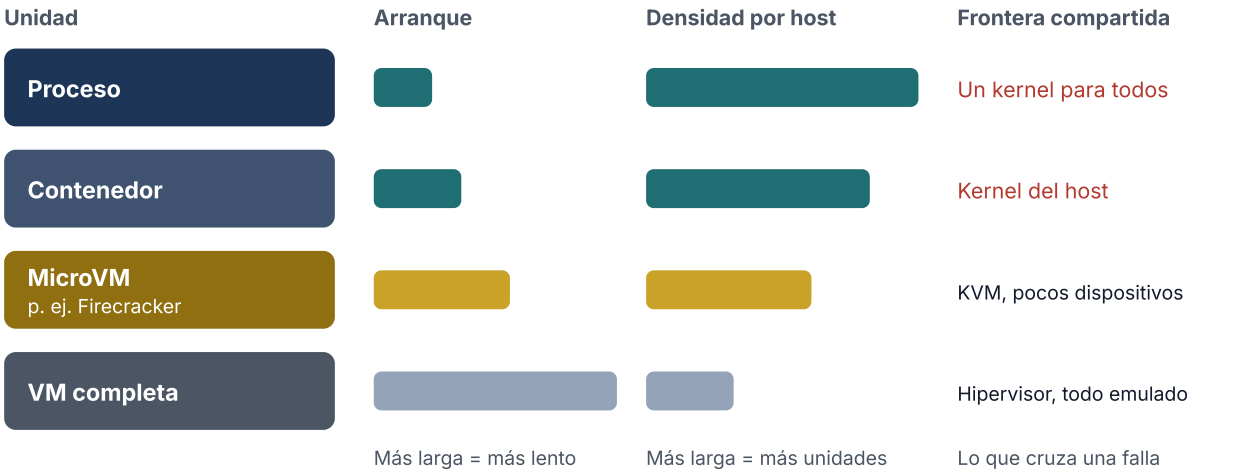
### El aislamiento tiene un costo unitario

Cada frontera de aislamiento del capítulo dos reaparece a escala de centro de datos, y cada una tiene un precio por carga de trabajo. Hay cuatro peldaños comunes.

- **Proceso.** El sistema operativo separa la memoria, pero todos los procesos comparten un mismo kernel. Arrancar uno es barato y se pueden ejecutar muchos. Una falla del kernel los alcanza a todos.
- **Contenedor.** Un proceso con su propia vista de archivos, red e identificadores de proceso, construido con mecanismos del kernel. Por debajo sigue habiendo un kernel compartido, así que el kernel sigue siendo la frontera. **gVisor**, escrito en Go, estrecha esa frontera con un "kernel de aplicación" llamado Sentry. Sentry intercepta las llamadas al sistema de la aplicación y las implementa por su cuenta; a su vez, solo usa un conjunto restringido de llamadas al sistema del host.
- **MicroVM.** Una máquina virtual pequeña con su propio kernel, sin los dispositivos que no necesita. Firecracker ejecuta sus invitados mediante KVM, la misma virtualización por hardware que usan las máquinas virtuales completas en Linux, y solo les ofrece un modelo de dispositivos mínimo. AWS anunció **Firecracker**, escrito en Rust, el 26 de noviembre de 2018. El anuncio decía que una microVM podía arrancar en "apenas 125 ms" con unos 5 MiB de sobrecarga de memoria, y que Firecracker ya era la base de AWS Lambda y AWS Fargate. El diseño se describe en el artículo de NSDI 2020 "Firecracker: Lightweight Virtualization for Serverless Applications".
- **Máquina virtual completa.** Un sistema operativo invitado completo sobre un hipervisor que emula un conjunto completo de dispositivos. Arranca más lento y pesa más que una microVM. No es, por sí sola, una frontera más fuerte: en ambos peldaños, la solidez depende del hipervisor, del modelo de dispositivos que expone y de cómo está configurado.

FIGURA 8.1

### La escalera del aislamiento




Los dos peldaños inferiores dan a cada carga su propio kernel, a cambio de arranque y densidad.

Las barras son cualitativas, no medidas. La escalera no es una clasificación de seguridad entre los dos peldaños de máquina virtual: ambos usan virtualización por hardware, y un modelo de dispositivos más pequeño significa menos código al alcance de un atacante. Las cifras de AWS de 2018 para Firecracker eran un arranque en apenas 125 ms y unos 5 MiB de sobrecarga por microVM. Conviene medir las propias cargas de trabajo antes de ponerles precio.

La elección es un costo unitario, no una cuestión de gusto. Un fabricante de software que ejecuta código ajeno, como plug-ins o scripts de clientes, puede asumir el costo de las microVM porque un kernel compartido hace que una falla pueda afectar a demasiadas cargas de trabajo. Un equipo que solo ejecuta sus propios servicios de confianza puede quedarse en contenedores e invertir el ahorro en parchear más rápido.

## La cadena de herramientas es una frontera de confianza

Leer el código fuente no demuestra lo que hace un binario. Un compilador, un servidor de compilación o un script de publicación comprometido puede añadir un comportamiento que el código fuente no muestra. Dos prácticas responden a esto.

**Compilaciones reproducibles.** El proyecto reproducible-builds define una compilación como reproducible cuando, con el mismo código fuente, el mismo entorno de compilación y las mismas instrucciones, "cualquier parte puede recrear copias idénticas bit a bit de todos los artefactos especificados". Si dos partes independientes compilan la misma versión y obtienen los mismos bytes, se descubre a un atacante que solo controla una máquina de compilación. No sirve si ambas partes parten del mismo código fuente malicioso o usan la misma toolchain comprometida: reproducirán el mismo binario malicioso. Ese es el límite que Thompson describió en 1984.

El caso xz es exactamente ese límite. Las compilaciones reproducibles no lo habrían detectado, porque el tarball malicioso era la entrada de código fuente: cualquiera que compilara a partir de él habría producido la misma biblioteca con puerta trasera, bit a bit. El disparador de la puerta trasera "nunca se incluyó en el repositorio Git", dice el aviso de xz; llegó solo en los tarballs de publicación, a través de una macro de compilación modificada, `build-to-host.m4`, que el repositorio no contenía. Lo que ayuda contra ese ataque es otra cosa: comparar cada tarball de publicación con el repositorio etiquetado y explicar cada diferencia, registrar de dónde vino el código fuente (procedencia del código fuente) y revisar los archivos de compilación generados, no solo el código escrito a mano.

**Listas de materiales de software.** Un **SBOM** es una lista formal de los componentes que hay dentro de un software y de cómo se relacionan. La Orden Ejecutiva 14028 de Estados Unidos, "Improving the Nation's Cybersecurity", emitida el 12 de mayo de 2021, ordenó al Departamento de Comercio, junto con la NTIA, publicar los elementos mínimos de un SBOM; la NTIA los publicó el 12 de julio de 2021. Un SBOM indica qué paquetes se entregan. Por sí solo no demuestra que el binario se compiló a partir de ellos, y por eso se combina con compilaciones reproducibles y firma.

**VEX.** Un SBOM tampoco dice si una falla listada importa. Esa es la tarea de **VEX**, Vulnerability Exploitability eXchange: una declaración del proveedor, legible por máquina, que indica si un producto está afectado por una vulnerabilidad concreta. El documento de CISA "Minimum Requirements for VEX" (abril de 2023) admite cuatro estados: no afectado, afectado, corregido y en investigación. Una declaración de "no afectado" necesita una justificación, como "código vulnerable fuera de la ruta de ejecución"; una de "afectado" debe decir qué tiene que hacer el cliente. VEX es una afirmación del proveedor, no una prueba: expresa lo que el proveedor cree, y "en investigación" significa que todavía no lo sabe. Entregar un archivo VEX junto a cada SBOM convierte "contenemos xz" en "contenemos xz, y esto es lo que indica si hay que actuar".

**CÓMO LLEGÓ A SER ASÍ**

En su conferencia del Premio Turing, publicada como "Reflections on Trusting Trust" en Communications of the ACM en agosto de 1984, Ken Thompson describió un compilador que insertaba una puerta trasera en el programa de inicio de sesión del sistema. Luego fue un paso más allá: el compilador también reconocería cuándo se estaba compilando a sí mismo y añadiría el mismo truco al nuevo compilador. Una vez que existiera el compilador modificado, se podía limpiar el código fuente y la puerta trasera sobreviviría en cada compilación futura. Ninguna lectura del código fuente la revelaría.

Cuatro décadas después, la lección sigue vigente. Las respuestas prácticas de la industria detectan cada una un ataque distinto, y ninguna los detecta todos. La firma detecta un paquete alterado después de que el firmante lo publicó, pero no a un firmante malicioso: los tarballs de xz los firmó Jia Tan. Las compilaciones reproducibles detectan una máquina de compilación manipulada, pero no un código fuente malo. Comparar los tarballs de publicación con el repositorio detecta código colado en un archivo de publicación. Un SBOM no detecta ningún ataque; indica dónde cayó un componente malo. Ninguna elimina la necesidad de confiar en las personas que escriben y aprueban el código fuente.

### Capacidad de recuperación: poseer el código fuente no es poseer la recuperación

Muchos equipos creen que son dueños de una capa porque tienen su código fuente. La propiedad se pone a prueba el día en que un proveedor falla: una empresa proveedora cierra, un certificado de firma vence, un mantenedor desaparece o se descubre que una versión es maliciosa. Ese día la pregunta no es "¿tenemos el código?". Es si se puede **recompilar**, **firmar** con una clave propia, **desplegar** por la ruta de actualización propia, **revertir** si la corrección resulta incorrecta y **restablecer** el funcionamiento de los equipos, y cuánto tarda cada paso.

Por eso cada fila de la lista de materiales del píxel necesita un número más: un **tiempo de recuperación probado**. Probado significa un simulacro que alguien realmente ejecutó, no una estimación. Para las capas alquiladas, la respuesta suele ser "esperamos al proveedor", lo cual es honesto y aceptable si queda escrito y acordado.

> **EJEMPLO**
>
> Un caso hipotético. Un grupo hotelero opera quioscos de check-in sobre una imagen Linux fijada con un motor de navegador empaquetado. Su simulacro: suponer que el proveedor del motor deja de publicar correcciones. El equipo recompila la imagen desde el código fuente en su propia cadena de compilación (build pipeline), la firma con su propia clave, la despliega en diez quioscos, la revierte y luego la despliega en toda la flota. El primer simulacro tarda once días, sobre todo porque la compilación necesita una máquina que nadie conservó. Tras corregir eso, el segundo simulacro tarda dos. El número entra en la lista de materiales, y el consejo de administración ve un tiempo de recuperación real en lugar de una esperanza.

## Esta capa en la lista de materiales del píxel

| Componente | Quién lo escribe | Quién paga | Quién puede actualizarlo | Plazo de soporte | ¿Expuesto a entradas no confiables? | Su decisión |
|---|---|---|---|---|---|---|
| Compilador (MSVC, GCC, Clang, Apple Clang) | Microsoft, proyecto GNU, proyecto LLVM, Apple | Proveedores y empresas que contribuyen | El proveedor; usted elige la versión | Ciclo de versiones del proveedor | En tiempo de compilación, si compila código externo como dependencias o contribuciones; no durante la ejecución en las máquinas de sus clientes. Uno comprometido contamina cada compilación | Alquilar; fijar y registrar versiones |
| Cadena de compilación y publicación, claves de firma | Su equipo, en servidores de compilación alquilados | Usted | Usted | El suyo | Sí, a través de las dependencias que descarga | Controlar (construir solo las piezas críticas para la confianza) |
| Distribución Linux y sus paquetes | Mantenedores de la distribución sobre proyectos originales (upstream) | Empresa de la distribución, patrocinadores, voluntarios | La distribución | La ventana de soporte de cada versión de la distribución | Sí, a través de servicios de red | Alquilar; financiar los proyectos pequeños de los que depende |
| Ruta de tienda o notarización (Microsoft Store, Snap Store, Apple) | Proveedor de la plataforma | Proveedor de la plataforma, con comisiones o la propia plataforma | El proveedor de la plataforma | Las reglas de la plataforma | Indirectamente | Alquilar; mantener un canal alternativo donde se permita |
| Runtime de aislamiento (contenedores, gVisor, Firecracker) | Proyectos abiertos, liderados en gran parte por proveedores de nube | Proveedores de nube | Su equipo de plataforma | Versiones del proyecto | Sí, aloja cargas no confiables | Alquilar; elegir el peldaño a propósito |

**LENTE DE SEGURIDAD**

**¿Quién puede cambiarlo?** Todos los que tienen acceso de escritura a su código fuente, sus dependencias, sus máquinas de compilación, sus claves de firma y su canal de distribución. El caso xz mostró que un colaborador de confianza de un pequeño proyecto original también pertenece a esa lista.

**¿Qué tan rápido se puede corregir?** Solo tan rápido como su ruta de actualización. Un equipo que puede recompilar, firmar y publicar en horas convierte cualquier corrección del proyecto original en protección el mismo día. Quien no pueda, dependerá de cada proveedor, cada vez.

**¿Qué cambia la IA aquí?** Los agentes de IA de programación que editan código fuente y archivos de compilación son nuevos colaboradores con acceso de escritura a esta capa, así que sus cambios necesitan la misma revisión y firma que los de una persona. Sus commits deben quedar en la revisión del control de código fuente, en la procedencia de compilación y sus declaraciones firmadas (attestations), como la procedencia SLSA, y en los registros de auditoría; el SBOM lista componentes, no quién los cambió.

**IMPLICACIONES PARA EL NEGOCIO**

**Seguridad.** El **fabricante de software** debería tratar su cadena de compilación y sus claves de firma como su sistema de producción más sensible. Fijar las versiones del compilador y de las dependencias, compilar de forma reproducible donde sea posible y publicar un SBOM y un archivo VEX con cada versión.

**Costo.** El aislamiento se paga por unidad. La **empresa regulada** que ejecuta cargas de clientes debería elegir su peldaño de la escalera a propósito, dejar escrito el porqué y presupuestarlo, en lugar de heredar lo que usó su primer clúster.

**Valor.** La ruta de actualización es una capacidad del producto que los clientes perciben. El **operador de una flota de dispositivos** que puede parchear cada quiosco en un día, y demostrarlo con un simulacro, tiene una posición contractual más fuerte y menos emergencias que otro con mejor hardware y ningún tiempo de recuperación probado.

**Este trimestre:** para cada capa que se dice poseer, ejecutar un simulacro de recuperación y anotar el tiempo probado en la lista de materiales.

CAPÍTULO NUEVE

# Setenta años de sistemas operativos




Cada propuesta comercial que llegue este año es una oscilación de un péndulo que se mueve desde la década de 1950. Primero, una sola empresa construía toda la máquina. Después, la máquina se dividió en capas, y cada capa se convirtió en un mercado. Ahora los mayores compradores de cómputo vuelven a juntar las capas, hasta llegar a sus propios chips. Saber dónde está el péndulo indica qué capas se ofrecerán y a qué precio.

## La primera oscilación: un proveedor, una máquina

Los primeros sistemas operativos los escribieron los clientes, no los proveedores. En 1956, General Motors Research y North American Aviation pusieron en producción un programa compartido para el IBM 704, llamado después GM-NAA I/O. Un **sistema operativo** es el programa que ejecuta otros programas y gestiona el hardware por ellos. Este hacía una tarea sencilla: arrancaba el siguiente trabajo por lotes en cuanto terminaba el anterior y daba a cada programa las mismas rutinas de entrada y salida. Solo con eso aumentó el número de trabajos que un 704 podía terminar cada día.

Dos ideas llegaron pronto. El **tiempo compartido**, en el que muchos usuarios en terminales comparten una computadora por turnos demasiado cortos para notarse, se mostró en el MIT en noviembre de 1961 con CTSS, el Compatible Time-Sharing System. Y el 7 de abril de 1964, IBM lanzó System/360, una familia de computadoras de distintos tamaños que podían ejecutar el mismo software. La propia historia de IBM la describe como la primera vez que el software escrito para una máquina podía ejecutarse en cualquier otra de la línea. OS/360, el sistema operativo previsto para la familia, se retrasó y superó su presupuesto, y Fred Brooks, que dirigió el trabajo, escribió *The Mythical Man-Month* (1975) sobre lo que aprendió.

La investigación fue más lejos. Multics, un proyecto conjunto del MIT, General Electric y Bell Labs, se describió en una serie de artículos en la Fall Joint Computer Conference de 1965. Aspiraba a ser un servicio público de cómputo, con funcionamiento continuo como la electricidad o el teléfono. VM/370 de IBM, anunciado el 2 de agosto de 1972, daba a cada usuario una **máquina virtual** completa: una copia del hardware en software capaz de ejecutar su propio sistema operativo. La nube todavía funciona sobre esa idea.

En esa época, el proveedor era dueño de cada capa, de los circuitos al compilador, y las vendía como un solo producto. La primera grieta apareció en 1969, cuando IBM empezó a poner precio a gran parte de su software por separado del hardware. El Computer History Museum resume el efecto en una línea: de la noche a la mañana, el software pasó de ser un regalo a ser un producto comercial competitivo.

## La oscilación horizontal: Unix, C y la PC

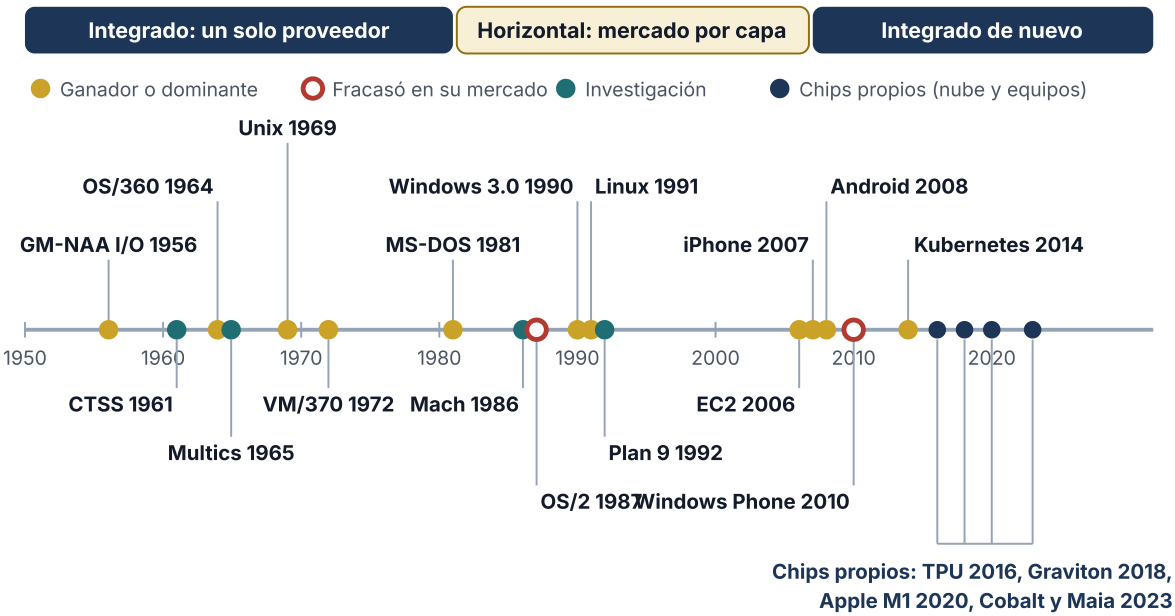
Bell Labs se retiró de Multics en abril de 1969, y ese año Ken Thompson y Dennis Ritchie empezaron Unix, como contó el capítulo 2. En el verano de 1973 lo reescribieron en C, un lenguaje que Ritchie diseñó para esa tarea. Su artículo en *Communications of the ACM* de julio de 1974 presentó el argumento en cifras: Unix funcionaba en hardware que costaba apenas 40 000 dólares, y el software principal del sistema había requerido menos de dos años-persona. Un sistema operativo escrito en un lenguaje portable podía pasar a una máquina nueva con un compilador nuevo, no con una reescritura. El sistema operativo ya no estaba atado al hardware de un solo fabricante.

La misma ruptura ocurrió en las máquinas pequeñas. En 1974, Gary Kildall mostró CP/M, el primer sistema operativo para computadoras personales con éxito comercial. Kildall y Glenn Ewing trasladaron después el código que tocaba el hardware a una capa separada llamada **BIOS** (Basic Input Output System), para que CP/M pudiera funcionar en computadoras con procesadores Intel de muchos fabricantes. Esa sola decisión de diseño hizo portable al sistema operativo.

IBM lanzó su Personal Computer el 12 de agosto de 1981 con PC DOS, licenciado por Microsoft en condiciones que le permitían vender el mismo sistema a otros fabricantes (el capítulo 10 cuenta cómo). Desde entonces, la PC fue una pila de mercados separados: Intel y otros vendían el procesador, Microsoft vendía el sistema operativo, decenas de fabricantes vendían la caja y miles de empresas vendían el software. Windows 1.0 llegó el 20 de noviembre de 1985, Windows 3.0 el 22 de mayo de 1990 y Windows 95 el 24 de agosto de 1995. La investigación siguió alimentando la pila: Mach pasó a formar parte de la base de macOS, y Linux dio al mundo horizontal un kernel que no pertenecía a ninguna empresa, como contó el capítulo 2.

FIGURA 9.1

## Setenta años en una línea, y el péndulo encima




Las fechas corresponden al primer uso en producción, la demostración pública, el anuncio o el lanzamiento, según las fuentes del capítulo. Las bandas son el péndulo: quién construye la pila, una sola empresa o un mercado de empresas.

### La oscilación de regreso: vuelve la integración

El 9 de enero de 2007, Apple presentó el iPhone. Apple diseñó el dispositivo y su sistema operativo, y desde 2008 revisó cada aplicación distribuida a través de su App Store. Las aplicaciones web, las aplicaciones internas distribuidas mediante el programa empresarial de Apple y, más tarde, las tiendas de aplicaciones alternativas en la UE son las excepciones. El 14 de julio de 2008, días después de abrir la App Store, Apple informó de más de 800 aplicaciones nativas y más de 10 millones de descargas en su primer fin de semana. El teléfono volvía a ser un solo producto, como lo había sido el mainframe.

El mismo movimiento llegó luego al escritorio y al centro de datos. El 22 de junio de 2020, Apple anunció que el Mac pasaría a Apple silicon, y el M1 llegó el 10 de noviembre de 2020, con el diseño de memoria unificada que describió el capítulo 1. Los proveedores de nube hicieron lo mismo a su propia escala:

- **Amazon Web Services.** A partir del tipo de instancia C5 en 2017, el Nitro System reemplazó la antigua pila de virtualización por un hipervisor pequeño basado en KVM y un conjunto de tarjetas Nitro dedicadas. El 26 de noviembre de 2018, en su conferencia re:Invent, AWS lanzó sus primeras instancias sobre Graviton, su propio procesador de servidor basado en Arm.
- **Google.** En mayo de 2016, Google dio a conocer la Tensor Processing Unit, un chip a medida para aprendizaje automático que ya funcionaba en sus centros de datos.

- **Microsoft.** El 15 de noviembre de 2023, en Ignite, Microsoft anunció Azure Maia, un acelerador de IA, y Azure Cobalt, una CPU basada en Arm. Su responsable de hardware expuso la estrategia sin rodeos: "somos una empresa de sistemas".

¿Por qué volver atrás? Porque, a suficiente escala, las costuras entre capas alquiladas cuestan más que ser dueño de ellas. Una empresa que compra millones de servidores puede pagar su propio chip y recuperar el costo en energía, densidad y control. Una empresa mediana no puede. Esa diferencia es el núcleo de la Parte III.

### Móvil y nube: un péndulo, dos mercados

El móvil se dividió como la PC, pero más rápido. Apple mantuvo el modelo integrado. Google tomó el horizontal: el 5 de noviembre de 2007 anunció Android junto con otras 33 empresas en la Open Handset Alliance, y el primer teléfono Android, el T-Mobile G1, se anunció en septiembre de 2008. El espacio de usuario de Android se licencia principalmente bajo Apache 2.0, una decisión que explica el capítulo 10. Un mercado, dos respuestas, y ambas sobrevivieron.

La nube se volvió horizontal en la parte de arriba mientras se integraba por debajo. El artículo del hipervisor Xen, presentado en SOSP en octubre de 2003, mostró muchos sistemas operativos compartiendo un servidor x86 con poca sobrecarga. Amazon anunció EC2, máquinas virtuales alquiladas por hora, en beta el 24 de agosto de 2006. Docker, presentado en público en PyCon en marzo de 2013, popularizó el **contenedor**: una aplicación empaquetada con sus archivos y ejecutada como un proceso aislado sobre un kernel compartido, más ligera que una máquina virtual completa. Google liberó como código abierto Kubernetes, un sistema que planifica contenedores en muchas máquinas, en 2014; su documentación dice que reúne más de 15 años de experiencia de Google ejecutando cargas de producción. Kubernetes 1.0 se publicó el 21 de julio de 2015, y Google cedió el proyecto a la nueva Cloud Native Computing Foundation ese mismo día.

Así que hoy se alquila una pila integrada en la base, donde un proveedor controla el chip, el hipervisor y las tarjetas, y más abierta arriba, donde Linux, los contenedores y Kubernetes están disponibles en todos los proveedores principales. Disponible no es lo mismo que portable: las bases de datos gestionadas, el almacenamiento, la red y los servicios de identidad de cada proveedor son distintos, y ahí es donde una migración se encarece.

> *Linux y Kubernetes funcionan en todas las nubes principales. Los servicios gestionados que los rodean no se mudan con ellos.*

| Dónde se ubica cada sistema | Windows | Linux | macOS |
|---|---|---|---|
| **En qué oscilación nació** | Horizontal: un sistema operativo licenciado a muchos fabricantes de PC | Horizontal: un kernel que no pertenece a ninguna empresa | Integrada: una empresa, Mach y BSD bajo su propio diseño |
| **Quién fabrica el hardware** | Muchos fabricantes de PC | Cualquiera; los proveedores de nube, sobre sus propios chips | Solo Apple |
| **Quién decide la capa de abajo** | Intel, AMD, Qualcomm y el fabricante de la PC | El fabricante del chip y la distribución | Apple, desde su propio silicio |
| **Dirección actual** | Atraído hacia la integración: el procesador de seguridad Pluton está dentro de la CPU y se actualiza con Windows Update (capítulo 1) | Ambas: proyecto original (upstream) abierto, integrado dentro de cada nube | Totalmente integrado, del chip a la App Store |

**QUÉ SIGNIFICA EL PÉNDULO PARA SU EMPRESA**

Una capa horizontal suele ofrecer más proveedores entre los que elegir; una integrada suele ofrecer un producto más pulido y menos voz sobre la hoja de ruta. Ninguna es mejor en general. Para cada capa clave, la pregunta útil es hacia dónde se mueve. Las cargas sobre Linux y Kubernetes conservan más libertad para elegir proveedor que las construidas sobre los chips propios y los servicios gestionados de un solo proveedor, aunque ningún cambio es gratis.

## Esta capa en la lista de materiales del píxel

La Parte II no añade filas nuevas. Añade una pregunta a cada fila que ya está en la lista: ¿esta capa se mueve hacia un solo proveedor o hacia un mercado, y qué pasa con su margen de decisión si sigue en esa dirección?

LENTE DE SEGURIDAD

**¿Quién puede cambiarlo?** En las eras integradas, una sola empresa puede cambiar toda la pila, lo que hace coherentes las correcciones y deja al cliente a merced de las prioridades de esa empresa. En las eras horizontales, cada capa tiene su propio dueño, y una sola falla puede necesitar a varios de ellos, como mostró el capítulo 1 con Spectre y Meltdown.

**¿Qué tan rápido se puede corregir?** Un proveedor integrado puede publicar una sola actualización que abarque el firmware del chip, el kernel y las aplicaciones. Una pila horizontal se corrige a la velocidad de su dueño más lento, a menudo un fabricante de dispositivos. La nube ofrece ambas cosas: el proveedor parchea sus hosts sin intervención del cliente, mientras el cliente sigue parcheando todo lo que está por encima del hipervisor.

**¿Qué cambia la IA aquí?** Los asistentes de IA reducen el costo de leer y portar código de una era anterior, como un trabajo por lotes de mainframe o una aplicación de escritorio de 32 bits, así que abandonar una era es más barato que antes. Cada era que se mantiene en funcionamiento sigue necesitando su propio responsable de parches, porque las interfaces antiguas y bien documentadas también son lectura fácil para las herramientas de los atacantes.

IMPLICACIONES PARA EL NEGOCIO

**Seguridad.** La empresa regulada debería ubicar cada sistema crítico en su era: un parque de mainframe, una flota Windows, Linux en la nube, teléfonos de un solo proveedor. Cada era tiene un responsable de parches distinto y un registro de evidencias distinto para los auditores, y los huecos están en las uniones entre eras.

**Costo.** El operador de una flota de dispositivos compra en el punto en que el péndulo es horizontal: computadoras estándar e imágenes estándar de Linux o Windows, muchos proveedores, competencia de precios. Debería resistirse a mover sus quioscos a la pila cerrada de un solo proveedor, salvo que ese proveedor también asuma por escrito el plazo de soporte.

**Valor.** El fabricante de software gana al construir por encima de la costura. Una aplicación que funciona en contenedores Linux y navegadores estándar puede moverse entre nubes y dispositivos a medida que oscila el péndulo. Conviene usar los chips propios de un proveedor donde la ganancia está medida y es grande, y mantener una ruta probada para salir de ellos.

**Primer paso:** para cada sistema crítico, anotar hacia dónde se mueven sus capas, hacia un solo proveedor o hacia un mercado, y qué se haría si siguieran en esa dirección.

CAPÍTULO DIEZ

# Por qué ganaron los ganadores




A los ingenieros les gusta creer que gana el mejor sistema. La historia dice otra cosa. Varios sistemas operativos que los expertos admiraban perdieron ante rivales con más aplicaciones, mejores acuerdos con los fabricantes de hardware y la promesa de que el software antiguo seguiría funcionando. Para un CTO esto no es una curiosidad: indica qué capas puede aspirar a poseer una empresa, y qué capa derrotó a empresas con mucho dinero que intentaron poseerla solas.

## Cuatro sistemas sólidos que perdieron

**OS/2.** IBM y Microsoft anunciaron OS/2 juntos el 2 de abril de 1987, y la versión 1.0 salió en diciembre de 1987: un sistema de 16 bits, multitarea y en modo protegido, con hilos y memoria virtual, muy por delante de DOS. Luego Microsoft lanzó Windows 3.0 el 22 de mayo de 1990. Windows funcionaba sobre el DOS que los fabricantes de PC ya incluían, con las aplicaciones DOS que los usuarios ya tenían, y Microsoft afirma que Windows 3.0 y 3.1 vendieron 10 millones de copias en sus dos primeros años. IBM mantuvo OS/2 durante años; después lo retiró de la venta el 23 de diciembre de 2005 y terminó el soporte estándar el 31 de diciembre de 2006.

**BeOS.** Be Inc., fundada por el ex ejecutivo de Apple Jean-Louis Gassée, construyó un sistema operativo rápido y orientado al multimedia para la década de 1990. Cuando Apple buscó un nuevo sistema operativo en 1996, consideró a Be y eligió NeXT, anunciado el 20 de diciembre de 1996. Be intentó entonces llegar a los compradores de PC como segundo sistema junto a Windows. En su demanda antimonopolio de 2002, Be alegó que las condiciones de licencia de Microsoft impedían a los fabricantes de PC incluir más de un sistema operativo preinstalado, y que un acuerdo con Hitachi se cayó por esa razón. Be vendió sus activos a Palm por 11 millones de dólares en 2001. En septiembre de 2003, Microsoft pagó a Be 23,25 millones de dólares para cerrar la demanda, sin admitir culpa. Sean cuales sean los méritos, la lección trata de la distribución: un sistema que no puede llegar a la máquina en la fábrica tiene que ganarse a cada usuario uno por uno.

**Plan 9.** El equipo de Bell Labs que creó Unix empezó Plan 9 a finales de la década de 1980, para construir un sistema operativo para un mundo en red en el que cada recurso parece un archivo. Su primera edición, en 1992, se entregó solo a universidades. La segunda, en 1995, se vendía por 350 dólares. No pasó a ser gratuita por Internet hasta junio de 2000, y está bajo la licencia MIT desde marzo de 2021. Para entonces el mercado ya había elegido Unix y Linux. Aun así, las ideas de Plan 9 se difundieron. UTF-8, la codificación de texto de la web, la diseñaron Ken Thompson y Rob Pike en

septiembre de 1992 y se puso en uso por primera vez en Plan 9. Su protocolo de archivos, 9P, es la forma en que Windows accede a los archivos dentro de una distribución Linux bajo WSL, desde Windows 10 versión 1903.

**Windows Phone.** Microsoft presentó Windows Phone 7 el 15 de febrero de 2010, más de tres años después del anuncio del iPhone. En febrero de 2011, Nokia lo convirtió en su plataforma principal de smartphones, y el 25 de abril de 2014 Microsoft completó la compra del negocio de dispositivos de Nokia. El 8 de julio de 2015, Microsoft anunció que registraría un cargo de unos 7600 millones de dólares contra ese negocio, y Satya Nadella describió un cambio: de hacer crecer un negocio de teléfonos independiente a hacer crecer el ecosistema Windows. La tabla actual de fin de soporte de Microsoft ("What does it mean if Windows isn't supported?") indica que Windows 10 Mobile terminó el 14 de enero de 2020; sus preguntas frecuentes de 2019 sobre el fin de soporte de Windows 10 Mobile daban el 10 de diciembre de 2019, cuando la mayoría de los usuarios dejó de recibir actualizaciones. Microsoft tenía el dinero, las herramientas para desarrolladores y un socio grande. Llegó cuando las dos tiendas de aplicaciones que importaban ya estaban llenas.

## Cuatro ganadores, y lo que tenían

**Windows** ganó por distribución y compatibilidad. La licencia no exclusiva de DOS, que se cuenta en el recuadro de abajo, hizo que cada nuevo clon de PC agrandara el mercado de Microsoft. La compatibilidad se volvió una promesa de producto: hoy WOW64 sigue ejecutando aplicaciones Windows de 32 bits en Windows de 64 bits, sin que el usuario tenga que activar nada. La propia documentación de Microsoft también muestra el límite: Windows de 64 bits no ejecuta aplicaciones de 16 bits. Hasta la mejor promesa de compatibilidad termina en algún punto.

**Linux** ganó en servidores, supercomputadoras y dispositivos embebidos, y como kernel dentro de Android, aunque no en el escritorio de consumo. Ganó por su apertura y por quién lo pagó. Linus Torvalds pasó el kernel a la GNU General Public License a principios de 1992, de modo que cualquiera podía distribuirlo, y quien distribuyera una versión modificada debía entregar u ofrecer el código fuente completo correspondiente a quienes la recibieran. Los cambios que se quedan dentro de una empresa no tienen esa obligación. El 12 de diciembre de 2000, IBM dijo que invertiría 1000 millones de dólares en Linux en 2001. Grandes empresas que financian un kernel que no les pertenece: ese es el patrón que describió el capítulo 2. Desde la lista TOP500 de noviembre de 2017, cada una de las 500 supercomputadoras más rápidas del mundo usa Linux.

**Android** ganó por los acuerdos de hardware. Google lo anunció el 5 de noviembre de 2007 junto con otras 33 empresas, entre ellas fabricantes de teléfonos, fabricantes de chips y operadores. Su documentación explica la elección de licencia en términos de negocio: Google prefiere Apache 2.0 para el espacio de usuario de Android porque los fabricantes de dispositivos lo entregan como una imagen de sistema fija, algo que encaja con una licencia permisiva. En mayo de 2021, Google dijo que Android funcionaba en más de 3000 millones de dispositivos activos. Los acuerdos también atrajeron a los reguladores: el 18 de julio de 2018, la Comisión Europea multó a Google con 4340 millones de euros por vincular sus aplicaciones a la licencia de Android. En septiembre de 2022, el Tribunal General de la UE confirmó en gran parte la decisión y fijó la multa en 4125 millones de euros, y el 2 de julio de 2026 el Tribunal de Justicia desestimó la última apelación de Google.

**iOS** ganó por los desarrolladores, desde dentro de un sistema cerrado. El 6 de marzo de 2008, Apple publicó un kit de desarrollo de software gratuito y fijó las condiciones: los desarrolladores ponen sus propios precios y se quedan con el 70 % de las ventas, y las aplicaciones se venden solo a través de la App Store, tras la aprobación de Apple. En los primeros días de la tienda, Apple informó de más de 800 aplicaciones nativas y más de 10 millones de descargas. Apple se quedó con el hardware y la tienda, y dejó la capa de aplicaciones a todos los demás.

FIGURA 10.1

## Los factores detrás de cada resultado

| | GANADORES | | | | PERDIERON CON VIRTUDES | | | |
|---|---|---|---|---|---|---|---|---|
| | Windows | Linux | Android | iOS | OS/2 | BeOS | Plan 9 | Win Phone |
| Desarrolladores y apps | ● | ● | ● | ● | ○ | ○ | ○ | ○ |
| Distribución | ● | ● | ● | ● | ◐ | ○ | ○ | ◐ |
| Acuerdos de hardware | ● | ◐ | ● | ● | ◐ | ○ | ○ | ◐ |
| Apertura | ◐ | ● | ● | ○ | ◐ | ◐ | ◐ | ◐ |
| Momento | ● | ● | ● | ● | ○ | ○ | ○ | ○ |
| Compatibilidad | ● | ● | ◐ | ◐ | ◐ | ○ | ○ | ◐ |

● Fortaleza clara ◐ Mixto o parcial ○ Debilidad en el momento decisivo

Es la lectura del autor sobre los casos de este capítulo, no una medición. Cada ganador fue fuerte en al menos cuatro de estos seis factores. Los perdedores tenían fortalezas reales que la matriz no muestra: OS/2 tenía un diseño más avanzado y el canal de ventas de IBM, Windows Phone tenía el dinero y las herramientas de Microsoft. iOS ganó siendo cerrado, así que la apertura por sí sola no decide nada.

## Las lecciones, factor por factor

| Factor | Qué significa | Ejemplo que lo muestra |
|---|---|---|
| Desarrolladores y aplicaciones | Los usuarios compran un sistema por lo que funciona en él | El SDK de Apple de 2008 y el 70 % de los ingresos para el desarrollador; la tienda tardía de Windows Phone |
| Distribución | Estar en la máquina cuando se vende | MS-DOS licenciado a todos los fabricantes de PC; BeOS excluido de las instalaciones de fábrica, según su propia versión |
| Acuerdos de hardware | Fabricantes y operadores que ganan dinero al distribuirlo | Android nació con 33 socios; la decisión de la UE de 2018 muestra que esos mismos acuerdos conllevan riesgo legal |
| Apertura | Otros pueden distribuirlo, cambiarlo y ampliarlo sin pedir permiso | Linux bajo la GPL desde 1992; Plan 9 gratuito solo desde 2000 |
| Momento | Llegar antes de que el mercado se asiente | Windows 3.0 en 1990 frente a OS/2; Windows Phone 7 en 2010 frente a dos tiendas de aplicaciones ya llenas |
| Compatibilidad | El software de ayer sigue funcionando | Windows sobre DOS, después WOW64 para aplicaciones de 32 bits; el límite: no hay aplicaciones de 16 bits en Windows de 64 bits |

**CÓMO LLEGÓ A SER ASÍ**

En 1980, mientras construía su Personal Computer, IBM intentó licenciar CP/M de Digital Research. Los relatos sobre por qué fracasaron las conversaciones difieren. La versión del Computer History Museum es que terminaron en un punto muerto por las condiciones económicas: IBM quería comprar el sistema de una vez y Digital Research buscaba una regalía por cada copia, mientras Gary Kildall creía que las dos partes básicamente habían llegado a un acuerdo. Lo que no se discute es lo que vino después. Microsoft compró un sistema operativo a Seattle Computer Products, lo licenció a IBM como PC DOS de forma no exclusiva y se reservó el derecho de venderlo a otros como MS-DOS. PC DOS tenía un precio de lista de 40 dólares, frente a los 240 de CP/M, y la mayoría de los compradores eligió el sistema más barato. El precio y la licencia, más que el código, dieron forma a los siguientes veinte años del escritorio.

### El escritorio actual, con los mismos factores

| Factor | Windows | Linux | macOS |
|---|---|---|---|
| **Quién lo instala en la máquina** | Los fabricantes de PC, en fábrica | Algunos fabricantes de PC; sobre todo lo instala TI, y es la opción habitual en servidores en la nube | Apple, en cada Mac |
| **De dónde vienen las aplicaciones** | Descargas directas, instaladores y la Microsoft Store | Repositorios de la distribución, Flatpak, Snap y contenedores | La Mac App Store y descargas directas |
| **Promesa de compatibilidad** | Fuerte: décadas de aplicaciones Win32, las de 32 bits mediante WOW64 | Fuerte en la interfaz del kernel hacia el espacio de usuario; más débil en las bibliotecas de escritorio | Apple sigue adelante cuando cambia de chips o de API |
| **Qué deja en manos de otros** | El hardware y la mayoría de las aplicaciones | Casi todo: cualquiera puede distribuirlo | Solo la capa de aplicaciones |

### La lección para los dueños

Conviene observar qué poseía cada ganador y qué dejaba poseer a otros. Microsoft poseía el sistema operativo y dejó que toda una industria construyera el hardware y las aplicaciones. Apple posee el dispositivo y la tienda, y deja que otras empresas posean las aplicaciones. Google posee la dirección de Android y sus servicios, y deja que muchos fabricantes construyan los teléfonos. Linux no pertenece a nadie, y por eso todos pueden construir sobre él. Cada ganador poseyó la parte que lo hacía distinto y cultivó a las empresas que construyen sobre él, mientras controlaba el acceso a la plataforma.

Los perdedores tuvieron que construir un ecosistema frente a un rival que ya tenía uno. Be tuvo que convencer a usuarios y desarrolladores uno por uno sin la instalación de fábrica, como alegó en la demanda descrita arriba. IBM tuvo que competir con la base de DOS y Windows que había ayudado a crear. Plan 9 no estuvo disponible libremente hasta que el mercado ya había elegido. Microsoft compró el negocio de dispositivos de Nokia para asegurar hardware para Windows Phone, y registró una pérdida contable de ese negocio unos quince meses después. El momento, las condiciones de licencia y la conducta de un rival influyeron, así que ningún factor por sí solo explica estos resultados. Lo que los casos tienen en común es que un ecosistema lo construyen miles de otras empresas que deciden, cada una por sus propias razones, sumarse, y en estos casos ni el dinero ni la buena ingeniería compraron esa decisión.

Para una empresa mediana, la regla es directa. Poseer lo que la hace distinta: su producto, sus datos, quizá una capa donde una corrección rápida vale dinero de verdad. Controlar su ruta de publicación: alquilar los servidores de compilación y construir solo las piezas críticas para la confianza, su cadena de publicación y sus claves. Alquilar las capas de ecosistema: el sistema operativo, las tiendas de aplicaciones, la base de controladores, el motor de navegador.

> *OS/2 tenía el mejor kernel de su época. Windows tenía la base instalada.*

## Esta capa en la lista de materiales del píxel

Este capítulo no añade filas nuevas. Marca tres tipos de fila en su lista: las capas de ecosistema que se alquilan, como las tiendas de aplicaciones, las capas de compatibilidad y las imágenes de sistema operativo de los fabricantes de dispositivos; la fila que se construye, su producto; y la fila que se controla, su ruta de publicación.

LENTE DE SEGURIDAD

**¿Quién puede cambiarlo?** El dueño del ecosistema. Una tienda puede retirar su aplicación, cambiar sus reglas de revisión o cambiar lo que una aplicación puede tocar, y una capa de compatibilidad puede desaparecer. Estas capas se alquilan, así que se heredan las decisiones de su dueño.

**¿Qué tan rápido se puede corregir?** Un dueño integrado puede enviar una corrección a todos los dispositivos que controla. En un ecosistema horizontal, la corrección espera a cada fabricante, como ocurre con las imágenes de sistema de Android que llegan a los teléfonos a través de sus fabricantes. Las capas de compatibilidad tardan en corregirse porque deben conservar el comportamiento antiguo, incluidos algunos errores antiguos.

**¿Qué cambia la IA aquí?** La IA reduce el costo de portar una aplicación a una segunda plataforma, así que alquilar dos ecosistemas en lugar de uno es más asequible que antes. No cambia el poder del dueño de la tienda: las reglas de revisión y el retiro de API siguen llegando al ritmo del dueño.

IMPLICACIONES PARA EL NEGOCIO

**Seguridad.** La empresa regulada debería tratar cada tienda de aplicaciones y cada fabricante de dispositivos como un proveedor con su propio calendario de parches. Pedir por escrito la ventana de actualizaciones antes de comprar un modelo de teléfono o tableta, y retirar los modelos cuando esa ventana termine.

**Costo.** El operador de una flota de dispositivos no debería construir su propio sistema operativo para quioscos o terminales de punto de venta para escapar de los costos de licencia. La historia de OS/2, BeOS y Windows Phone dice que el costo oculto son los controladores, las herramientas y las aplicaciones que nadie más escribirá en su lugar. Conviene alquilar un sistema operativo dominante y endurecer su configuración.

**Valor.** El valor del fabricante de software está en su producto, no en la plataforma que tiene debajo. Conviene publicar en las plataformas que sus clientes ya usan, a través de sus tiendas e instaladores, e invertir el esfuerzo ahorrado en las funciones que lo hacen distinto.

**Próximo ciclo de planificación:** listar cada capa que se intenta poseer; para cada una que sea una capa de ecosistema, anotar quién la controla ya y cuánto costaría alquilarla.

CAPÍTULO ONCE

# Las costuras, en números




Cada costura de la pila envía una factura, pero pocas empresas suman las facturas. Una costura es el punto donde el código de un dueño entrega trabajo al código de otro dueño. Una costura puede traer una copia que almacenar, un canal que parchear o una demora antes de que llegue la corrección, y las costosas traen las tres cosas. La Parte I nombró a los dueños capa por capa. Este capítulo cuenta lo que cuestan las costuras entre ellos, con números que se pueden verificar y una hoja de trabajo que cada empresa puede completar con sus propios datos.

## Cuatro cosas que se pueden contar

"Sobrecosto de las costuras" es el nombre que este libro da al costo que paga una empresa porque su pila tiene muchos dueños. Se esconde dentro de otros presupuestos: memoria en las computadoras portátiles, horas de pruebas de parches, días de exposición después de una corrección de seguridad. Se vuelve manejable cuando se divide en cuatro cantidades que un equipo puede contar:

- **Copias.** Cuántas copias separadas del mismo runtime o biblioteca hay en una máquina.
- **Canales.** Cuántas rutas de actualización entregan código a esa máquina, cada una con su propio dueño y calendario.
- **Retraso.** Cuántos días pasan entre una corrección upstream y el momento en que su flota la está ejecutando.
- **Exposición.** A cuántas de esas copias puede llegar una entrada no confiable, como una página web, un correo electrónico o una imagen.

Los tres sistemas dan forma a estos números de manera distinta. La tabla compara las piezas que cada uno ofrece a una aplicación para compartir, y de dónde salen las copias cuando no se comparten.

| Pregunta | Windows | Linux (distribución tipo Debian) | macOS |
|---|---|---|---|
| **Motor web compartido que una aplicación puede tomar prestado** | WebView2 Evergreen, parcheado por Microsoft | WebKitGTK o Qt WebEngine, parcheados por la distribución | WKWebView, parcheado por Apple |
| **De dónde salen las copias adicionales del motor** | Cada aplicación Electron y cada aplicación con WebView2 Fixed Version | Cada aplicación Electron y los navegadores que empaquetan (bundle) sus propias bibliotecas | Cada aplicación Electron y cada navegador de terceros |
| **Canales de actualización que la plataforma ofrece a las aplicaciones** | Microsoft Store, WinGet o el actualizador propio de la aplicación | Los repositorios de la distribución, Flatpak, Snap | Mac App Store o el actualizador propio de la aplicación |
| **Quién corrige una falla en una biblioteca compartida** | Microsoft, para las bibliotecas del sistema y el runtime WebView2 Evergreen; cada producto, para un runtime Fixed Version u otra copia privada | La distribución, una vez por cada paquete fuente que lleva una copia | Apple para las copias del sistema; cada proveedor para las suyas |

## Runtimes duplicados, medidos

La propia documentación de Electron dice que integra "el motor de navegador Chromium y el runtime Node.js en su binario". Por lo tanto, cada aplicación Electron lleva un motor de navegador completo, como explicó el capítulo 7. La pregunta para un CTO es qué tan grande es esa copia.

El proyecto Electron publica sus compilaciones del runtime en GitHub, con tamaños de archivo exactos. Para Electron 26.2.1, la versión que corrigió la falla de libwebp en la línea 26, los archivos publicados pesan 101,8 MB para Windows x64, 97,5 MB para Linux x64 y 90,3 MB para macOS en Apple silicon. Descomprimida, la compilación de Windows ocupa 248,6 MB en disco; solo el ejecutable principal pesa 165,9 MB. Estas cifras corresponden al runtime por sí solo, antes de que el proveedor de la aplicación añada una sola línea de su propio código. Describen la versión en uso durante el incidente de 2023 que se cuenta más abajo, y los tamaños cambian con cada versión: el archivo para Windows x64 de Electron 45.3.3, publicado el 14 de noviembre de 2026, pesa 158,3 MB, es decir, más, no menos. (Los tamaños de archivo provienen de los registros de versiones de GitHub; el tamaño descomprimido de 26.2.1 se midió el 17 de noviembre de 2026.)

**HOJA DE TRABAJO A (UNA ILUSTRACIÓN): COPIAS DEL MOTOR EN UNA COMPUTADORA PORTÁTIL**

**Supuestos:** una portátil ejecuta cinco aplicaciones de escritorio construidas sobre Electron, como chat, reuniones, un gestor de contraseñas, una herramienta de notas y su propio producto, cada una sobre un runtime del tamaño aproximado de Electron 26. Esta hoja de trabajo es independiente; las hojas B y C de más abajo usan otra imagen de portátil.

**Resultado:** unos 5 × 248,6 MB, es decir, cerca de 1,2 GB de disco, guardan cinco copias de casi el mismo motor. Cinco copias también significan cinco rutas de parches, cada una al ritmo de su dueño, y cinco consumos de memoria cuando las aplicaciones se ejecutan a la vez. El uso de memoria depende mucho de cada aplicación, así que conviene medirlo en su propia imagen en lugar de tomar prestado un número de un foro.

El disco es barato. Las copias importan porque cada una es algo distinto que parchear, y porque cada una procesa (parsing) contenido que viene de fuera de la empresa.

## Retraso de parches: una falla, día por día

Esta es la historia completa de CVE-2023-4863, la falla a la que remiten otros capítulos. **WebP** es un formato de imagen, y **libwebp** es la biblioteca abierta que lo decodifica. El 6 de septiembre de 2023, el equipo de ingeniería de seguridad de Apple y el Citizen Lab comunicaron un error a Chrome: un desbordamiento de búfer en el heap al manejar WebP, un error de memoria en el que una imagen manipulada escribe más allá del final de un búfer. La corrección llegó a upstream al día siguiente, en el código que construye las tablas de Huffman para imágenes sin pérdida. El 11 de septiembre, la versión estable de Chrome la publicó como corrección crítica, y Google dijo que existía un exploit activo: los atacantes ya lo estaban usando.

Al principio, el registro público lo describía como un error "en WebP en Google Chrome". Pero la falla estaba en libwebp, y libwebp está dentro de otros navegadores, de aplicaciones Electron que empaquetan Chromium, de bibliotecas de imágenes y de software de servidor. Cada uno de esos productos llevaba su propia copia, y cada copia tenía un dueño que debía publicar una corrección. Estos son los hechos en números. El día 0 es el 6 de septiembre de 2023, la fecha del informe en la nota de versión de Chrome. Las fechas son de publicación en UTC: Electron etiquetó sus cuatro versiones corregidas el 12 de septiembre y las publicó en unas ocho horas, a uno y otro lado de la medianoche UTC.

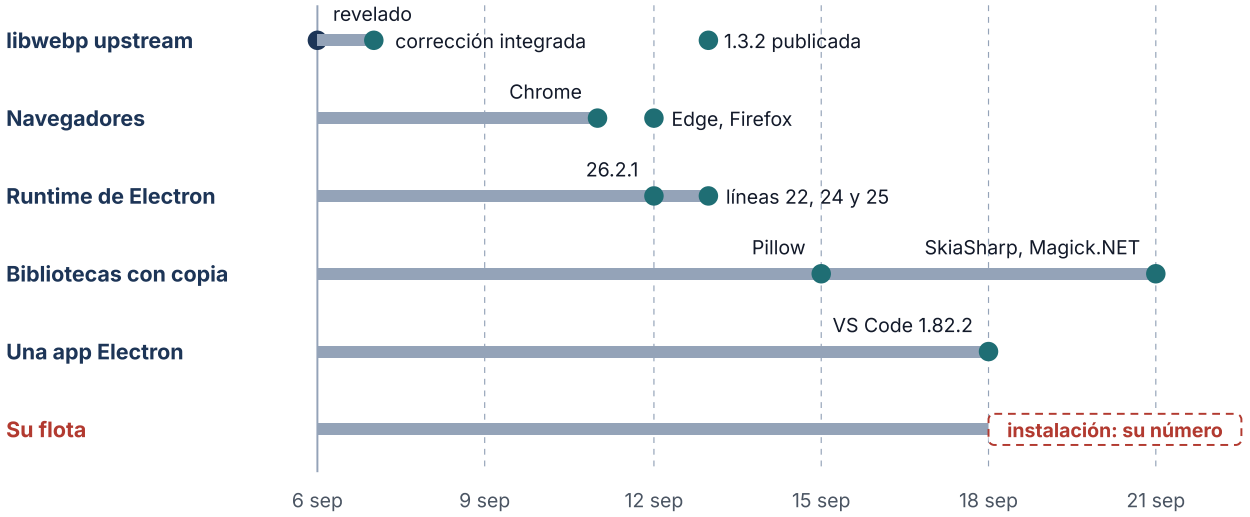
| Día | Fecha (2023) | Dueño | Hecho |
|---|---|---|---|
| 0 | 6 sep | Quienes lo revelaron a Chrome | Error revelado |
| 1 | 7 sep | Proyecto libwebp | Corrección integrada (commit) en upstream ("Fix OOB write in BuildHuffmanTable") |
| 5 | 11 sep | Google | Chrome estable 116.0.5845.187/.188 publica la corrección; exploit activo conocido |
| 6 | 12 sep | Microsoft, Mozilla, Electron | Edge 116.0.1938.81; Firefox 117.0.1 y versiones ESR; Electron 26.2.1 |
| 7 | 13 sep | Electron, libwebp, CISA | Electron 22.3.24, 24.8.3 y 25.8.1; se publica libwebp 1.3.2; CISA añade el CVE a su lista de vulnerabilidades explotadas |
| 9 | 15 sep | Biblioteca de imágenes de Python | Pillow 10.0.1 en PyPI |
| 12 | 18 sep | Un proveedor de aplicaciones Electron | Se publica Visual Studio Code 1.82.2, que pasa a Electron 25.8.1 |
| 15 | 21 sep | Proveedores de bibliotecas .NET | SkiaSharp 2.88.6 y Magick.NET 13.3.0 publicados en NuGet |
| ? | Después | Su equipo de TI y sus usuarios | El último dispositivo de su flota instala cada una de estas versiones |

Destacan tres hechos. Primero, upstream fue el primero con la corrección, pero no con la versión: la corrección se integró en libwebp el día 1, Chrome la publicó el día 5 y la versión oficial de upstream, libwebp 1.3.2, apareció el día 7. Segundo, las versiones forman un árbol con ramas, no una sola cadena. Los proveedores de navegadores y bibliotecas parchearon en paralelo a partir de la corrección upstream, pero cuando un producto dependía de otro, esperaba: una aplicación construida sobre Electron 25 no podía usar una versión oficial parcheada de Electron 25 antes de que apareciera Electron 25.8.1 el día 7, salvo que retroportara la corrección (backport) y compilara Electron por su cuenta, y cada dependencia de ese tipo añadía de uno a varios días. Tercero, la última fila es la que ningún proveedor publica. Es la fila que la empresa controla, y la que puede medir.

La forma en que una aplicación integraba su motor decidía en qué rama quedaba. Una aplicación sobre el runtime WebView2 Evergreen recibió la corrección del motor cuando Microsoft actualizó el runtime compartido, tras un reinicio. Cada aplicación Electron tuvo que tomar la nueva versión de Electron, recompilar, probar, firmar y publicar su propia actualización, y luego esperar a que los usuarios la instalaran. Una aplicación sobre un runtime WebView2 Fixed Version estaba en la misma situación que Electron, porque lleva su propia copia. Una aplicación WKWebView en macOS se apoya en WebKit de Apple y en los decodificadores de imágenes propios de Apple, no en Chromium, así que quedaba fuera de este árbol; las correcciones de su motor llegan con las actualizaciones del sistema de Apple.

FIGURA 11.1

### Retraso de parches a través de las costuras: CVE-2023-4863, septiembre de 2023




Barra gris: días que una copia esperó a su dueño. Punto: ese dueño publicó.

Las versiones se ramifican: las filas que empaquetan código de otro dueño, como una aplicación Electron, esperan a ese dueño, mientras otras parchean en paralelo a partir de la corrección upstream. El registro público termina en la versión del proveedor de la aplicación; el tiempo hasta que su último dispositivo la instala le toca medirlo a la empresa. Fechas tomadas de los registros de versiones de Chrome, Microsoft, Mozilla, Electron, libwebp, PyPI, NuGet y Visual Studio Code.

## Canales de actualización en una computadora de empresa

Un canal de actualización es cualquier ruta por la que llega código nuevo a una máquina: el actualizador del sistema operativo, un servicio en segundo plano de un proveedor, una tienda de aplicaciones o un actualizador integrado en una aplicación. Cada canal tiene un dueño, un calendario, un ciclo de pruebas y una forma de fallar. Nadie publica un conteo estándar, así que aquí hay uno resuelto.

**HOJA DE TRABAJO B (UNA ILUSTRACIÓN): CANALES DE ACTUALIZACIÓN EN LA PORTÁTIL DE UNA EMPRESA REGULADA**

**Supuestos:** una portátil con Windows 11 en una empresa regulada, gestionada por TI, con un conjunto típico de herramientas de oficina. El conteo es ilustrativo; su imagen será distinta.

- Windows Update para el sistema operativo, más los controladores que entrega: **1**
- La herramienta de firmware y controladores del fabricante de la computadora: **1**
- El actualizador de Microsoft para Edge y el runtime WebView2 Evergreen: **1**
- Un segundo navegador con su propio actualizador: **1**
- El servicio de actualización propio de la suite ofimática: **1**
- La Microsoft Store, para las aplicaciones de serie que sirve: **1**
- Tres aplicaciones Electron, cada una con un actualizador integrado: **3**
- El agente de seguridad de los equipos (endpoints), que descarga su propio contenido: **1**
- El actualizador de su propio producto: **1**

**Resultado:** unos 11 canales, cada uno de un equipo o empresa distinta. Solo uno de ellos, el último, es suyo. Al menos tres llevan un motor de navegador, y uno de ellos, el agente de seguridad, puede llegar al kernel, como describió el capítulo 2.

El conteo importa por dos razones. Cada canal es un punto de entrada a la cadena de suministro, así que pertenece al modelo de amenazas. Y cada canal puede publicar una regresión en su propio día, así que, cuando una flota se rompe, la primera pregunta es cuál de los once dueños se movió.

## El reloj se aceleró y la cola se alargó

El retraso de parches solo importa si los atacantes se mueven más rápido que ese retraso. Lo hacen, y la brecha lleva años creciendo. Este es el relato completo de una cifra que citan capítulos posteriores. Mandiant, parte de Google Cloud, mide el tiempo hasta la explotación (TTE, time-to-exploit), que define como "el tiempo promedio que se tarda en explotar una vulnerabilidad antes o después de que se publique un parche". Su análisis de octubre de 2024 sobre 138 vulnerabilidades divulgadas en 2023 y explotadas activamente encontró:

| Período | Tiempo promedio hasta la explotación |
|---|---|
| 2018 a 2019 | 63 días |
| 2020 a principios de 2021 | 44 días |
| 2021 a 2022 | 32 días |
| 2023 | 5 días (sin 15 valores atípicos; 47 días con ellos) |

El promedio de 2023 es sensible a unos pocos casos extremos: Mandiant quitó 15 valores atípicos estadísticos para llegar a 5 días, y con ellos el promedio es de 47 días. Conviene leerlo como una dirección, no como una constante. El promedio mezcla dos tipos de falla: 97 de las 138, cerca del 70 %,

fueron zero-days, explotadas antes de que hubiera un parche, y 41 fueron n-days, explotadas por primera vez después de que existiera un parche; los 15 valores atípicos quitados fueron 13 zero-days y 2 n-days. Mandiant considera conservadoras sus cifras, porque usa la primera fecha documentada de explotación, y la explotación real casi seguro es anterior. La curva ya bajaba mucho antes de que las herramientas de IA generativa estuvieran ampliamente disponibles, así que estos datos no muestran que la IA la cause o la acelere; hoy ambos lados usan herramientas de IA, como cuenta el capítulo 13. Conviene comparar el promedio de 5 días con la tabla anterior: una aplicación Electron de uso muy extendido publicó la corrección de libwebp 12 días después del informe y 7 días después de la corrección pública de Chrome. Y en un zero-day, el reloj del atacante empieza antes del día 0, porque el exploit existe antes de que alguien revele la falla: para CVE-2023-4863, Google dijo que ya existía un exploit activo cuando Chrome publicó su corrección.

La cola de fallas conocidas creció al mismo tiempo. Un conteo independiente de la lista pública de CVE encontró unos 40 000 registros publicados en 2024 (40 009 en el momento del conteo) y unos 48 000 en 2025 (48 185), cada uno un año récord. Los totales exactos varían entre fuentes, según cómo se traten los registros rechazados. Parte del salto es de proceso, no de peligro nuevo: el proyecto del kernel Linux se convirtió en su propia CVE Numbering Authority en febrero de 2024 y asigna un CVE a cualquier corrección de error que considere que podría importar. El mismo conteo atribuye al kernel 4325 CVE en 2024, cerca del 11 % del año. Más registros significan más triaje para cada dueño de la figura 11.1, sea o no peligroso cada registro.

> *Hoy a los atacantes les bastan días. Su ruta de parches es una cadena de dueños, y su longitud es el número que la empresa puede cambiar.*

### Exposición de los decodificadores: un número para un analizador compartido

Un decodificador es el código que convierte un archivo comprimido, como una imagen, una fuente o un video, en algo que un programa puede dibujar. Los decodificadores procesan formatos complejos que vienen de desconocidos, lo que los convierte en un blanco favorito. Ningún decodificador está en todas las rutas hacia el píxel. Pero un decodificador puede estar en muchas rutas a la vez, bajo muchos dueños, y eso es lo que dejó al descubierto CVE-2023-4863.

Este libro define la **exposición de los decodificadores** (reachability) como un conjunto de conteos que se pueden calcular para cualquier analizador en cualquier máquina. Miden cosas distintas, así que conviene mantenerlos separados:

- **Copias físicas (C):** copias separadas del código del decodificador en disco. Los runtimes compartidos y los enlaces duros hacen que este número sea menor que el siguiente.
- **Instalaciones (I):** instalaciones parcheadas de forma independiente que contienen el decodificador. Cada una necesita su propia actualización.
- **Dueños (O):** dueños distintos que deben publicar cada uno una corrección antes de que todas las instalaciones estén a salvo.
- **Instalaciones expuestas (R):** las instalaciones a las que una entrada no confiable, como una página web, un adjunto de chat o la vista previa de un correo, puede llegar de verdad.

- **La ventana de exposición, en dos números.** El **retraso máximo (L)** son los días transcurridos desde la primera corrección pública hasta que se corrige la última instalación expuesta; es el número que se compara con el reloj del atacante. Los **días-copia (E)** suman, para cada instalación expuesta, los días que siguió sin corregir; miden el volumen de exposición, la exposición total sin corregir que crean las costuras, no el trabajo de parchear ni cuánto tiempo tuvo un atacante.

Los registros públicos ya dan conteos parciales para libwebp en 2023. El aviso de GitHub lista 13 paquetes afectados en cinco ecosistemas: npm, PyPI, NuGet, Go y Cargo de Rust. El rastreador de seguridad de Debian registra correcciones separadas en cinco paquetes fuente: `chromium`, `firefox`, `firefox-esr`, `libwebp` y `thunderbird`, porque varios programas llevaban su propia copia en lugar de usar la biblioteca del sistema. Un solo aviso de Mozilla cubrió cinco líneas de producto: Firefox, dos líneas de Firefox ESR y dos líneas de Thunderbird.

Contar las copias es el primer paso; el segundo es saber a cuáles se puede llegar. Conviene pedir a cada proveedor una declaración **VEX** (Vulnerability Exploitability eXchange) junto con su SBOM: una instalación que su proveedor marca de forma creíble como "no afectada" sale de R (capítulo 8).

#### HOJA DE TRABAJO C (UNA ILUSTRACIÓN CON SUPUESTOS EXPLÍCITOS): EXPOSICIÓN DE LOS DECODIFICADORES EN UNA PORTÁTIL

**Supuestos:** la portátil de la hoja de trabajo B, en septiembre de 2023. Instalaciones que leen imágenes web: Chrome, Edge, el runtime WebView2 Evergreen, tres aplicaciones Electron y su propio producto, que integra Electron. Edge y el runtime WebView2 son productos separados con instalaciones separadas; la documentación de Microsoft dice que sus binarios se enlazan con enlaces duros en los sistemas que cumplen los requisitos cuando ambos están en la misma versión, así que pueden compartir una sola copia física. La exposición se cuenta desde la corrección de Chrome del 11 de septiembre. Chrome y Edge publican en las fechas de la tabla anterior; se supone que el runtime WebView2 publica el mismo día que Edge; se supone que las cuatro aplicaciones basadas en Electron publican 10 días después de Chrome; se supone que los usuarios instalan dentro de los 3 días siguientes a cada versión. Estos retrasos son supuestos, no mediciones.

**Resultado:** C = 6 copias físicas (7 si Edge y WebView2 no están enlazados con enlaces duros), I = 7 instalaciones, O = 6 dueños (Google, Microsoft, tres proveedores de aplicaciones y su propio equipo) y R = 7. El retraso máximo L es de 13 días: las aplicaciones Electron, 10 días para publicar más 3 para instalar. Los días-copia E son 3 (Chrome) + 4 (Edge) + 4 (WebView2) + 4 × 13 (las aplicaciones Electron) = 63.

Conviene comparar L, no E, con el atacante: 13 días transcurridos frente al promedio de 5 días de Mandiant, y en un zero-day como este el atacante empezó antes del día 0. E mide el volumen de exposición, no el trabajo: siete instalaciones sin corregir durante un día son siete días-copia, no una ventana de siete días, y un despliegue lento suma días-copia sin sumar esfuerzo de ingeniería. Si las cuatro aplicaciones basadas en Electron usaran en cambio el runtime compartido WebView2 Evergreen, I y R bajarían de 7 a 3, O para la corrección del motor de 6 a 2, L de 13 días a 4 y E de 63 a 11, con los mismos supuestos.

**CÓMO LLEGÓ A SER ASÍ**

Google anunció WebP en septiembre de 2010, construido sobre el códec de video VP8, y publicó libwebp como biblioteca abierta para que cualquiera pudiera decodificar el formato. Esa apertura explica por qué se extendió a navegadores, bibliotecas de imágenes y frameworks. El registro público también se retrasó: CVE-2023-4863 se publicó el 12 de septiembre como un error de Chrome, y un registro separado presentado contra libwebp el 25 de septiembre se rechazó dos días después por duplicado, mientras cada producto que empaquetaba la biblioteca tenía que decidir por su cuenta si estaba afectado.

## La hoja de trabajo del sobrecosto de las costuras

Las hojas de trabajo A a C son ilustraciones. Esta hoja en blanco convierte el capítulo en una página que un comité de finanzas puede leer. Se completa una copia por cada producto o imagen de dispositivo. Conviene contar a partir de un inventario, no de memoria, y nombrar a un responsable por línea.

| Partida | Qué contar | Conteo | Responsable | Factor de costo |
|---|---|---|---|---|
| Runtimes | Copias separadas de cada motor o runtime por máquina (Chromium, Node.js, JVM, .NET, Python) | | | Disco, memoria, tiempo de pruebas por copia |
| Canales de actualización | Rutas distintas que entregan código a la máquina | | | Regresiones, puntos de entrada a la cadena de suministro, revisiones de cambios |
| Retraso de parches | Días desde la corrección upstream hasta el último dispositivo corregido, por canal, para el último CVE crítico | | | Días de exposición, horas de incidente |
| Exposición de los decodificadores | C, I, O y R, más el retraso máximo L y los días-copia E, para cada analizador de imágenes, fuentes y video alimentado por entradas no confiables | | | Versiones de emergencia por año |
| Bifurcaciones (forks) | Componentes que se parchean localmente en lugar de usar las versiones upstream | | | Tiempo de ingeniería para reaplicar los cambios sobre cada versión upstream (rebase) |
| Densidad de aislamiento | Cargas de trabajo por host para cada unidad de aislamiento que se usa: proceso, contenedor, VM, microVM | | | Sobrecarga de memoria por unidad, hosts por carga, factura de nube |

La densidad de aislamiento pertenece a la hoja porque el aislamiento también es una costura. El capítulo 8 dio la cifra de la propia AWS: unos 5 MiB de sobrecarga de memoria por microVM de Firecracker. El navegador paga el mismo tipo de precio: el proyecto Chromium informa que el aislamiento completo de sitios en Chrome 67 para escritorio costaba entre un 10 y un 13 % más de memoria con muchas pestañas abiertas. Cada frontera que protege a la empresa cuesta memoria, y la hoja de trabajo hace explícito ese intercambio en lugar de dejarlo al azar.

## Esta capa en la lista de materiales del píxel

| Componente | Quién lo escribe | Quién paga | Quién puede actualizarlo | Plazo de soporte | ¿Expuesto a entradas no confiables? | Su decisión |
|---|---|---|---|---|---|---|
| Chromium empaquetado en cada aplicación Electron | Proyecto Chromium y equipo de Electron | Google, patrocinadores de Electron, cada proveedor de aplicaciones | Cada proveedor de aplicaciones, al publicar una versión | Las tres versiones principales más recientes de Electron, una nueva cada 8 semanas | Sí, si la aplicación carga contenido externo | Alquilar; controlar la ruta de parches y medir su retraso |
| Copias de decodificadores de imágenes compartidos (libwebp) | Un pequeño equipo upstream | Sobre todo Google | Cada producto que empaqueta una copia | Versiones upstream | Sí, cualquier imagen | Alquilar y financiar; seguir instalaciones, dueños y retraso, y pedir VEX |
| Canal de actualización del sistema operativo | Proveedor del sistema operativo o distribución | El proveedor | El proveedor | Fechas de soporte del sistema operativo | Indirectamente | Alquilar |
| Actualizadores de aplicaciones de terceros | Cada proveedor de software | Cada proveedor | Cada proveedor | La política de cada proveedor | Sí: cada actualizador descarga y ejecuta código | Alquilar; contarlos y limitar su número |
| Ruta de publicación de su producto | Su equipo | La empresa | La empresa | Su política | Sí | Controlar (construir solo las piezas críticas para la confianza) |

LENTE DE SEGURIDAD

**¿Quién puede cambiarlo?** En las costuras, todos los de la figura 11.1 pueden cambiar código en sus máquinas: el proyecto upstream, los proveedores de navegadores, el equipo del runtime, cada proveedor de bibliotecas y cada proveedor de aplicaciones. La empresa solo puede cambiar su propia ruta de publicación y la rapidez con que su flota instala las actualizaciones.

**¿Qué tan rápido se puede corregir?** Tan rápido como la cadena de dueños más larga de la instalación más lenta. En el caso de libwebp, la corrección pública de Chrome llegó el día 5, y las flotas pudieron empezar a instalarla ese día; las copias empaquetadas esperaron más, hasta el día 15 para las últimas bibliotecas de la tabla y más aún para cada producto construido sobre ellas. Al contar los saltos de los tres runtimes principales se conoce el mejor caso posible.

**¿Qué cambia la IA aquí?** Los buscadores de errores con IA ya han revelado fallas en bibliotecas muy compartidas (capítulo 13), y cada hallazgo de ese tipo en una biblioteca empaquetada se convierte en una corrección multiplicada por I instalaciones y O dueños. La IA puede ayudar a hacer el triaje de la cola de CVE, pero solo las decisiones de diseño, menos copias y menos dueños, eliminan un salto del árbol de parches.

IMPLICACIONES PARA EL NEGOCIO

**Seguridad.** El fabricante de software que publica una aplicación Electron está en medio de la cadena de parches de otro. Conviene publicar su propio retraso: los días desde una versión de seguridad upstream de Chromium hasta su actualización publicada, medidos en cada versión, y darle la misma prioridad de incidente que a un defecto de su propio código.

**Costo.** La empresa regulada debería aplicar la hoja de trabajo a su imagen estándar de portátil. Cada copia de runtime y cada canal de actualización que elimine es un elemento menos que inventariar, probar y explicar a un auditor, y un dueño menos a quien perseguir durante el próximo zero-day.

**Valor.** El operador de una flota de dispositivos paga el sobrecosto de las costuras miles de veces, una por quiosco. Menos copias y menos canales en la imagen significan una última fila más corta en la figura 11.1, que es la única fila que el operador controla y la que sienten sus clientes.

**Un ejercicio:** elegir el peor CVE de motor de navegador del año pasado y anotar, para cada instalación de su imagen estándar, el día en que quedó corregido en su último dispositivo.

CAPÍTULO DOCE

# El precio de poseer




Ser dueño de una capa da control y, con él, una nómina que dura tanto como la capa. El código es la parte barata. Las partes caras son las personas que lo entienden, la respuesta de seguridad que ahora hay que operar uno mismo, la compatibilidad que hay que mantener y el plazo de soporte que sigue corriendo después de la fiesta de lanzamiento. Este capítulo pone precio a esas partes y luego muestra cómo les fue a dueños reales cuando las pagaron, o cuando dejaron de pagarlas.

## Cinco costos que llegan después del código

Cuando una empresa pasa una capa de alquilar a poseer, cinco costos se mueven con ella. Ninguno aparece en la estimación de construcción.

- **Contratación.** Se necesitan personas capaces de cambiar la capa con seguridad, y suficientes como para que una renuncia no sea una crisis. El capítulo 8 mostró con xz lo que ocurre cuando una capa crítica descansa sobre un solo mantenedor no remunerado y bajo presión.
- **Respuesta de seguridad.** Alguien debe leer cada aviso que afecte a la capa, evaluarlo, corregirlo y publicarlo, con el reloj del atacante (el capítulo 11 lo mide).
- **Compatibilidad.** Todo lo que está por encima de la capa depende de su comportamiento. Cada cambio que se hace, y cada cambio que no se hace, perjudica a alguien.
- **Mantenimiento del fork.** Si se mantiene una copia modificada de un proyecto upstream, cada versión upstream es trabajo: fusionar, recompilar, volver a probar. Un fork también trae obligaciones de licencia, que se tratan más abajo.
- **El plazo de soporte.** Cada capa tiene una fecha de fin, fijada por uno mismo o por el proveedor. Los dispositivos en campo sobreviven a muchas de ellas.

El mantenimiento del fork es el costo que los equipos subestiman con más frecuencia, porque el proyecto upstream sigue avanzando, se le siga o no. La propia documentación de procesos de Chromium indica que Chrome publica una nueva versión mayor en el canal estable cada cuatro semanas y actualiza el canal estable cada semana. Un equipo que mantiene sus propios cambios sobre Chromium hereda esa cadencia.

La licencia es la otra parte del precio de un fork. Las preguntas frecuentes del proyecto GNU señalan que la GPL no obliga a publicar una versión modificada que solo se usa internamente, y que hacer copias dentro de una misma organización no es distribución. Pero en cuanto se distribuye, por ejemplo en dispositivos que se venden o en un producto que se entrega a clientes, hay que dar a quienes lo reciben

el código fuente correspondiente completo de esa versión, no solo los cambios propios, bajo la misma licencia. El kernel Linux se licencia solo bajo la GPL versión 2, que permite entregar ese código fuente con el dispositivo o incluir una oferta escrita, válida durante al menos tres años, para suministrarlo. La LGPL, usada por muchas bibliotecas, añade una obligación distinta: la versión 3, sección 4, permite combinar la biblioteca con código propio bajo términos propios solo si los usuarios pueden volver a enlazar la aplicación con una versión modificada de la biblioteca, o si se enlaza como biblioteca compartida que ellos pueden reemplazar. Hay que presupuestar revisión legal y un proceso de publicación de código fuente para cada fork distribuido de código copyleft.

Los casos de grandes empresas ilustran lo que implica esta decisión. ArkWeb de Huawei, el motor web de HarmonyOS, está, en palabras de la documentación de OpenHarmony, "desarrollado sobre la base del núcleo Google Chromium". La misma página asocia cada versión del sistema operativo a un hito de Chromium: M99 para OpenHarmony 4.0 y anteriores, M114 para 4.1 a 5.1 y M132 para 6.0. Chrome 114 llegó al canal estable el 30 de mayo de 2023 y Chrome 132 el 14 de enero de 2025. Pasar de una base a la siguiente es un proyecto, y entre un salto y otro el dueño carga con la brecha. Eso no es un fracaso. Es el precio de poseer un fork, pagado por una empresa lo bastante grande como para pagarlo.

**EJEMPLO: EL COSTO ANUAL DE MANTENER UN FORK AL DÍA**

**Supuestos:** un fabricante de software mantiene 40 parches locales sobre Chromium dentro de su producto. Sigue cada versión mayor: 13 al año con una cadencia de cuatro semanas. Cada rebase lleva 2 semanas-ingeniero para fusionar, compilar y resolver conflictos. Cada actualización semanal de seguridad lleva 1 día-ingeniero para fusionar y probar. Las pruebas de compatibilidad de cada versión mayor llevan otra semana-ingeniero. Un ingeniero en un turno rotativo de guardia debe estar disponible para publicaciones de emergencia.

**Resultado:** 13 × 2 = 26 semanas de rebase, 52 días (unas 10 semanas) de fusiones de seguridad y 13 semanas de pruebas: unas 49 semanas-ingeniero, es decir, un ingeniero a tiempo completo, antes de la guardia y antes de que nadie construya una funcionalidad. Dos ingenieros es el mínimo realista, para que uno pueda tomar vacaciones. Conviene ajustar los supuestos al número de parches propio; la forma no cambiará.

## Dueños que abandonaron

La historia de las plataformas está llena de empresas que fueron dueñas de una capa, la pagaron durante años y luego dejaron de pagarla. Las razones rara vez tuvieron que ver con mala ingeniería. El capítulo 10 contó por qué algunos sistemas sólidos perdieron el mercado; aquí la pregunta es más concreta: qué les pasó a quienes dependían de la capa cuando su dueño dejó de pagar.

| Capa y dueño | Qué pasó | Quién pudo corregirla después |
|---|---|---|
| Copland, el siguiente Mac OS de Apple | Archivado en agosto de 1996 en favor de actualizaciones incrementales de System 7; Apple eligió NeXT como su nueva base en diciembre de 1996 | Nadie: el código nunca se publicó como sistema. Mac OS X, construido sobre NeXT, se lanzó el 24 de marzo de 2001 |
| BlackBerry 10, BlackBerry | BlackBerry anunció el 28 de septiembre de 2016 que dejaría de desarrollar teléfonos internamente; sus servicios heredados para BlackBerry 10 terminaron el 4 de enero de 2022 | Nadie para el servicio: BlackBerry advirtió que los dispositivos afectados "dejarán de funcionar de forma fiable, incluidos datos, llamadas, SMS y 9-1-1" |
| Windows 10 Mobile, Microsoft | La última versión de la plataforma telefónica de Microsoft; la tabla de soporte de Microsoft indica el fin de soporte el 14 de enero de 2020 | Nadie: no hubo más actualizaciones de seguridad del dueño |
| Firefox OS, Mozilla | En diciembre de 2015 Mozilla dejó de vender teléfonos a través de operadores; en febrero de 2016 fijó el fin del desarrollo para smartphones en la versión 2.6 | La comunidad, mediante forks; la pila pasó a los televisores inteligentes |
| Ubuntu Touch, Canonical | El 5 de abril de 2017 Canonical anunció que pondría "fin a nuestra inversión en Unity8, el shell para teléfonos y convergencia" | La comunidad UBports tomó el código fuente y siguió desarrollando Ubuntu Touch |

Los dueños fueron francos sobre las razones. El anuncio de Mozilla decía: "no pudimos crear una propuesta de valor convincente y diferenciadora para el usuario final y no logramos construir el ecosistema completo". Mark Shuttleworth escribió que esperaba que la comunidad y la industria recibieran bien la convergencia sobre software libre, y "me equivoqué en ambos casos". En los dos casos el sistema operativo funcionaba. El costo que no se pudo cubrir fue el ecosistema a su alrededor: las aplicaciones, los operadores, los fabricantes de dispositivos.

Para un CTO, la lección está en la columna de la derecha. Cuando el dueño abandonó la capa, la capacidad de corregir la capa desapareció o pasó a quien tuviera el código fuente. Ubuntu Touch sobrevivió porque su código era abierto y había una comunidad dispuesta a mantenerlo. Los dispositivos BlackBerry 10 perdieron funciones básicas porque el servicio detrás de ellos era del dueño, y el dueño podía apagarlo. Antes de construir sobre cualquier capa, hay que preguntarse qué diría esa columna si su dueño la abandonara.

**CÓMO LLEGÓ A SER ASÍ**

Copland debía ser el siguiente Mac OS de Apple, previsto como Mac OS 8 y construido en torno a un nuevo micronúcleo. En agosto de 1996 Apple lo archivó en favor de actualizaciones incrementales semestrales que trasladarían parte de su tecnología. Su micronúcleo nunca se publicó. El 20 de diciembre de 1996 Apple anunció que adquiriría NeXT y, con ella, el sistema que se convertiría en Mac OS X. Apple anunció el 9 de enero de 2001 que Mac OS X saldría el 24 de marzo de 2001, más de cuatro años después de la adquisición. Apple fue dueña del sistema operativo del Mac todo ese tiempo. Ser dueña no salvó a Copland; significó que solo Apple podía decidir abandonarlo, y solo Apple podía pagar el reemplazo.

## Una capa propia que tuvo un mal lanzamiento, y otra que su dueño abandonó

**Apple Maps, 2012.** Con iOS 6, Apple reemplazó los mapas de sus teléfonos por un servicio construido por ella misma. En menos de una semana, el director ejecutivo de Apple estaba pidiendo disculpas. El 28 de septiembre de 2012 Tim Cook publicó una carta a los clientes: "Con el lanzamiento de nuestros nuevos Mapas la semana pasada, no estuvimos a la altura de este compromiso. Lamentamos profundamente la frustración que esto ha causado a nuestros clientes". Indicó a los usuarios que podían probar "Bing, MapQuest y Waze" desde el App Store, o los mapas de Google o Nokia en la web, mientras Apple mejoraba los suyos.

La carta muestra en un solo documento lo que se acepta al ser dueño. Ser dueña de la capa de mapas no hizo mejor el producto el día del lanzamiento. Lo hizo peor. Lo que cambió al poseerla fue quién podía corregirla: solo Apple, en los plazos de Apple, con el dinero de Apple. La misma carta dice que "ya hay más de 100 millones de dispositivos iOS que usan los nuevos Mapas de Apple", así que la corrección tenía que llegar a todos ellos por la propia ruta de actualización de Apple.

**EdgeHTML, 2018 a 2021.** El capítulo 7 contó cómo Microsoft anunció en diciembre de 2018 que reconstruiría Edge sobre Chromium. El paso final llegó después: Microsoft terminó el soporte de la antigua aplicación de escritorio Edge Legacy el 9 de marzo de 2021, indicando que "ya no recibirá futuras actualizaciones de seguridad". Microsoft podía permitirse seguir construyendo su propio motor. Decidió que el costo superaba el valor y trasladó Edge al proyecto de código abierto Chromium. Fue una elección, no una derrota. La siguiente sección muestra una empresa que tomó la decisión contraria porque quizá no se le permitiría alquilar.

## Poseer porque quizá no se permita alquilar: HarmonyOS

El 16 de mayo de 2019, la Oficina de Industria y Seguridad de EE. UU. añadió a Huawei y a 68 de sus filiales fuera de EE. UU. a la Entity List, que exige una licencia para exportar a Huawei artículos sujetos a las normas de exportación estadounidenses. Huawei presentó HarmonyOS en su conferencia de desarrolladores el 9 de agosto de 2019, describiéndolo como un sistema operativo multidispositivo construido sobre un micronúcleo. Una capa que Huawei alquilaba a terceros se había convertido en una capa que quizá no se le permitiría alquilar.

Las cifras disponibles sugieren que la apuesta se ha sostenido, sobre todo en China. Las cifras trimestrales de Counterpoint Research, a noviembre de 2026, muestran a los teléfonos con HarmonyOS en el 17 % de las ventas de smartphones en China en el tercer trimestre de 2024 y en el 24 % en el segundo trimestre de 2026, por delante de iOS en la mayoría de los trimestres mostrados. El informe anual 2025 de Huawei indica que a finales de 2025 más de 36 millones de dispositivos ejecutaban HarmonyOS 5 o HarmonyOS 6, más de 10 millones de desarrolladores se habían registrado y AppGallery ofrecía más de 350 000 aplicaciones y servicios. Son cifras de la propia Huawei.

Dos matices ponen la lección en su sitio. Primero, poseer tuvo sentido aquí porque la alternativa era el riesgo de no poder alquilar en absoluto. Ese caso no se aplica a la mayoría de las empresas medianas: salvo que controles de exportación, sanciones o un mandato de soberanía puedan dejar a la empresa sin acceso a su proveedor, HarmonyOS no es un modelo para su hoja de ruta. Segundo, incluso aquí el dueño alquila donde alquilar funciona: como mostró la primera sección, ArkWeb está basado en Chromium. Huawei es dueña del sistema operativo y de un fork del motor, y paga para que ese fork siga avanzando. Incluso donde poseer ha funcionado, el precio no desapareció.

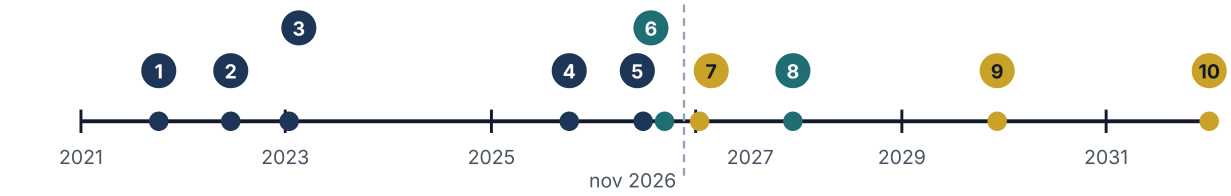
> *Poseer pone la corrección en sus manos. No promete que la corrección sea buena.*

### El calendario de vencimientos de la flota

El operador de una flota de dispositivos del Resumen para el lector, un grupo hotelero con quioscos de registro o un minorista con terminales de punto de venta, conserva sus máquinas muchos años. Cada máquina lleva varios plazos, y cada plazo pertenece a otro dueño. El quiosco sigue funcionando cuando un plazo se agota. Simplemente deja de ser confiable, o deja de poder corregirse.

FIGURA 12.1

## Plazos en una imagen de quiosco, 2021 a 2032




**1** sep 2021: vence DST Root CA X3 de Let's Encrypt

**2** jun 2022: fin del soporte de IE11 de escritorio

**3** ene 2023: Chrome 109, último para Windows 7 y 8.1

**4** oct 2025: fin del soporte de Windows 10 Home y Pro

**5** jun 2026: empiezan a vencer las CA 2011 de Secure Boot

**6** sep 2026: rige la notificación de vulnerabilidades CRA

**7** ene 2027: fin del soporte de Enterprise LTSC 2021

**8** dic 2027: el CRA de la UE rige por completo

**9** 2029: modo IE de Edge, "al menos hasta 2029"

**10** ene 2032: fin de Windows 10 IoT Enterprise LTSC 2021

pasado · regulación · por venir

Diez plazos que puede llevar una sola imagen de quiosco, cada uno fijado por un dueño distinto. Fechas de Let's Encrypt, las páginas de ciclo de vida de Microsoft y KB5062710, la página de soporte de Chrome de Google y la Comisión Europea. Las fechas de Secure Boot se explican en el capítulo 1.

Conviene leer el calendario desde el punto de vista del quiosco:

- **Certificados.** La antigua raíz de Let's Encrypt, DST Root CA X3, venció el 30 de septiembre de 2021. Let's Encrypt advirtió que los dispositivos que no confiaran en su nueva ISRG Root X1 mostrarían errores de certificado, y que los clientes con OpenSSL 1.0.x podrían fallar incluso si confiaban en ella. Un quiosco con un almacén de raíces congelado y una biblioteca TLS antigua pierde los servicios de red de los que depende, en una fecha que nadie en el hotel eligió.
- **Motores de navegador sin soporte.** Chrome 109, publicado el 10 de enero de 2023, fue la última versión para Windows 7 y 8.1. Electron extendió su versión 22 hasta el 10 de octubre de 2023 para cubrir los mismos sistemas. Después de eso, un quiosco con esos sistemas ejecutaba un motor que ningún dueño parcheaba.
- **Fin del soporte del sistema operativo.** El mismo código base de Windows 10 lleva plazos muy distintos. Windows 10 Home y Pro llegaron al fin de soporte el 14 de octubre de 2025. Windows 10 Enterprise LTSC 2021 llega al fin de soporte el 12 de enero de 2027; Microsoft vende Extended Security Updates para esta edición por hasta tres años más, hasta el 8 de enero de 2030, a 61 USD por dispositivo el primer año, a noviembre de 2026. Windows 10 IoT Enterprise LTSC 2021, la edición pensada para dispositivos de propósito fijo, tiene soporte hasta enero de 2032. Elegir la edición es elegir el plazo.
- **Regulación.** El Reglamento de Ciberresiliencia de la UE (Cyber Resilience Act, CRA), Reglamento (UE) 2024/2847, entró en vigor el 10 de diciembre de 2024. Sus obligaciones de notificación se aplican desde el 11 de septiembre de 2026 y sus obligaciones principales desde el 11 de diciembre de 2027. El artículo 13(8) exige a los fabricantes fijar un periodo de soporte que refleje cuánto tiempo se

espera que el producto esté en uso, y "el periodo de soporte será de al menos cinco años" salvo que se espere que el producto se use menos tiempo. Para una empresa que fabrica o vende dispositivos conectados en la UE, el plazo de soporte se convierte en una obligación legal.

El calendario de vencimientos es la versión del sobrecosto de las costuras para el operador de la flota. Cada plazo pertenece a un dueño distinto, y el operador controla lo único que importa: la fecha límite para que todos los dispositivos hayan migrado.

## Cuatro decisiones, cuatro precios

Cada fila de la lista de materiales del píxel termina en una de cuatro decisiones. Cada una tiene un precio, y cada precio se paga en una moneda distinta.

| Decisión | Qué se paga | Quién puede corregir un defecto | Velocidad típica | Riesgo principal |
|---|---|---|---|---|
| **Alquilar** | Licencia o suscripción; el tiempo del equipo para seguir la hoja de ruta del proveedor | El proveedor | El ciclo de publicaciones del proveedor | El proveedor cambia o termina la capa |
| **Financiar** | Dinero o tiempo de ingenieros entregado a los mantenedores de un proyecto abierto | Los mantenedores, con los ingenieros de la empresa participando | La velocidad upstream, y uno se entera antes | El proyecto aun así fracasa si la financiación es demasiado escasa |
| **Bifurcar (fork)** | El mantenimiento anual del ejemplo anterior, para siempre, más las obligaciones de licencia al distribuirlo | La empresa, para sus cambios; upstream para el resto | Tan rápido como se decida, después de cada rebase | Se queda atrás respecto a upstream y carga con la brecha |
| **Construir** | El equipo completo, la respuesta de seguridad, el ecosistema | Solo la empresa | Tan rápido como su propio equipo | Carga ella sola con todos los costos de este capítulo |

Financiar es la forma más barata de ganar influencia sobre una capa, y la más descuidada. No es propiedad y no compra control. Compra una voz, alerta temprana y un proyecto más sano, por una fracción del costo de un fork. La puerta trasera de xz (capítulo 8) mostró lo que puede llegar a cargar un solo mantenedor sin financiación. Una empresa que financia los pequeños proyectos bajo su producto paga la prima de un seguro que espera no tener que reclamar nunca.

El capítulo 15 convierte estos precios en puertas de control. La versión corta a partir de los casos de este capítulo: bifurcar solo con una partida presupuestaria específica para el mantenimiento y, antes de alquilar, leer la columna de la derecha de la tabla de "dueños que abandonaron".

## Esta capa en la lista de materiales del píxel

| Componente | Quién lo escribe | Quién paga | Quién puede actualizarlo | Plazo de soporte | ¿Expuesto a entradas no confiables? | Su decisión |
|---|---|---|---|---|---|---|
| Imagen del sistema operativo del quiosco (Windows 10 IoT Enterprise LTSC 2021) | Microsoft | Microsoft, a través de su licencia | Microsoft | Hasta enero de 2032 | Sí, por la red y las aplicaciones | Alquilar; planificar la siguiente imagen con años de antelación |
| Un motor de navegador bifurcado en su producto | El proyecto Chromium, más sus parches | Google, más su presupuesto de mantenimiento | La empresa, después de cada rebase | Mientras financie el rebase | Sí, cualquier página o imagen | Bifurcar solo con una partida presupuestaria específica |
| Almacén de raíces TLS y biblioteca en los dispositivos | El proveedor del sistema operativo o el proyecto de la biblioteca | El proveedor | El proveedor, o la empresa si está congelado | Fechas de vencimiento de los certificados raíz | Sí, cada conexión | Alquilar; nunca congelarlo |
| Pequeña biblioteca de código abierto bajo su producto | Unos pocos mantenedores | A menudo nadie | Los mantenedores | Mientras ellos continúen | A menudo | Financiar |
| Su compromiso de soporte de dispositivos | Su equipo | La empresa | La empresa | Al menos cinco años según el CRA, para productos en su ámbito | No se aplica | Construir |

LENTE DE SEGURIDAD

**¿Quién puede cambiarlo?** En una capa alquilada, el proveedor; en una financiada, los mantenedores a quienes se apoya; en un fork o una construcción propia, la empresa, y solo ella para sus propios cambios. Cuando un dueño abandona la capa, la respuesta cambia de la noche a la mañana, como comprobaron los usuarios de BlackBerry 10 en enero de 2022.

**¿Qué tan rápido se puede corregir?** Un dueño puede corregir una capa tan rápido como su propio equipo pueda publicar, lo cual es rápido solo si el equipo existe, tiene personal para la respuesta de seguridad y ha mantenido su fork al día. Apple corrigió Mapas en los plazos de Apple; nadie más podía hacerlo.

**¿Qué cambia la IA aquí?** La asistencia de IA abarata el rebase, la adaptación de los parches y las pruebas de un fork, lo que reduce el ejemplo de mantenimiento anterior. No aporta el personal de guardia para la respuesta de seguridad, no cumple una obligación de licencia ni extiende la fecha de soporte de un proveedor.

IMPLICACIONES PARA EL NEGOCIO

**Seguridad.** La empresa regulada debería listar cada capa cuyo dueño podría abandonarla y anotar quién podría parchearla al día siguiente. Donde la respuesta sea "nadie", esa capa necesita un plan de salida antes de que lo pida un auditor.

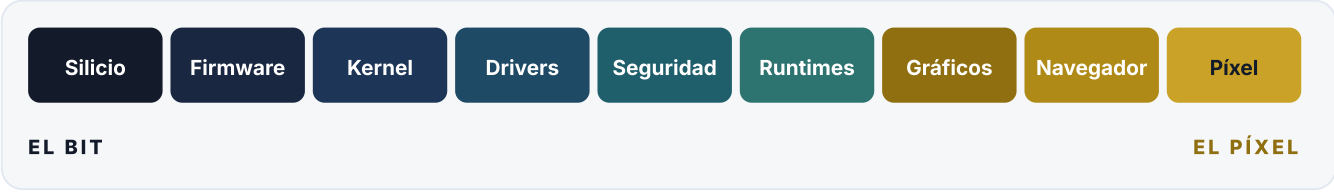
**Costo.** El fabricante de software que mantiene sus propios parches sobre un gran proyecto upstream debería poner el mantenimiento anual del fork en el presupuesto como una partida presupuestaria específica, con al menos dos ingenieros. Un fork sin presupuesto es un fork que se queda atrás.

**Valor.** El operador de una flota de dispositivos debería comprar hardware y ediciones del sistema operativo según sus plazos de soporte, no según su precio de lista, y mantener el calendario de vencimientos de la flota junto al calendario de depreciación. Un quiosco más barato de comprar pero con un plazo de soporte más corto puede costar más a lo largo de su vida.

**Antes del próximo presupuesto:** para cada capa que posea o bifurque, nombrar a las personas que pueden corregirla y el presupuesto que las sostiene; para cada capa que alquile, anotar la fecha en que se agota su plazo.

CAPÍTULO TRECE

# La IA mueve las costuras




La IA no inventó un nuevo tipo de seguridad. Cambió los costos. Encontrar una falla se abarató, los atacantes obtuvieron nuevas herramientas para una carrera que ya se estaba acelerando, el software que hay que defender creció y una cara en una pantalla dejó de ser prueba de nada. Los defensores cuentan con las mismas herramientas. El único reloj que no se aceleró por sí solo es el tiempo que tarda su empresa en publicar una corrección.

Este capítulo recorre la pila del bit al píxel con ese cambio en mente. A la pregunta del libro sobre quién puede corregir cada capa le añade un reloj: cuando las fallas se encuentran y se explotan más rápido, la velocidad de la corrección es lo que decide el resultado. Los capítulos anteriores ya presentaron partes de esta historia. El capítulo 4 trató los agentes que hacen clic en "Acepto" y las videollamadas con deepfakes. El capítulo 5 trató la seguridad de memoria y la traducción de C a Rust asistida por IA. El capítulo 8 trató la cadena de herramientas (toolchain) como frontera de confianza. Aquí las piezas se juntan.

## Cuatro costos que cambiaron

Una forma útil de leer la evidencia es como cuatro costos, cada uno con un ejemplo fechado. Ninguno es un pronóstico.

**Los ataques se abarataron.** El 13 de noviembre de 2025 Anthropic informó que a mediados de septiembre había detectado una campaña de espionaje que atribuyó, con alta confianza, a un grupo patrocinado por el Estado chino al que designó GTG-1002. El grupo usó Claude Code, el agente de programación de Anthropic, contra unos treinta objetivos y tuvo éxito en un número reducido de casos. El informe de Anthropic dice que el actor usó la IA "para ejecutar de forma independiente el 80-90 % de las operaciones tácticas", con humanos en unos pocos puntos de decisión. El mismo informe añade un límite importante: Claude "con frecuencia exageraba los hallazgos y en ocasiones fabricaba datos", al asegurar que había obtenido credenciales que no funcionaban y descubrimientos que ya eran públicos. En este caso la IA no fue un atacante impecable, pero sí uno barato.

**Los exploits llegaron más rápido.** El promedio de tiempo hasta la explotación de Mandiant cayó durante años antes de que las herramientas de IA generativa estuvieran ampliamente disponibles (el capítulo 11 da las cifras y sus salvedades), por lo que esos datos no pueden demostrar que la IA lo causara o lo acelerara. Lo que ha cambiado desde entonces es que las herramientas de IA ahora se usan en ambos lados: por los atacantes, como muestra GTG-1002, y por los defensores, como muestra Big Sleep de Google en la siguiente sección.

**La superficie creció.** Cada producto que incorpora un modelo de lenguaje incorpora un nuevo tipo de entrada. El OWASP Top 10 for Large Language Model Applications situó la **inyección de prompts** en primer lugar en su edición de 2025. La inyección de prompts consiste en ocultar instrucciones en el contenido que lee el modelo, como una página web, un correo o un documento, para que el modelo las siga. El texto de OWASP de 2025 dice que "no está claro si existen métodos de prevención infalibles". La edición de 2026, publicada en agosto de 2026, mantiene la inyección de prompts en el número uno. La orientación pasó de prevenir el truco a limitar lo que puede hacer un modelo engañado.

**La pantalla dejó de ser prueba.** El capítulo 4 contó el caso Arup: en enero de 2024 un empleado de finanzas en Hong Kong hizo 15 transferencias por un total de 200 millones de dólares de Hong Kong tras una videollamada en la que todos los demás participantes eran deepfakes. Para este capítulo la idea es más concreta. Un píxel que muestra a una persona, un documento o una confirmación ahora cualquier otro puede producirlo a bajo costo. La confianza tiene que venir de la pila que está debajo: identidad, firma y un segundo canal.

## Los defensores también cuentan con estas herramientas

Las mismas herramientas sirven para la defensa, y la evidencia ahí es igual de concreta.

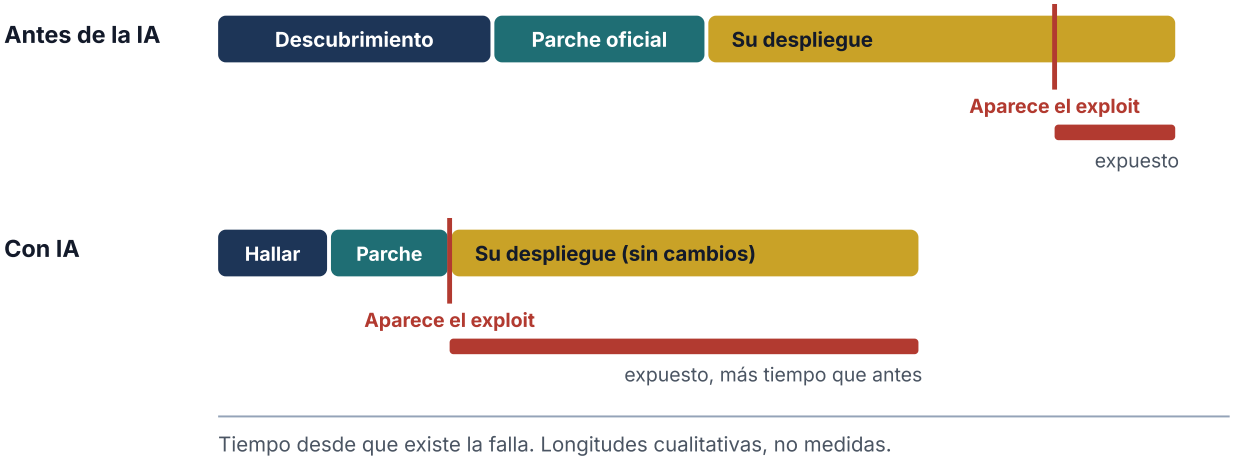
El 1 de noviembre de 2024 Project Zero y DeepMind de Google describieron **Big Sleep**, un agente basado en un modelo de lenguaje que encontró un subdesbordamiento de búfer en la pila de llamadas (stack buffer underflow) explotable en SQLite, el motor de base de datos integrado en navegadores, teléfonos e incontables aplicaciones. El equipo lo comunicó a principios de octubre de 2024, los desarrolladores de SQLite lo corrigieron el mismo día y la falla nunca llegó a una versión oficial. En julio de 2025 Google dijo que Big Sleep había encontrado `CVE-2025-6965`, una falla de corrupción de memoria de SQLite corregida en la versión 3.50.2. Según Google, su equipo de inteligencia de amenazas vio indicios de que unos atacantes preparaban un zero-day, y Big Sleep ayudó a aislar la falla antes de que se usara. En agosto de 2025 el vicepresidente de seguridad de Google anunció el primer lote de 20 fallas divulgadas por Big Sleep en proyectos de código abierto, entre ellos FFmpeg e ImageMagick.

La agencia de investigación estadounidense DARPA organizó el **AI Cyber Challenge** (AIxCC) para probar esto a escala. En la ronda final en DEF CON, en agosto de 2025, según DARPA, los sistemas competidores "descubrieron 54 vulnerabilidades sintéticas únicas en los 63 desafíos de la Competencia Final" y parchearon 43 de ellas. (DARPA corrigió una cifra anterior de 70 vulnerabilidades sintéticas a 63.) También encontraron 18 fallas reales que no habían sido sembradas, y aportaron 11 parches para ellas. El tiempo promedio para enviar un parche fue de 45 minutos, con un costo promedio de unos 152 dólares por tarea. Team Atlanta ganó el primer premio, de 4 millones de dólares, Trail of Bits quedó segundo y Theori tercero.

Hay que leer estas cifras por lo que son: un concurso sobre código preparado y un proveedor describiendo su propia herramienta. Muestran que las máquinas pueden encontrar y corregir errores reales. No muestran que su flota esté más segura. Un parche escrito en 45 minutos no protege a nadie hasta que se compila, se firma, se prueba y se instala, y ese trabajo ocurre en su lado de la costura.

FIGURA 13.1

## El reloj de respuesta, antes y después de la IA




Tiempo desde que existe la falla. Longitudes cualitativas, no medidas.

Allí donde las herramientas de IA acortan el descubrimiento, la corrección del proveedor y el exploit del atacante, no acortan su despliegue a menos que la empresa lo haga. Las longitudes son una ilustración, no una medición. Cuando el exploit se mueve a la izquierda y su despliegue no, la ventana roja crece.

> *Las herramientas de IA han demostrado que pueden encontrar una falla y escribir una corrección en minutos. El tiempo para instalar esa corrección en sus equipos sigue dependiendo de la empresa.*

Este es el argumento de las costuras del capítulo 11 con un reloj encima. La exposición la fija el dueño más lento en el camino entre una corrección publicada y una máquina en funcionamiento. Cuando una capa tiene cinco dueños, cada uno con su propio ciclo de publicaciones, el más lento de los cinco fija su ventana. Allí donde las herramientas de IA aceleran las partes rápidas, las partes lentas se vuelven más visibles. No conviene adoptar un plazo universal de parches sacado de un informe del sector. Conviene medir el tiempo propio desde la divulgación hasta la mitigación, por capa, y anotarlo.

### El píxel es una entrada: la semántica de la interfaz se convierte en una superficie de integración versionada

La pantalla no era solo para personas antes de 2024: scripts escritos para una aplicación a la vez ya manejaban software empresarial a través de sus pantallas, como cuenta el recuadro de abajo. Lo que cambió en 2024 y 2025 fue la llegada de agentes de propósito general que usan una computadora como lo hace una persona. El 22 de octubre de 2024 Anthropic lanzó computer use (uso de la computadora por agentes) en beta pública: un modelo que interpreta capturas de pantalla, mueve el cursor, hace clic y escribe. El 23 de enero de 2025 OpenAI lanzó Operator como vista previa de investigación, impulsado por un modelo que llama Computer-Using Agent, que "ve" un navegador a través de capturas de pantalla y actúa con el mouse y el teclado.

Las capturas de pantalla son una vía de entrada. La otra son los datos de accesibilidad. Cada sistema de escritorio expone una descripción estructurada de lo que hay en pantalla, creada para que los lectores de pantalla puedan decirle a un usuario ciego que esto es un botón con la etiqueta "Enviar" y aquello un campo de texto con la etiqueta "Monto". La especificación Core Accessibility API Mappings del W3C describe cómo el contenido web se asigna a la interfaz de cada plataforma: UI Automation en Windows, ATK y AT-SPI en Linux, y la AX API en macOS. Las herramientas de agentes ahora leen esos mismos datos. El servidor Playwright MCP de Microsoft, por ejemplo, permite que un modelo maneje un navegador "mediante instantáneas estructuradas de accesibilidad", y su documentación dice que usa el árbol de accesibilidad, "no entrada basada en píxeles".

| Pregunta | Windows | Linux | macOS |
|---|---|---|---|
| **Interfaz de accesibilidad que un agente puede leer** | UI Automation | ATK y AT-SPI | AX API (NSAccessibility) |
| **Quién define la semántica de los controles** | Los frameworks de interfaz de Microsoft, y luego cada equipo de aplicación | Proyectos de toolkit como GTK y Qt, y luego cada equipo de aplicación | AppKit y SwiftUI de Apple, y luego cada equipo de aplicación |
| **Cómo pide una aplicación capturar la pantalla** | Windows.Graphics.Capture: el usuario elige en la interfaz del sistema, y un borde amarillo marca lo que se captura | En escritorios Wayland, el portal ScreenCast de xdg-desktop-portal | El permiso de Grabación de pantalla bajo TCC (capítulo 4) |

Si se juntan esos hechos, se desprende una consecuencia en la que pocos equipos de producto han reparado. Su interfaz de usuario es ahora una superficie de integración. El agente de un cliente, la automatización de un socio o su propio bot de soporte pueden depender de la etiqueta de un botón, del orden de los campos o de la estructura de una página. Basta con cambiar "Enviar" por "Continuar", mover el campo de monto a un diálogo o reemplazar los controles nativos por un canvas para que la automatización de alguien se rompa, en silencio, en producción. Nada en el registro de cambios de su API se lo advirtió, porque la API que cambió fue la pantalla.

La respuesta es tratar la semántica de la interfaz como se trata una API. Dar nombres y roles estables a los controles importantes. Anunciar los cambios en ellos. Probarlos con las mismas herramientas de accesibilidad que usa un lector de pantalla. Esto tiene un efecto secundario agradable: el trabajo de accesibilidad y la preparación para agentes son ahora el mismo trabajo. Un producto con un árbol de accesibilidad limpio sirve a los usuarios ciegos y da a los agentes algo fiable que leer. También es una parte necesaria para cumplir la Ley Europea de Accesibilidad (European Accessibility Act) que describió el capítulo 6, pero no toda: el anexo de la ley también pide servicios perceptibles, operables, comprensibles y robustos, alternativas al color y al contenido no textual, e información sobre cómo funciona el servicio con tecnologías de apoyo. Un producto que lo dibuja todo en un canvas debe reconstruir ese árbol, como explicó el capítulo 6, y ahora lo reconstruye para dos públicos.

**CÓMO LLEGÓ A SER ASÍ**

El software leía las pantallas de otro software mucho antes de los modelos de lenguaje. Las herramientas de automatización robótica de procesos (RPA) manejaban aplicaciones empresariales a través de sus interfaces de usuario cuando esas aplicaciones no tenían API, y el mercado creció lo suficiente como para que UiPath, uno de sus proveedores más conocidos, empezara a cotizar en la Bolsa de Nueva York el 21 de abril de 2021. Los investigadores también construyeron conjuntos de datos a partir de pantallas: Rico, presentado en UIST 2017, reunió unas 72 000 pantallas de Android de unas 9700 aplicaciones, cada una con una captura de pantalla y la jerarquía de vistas que tenía detrás. Los hábitos ya existían. Los agentes de computer use de octubre de 2024 y enero de 2025 cambiaron la escala y la habilidad: un agente ya no necesitaba un script escrito para cada pantalla.

## Los datos de pantalla son datos de IA

Si los agentes trabajan con capturas de pantalla e instantáneas de accesibilidad, esas imágenes y esos árboles son datos, y contienen lo que hubiera en la pantalla: nombres de clientes, números de cuenta, historiales médicos, precios internos. Tres flujos merecen un lugar en su inventario de datos.

- **Capturas de los agentes.** Un agente de computer use envía imágenes de la pantalla a un modelo para decidir su siguiente paso. Si el modelo se ejecuta fuera de la empresa, esas imágenes salen de la máquina. Dónde se guardan, durante cuánto tiempo y si el proveedor puede usarlas lo fija su contrato con ese proveedor, no el agente.
- **Memoria de pantalla del sistema operativo.** Recall de Microsoft guarda instantáneas periódicas en las computadoras Copilot+ para que un usuario pueda buscar lo que ha visto. El capítulo 4 describió su rediseño. Microsoft afirma que las instantáneas permanecen en el dispositivo, están cifradas y no se usan para entrenar sus modelos de IA.
- **Grabaciones y reproducciones.** Las grabaciones de pantalla para soporte, la reproducción de sesiones para analítica y las pantallas compartidas en reuniones crean imágenes almacenadas de datos de clientes.

Hay que ser preciso aquí, porque el debate a menudo no lo es. **Capturar no es entrenar.** Una instantánea guardada en una computadora portátil es una cuestión de privacidad y seguridad: quién puede leerla y qué pasa cuando roban la portátil. Una captura de pantalla enviada a un proveedor de modelos es una cuestión de tratamiento de datos: lo que permite el contrato. El entrenamiento es una tercera cuestión, y solo se aplica cuando alguien realmente construye un conjunto de datos, como hicieron los investigadores de Rico de forma deliberada y abierta. Cada una necesita su propio control.

## Esta capa en la lista de materiales del píxel

Estas son las filas que la IA añade a la lista. El capítulo 15 las reúne con todas las demás capas, añade la columna **plazo de parche si la IA lo encuentra primero** y pregunta: si una máquina encontrara mañana una falla en este componente, ¿cuánto tardaría la corrección en ejecutarse en sus máquinas, y qué reloj decide?

| Componente | Quién lo escribe | Quién paga | Quién puede actualizarlo | Plazo de soporte | ¿Expuesto a entradas no confiables? | Su decisión |
|---|---|---|---|---|---|---|
| Biblioteca de código abierto integrada (SQLite, FFmpeg, decodificadores de imagen) | Un pequeño equipo de mantenedores | Patrocinadores y voluntarios | Los mantenedores, y luego cada producto que la empaqueta | Las versiones del proyecto | Sí: archivos y datos de fuera | Alquilar y financiar; mantener ensayada la recompilación |
| Modelo alojado detrás de su agente o funcionalidad | El proveedor del modelo | La empresa, por uso | Solo el proveedor | El calendario de retirada de modelos del proveedor | Sí: prompts, documentos y páginas que lee | Alquilar; pactar por contrato el aviso de retirada de modelos, la retención de capturas y prompts, y el uso para entrenamiento |
| Identidad, permisos y lista de herramientas del agente | Su equipo, sobre un framework de agentes alquilado | La empresa | La empresa | El suyo | Sí: cada página y archivo que lee el agente | Construir la política: identidad propia, mínimo privilegio, aprobación humana para acciones irreversibles |
| Etiquetas, roles y árbol de accesibilidad de sus pantallas | Su equipo de aplicación, sobre el toolkit del sistema operativo | La empresa | La empresa, dentro de los límites del toolkit | Su ciclo de publicaciones | Indirectamente: los agentes actúan según lo que dice | Construir; nombrar, anunciar y probar los cambios como se haría con una API |

LENTE DE SEGURIDAD

**¿Quién puede cambiarlo?** Más partes que antes. Toda parte que puede publicar actualizaciones en sus máquinas sigue pudiendo hacerlo, y ahora los agentes que actúan con los privilegios de sus usuarios también pueden cambiar datos y configuraciones. El contenido que lee un agente, como una página web o un correo, puede intentar cambiar lo que hace el agente. Incluir a los agentes como actores con acceso de escritura, igual que el capítulo 8 incluyó a los colaboradores y las máquinas de compilación.

**¿Qué tan rápido se puede corregir?** Las correcciones upstream pueden ser rápidas: SQLite corrigió la primera falla de Big Sleep el mismo día, y los sistemas de AIxCC enviaron parches en 45 minutos en promedio sobre código preparado. Ninguno de los dos ejemplos midió el despliegue, que es la parte de su lado. Medir el tiempo desde la divulgación hasta la mitigación para cada fila de la lista, y trabajar primero en el más largo.

**¿Qué cambia la IA aquí?** En esta capa la IA convierte en entrada la pantalla, el árbol de accesibilidad y cada página que lee un agente, y el texto de OWASP de 2025 dice que no está claro si la inyección de prompts puede prevenirse de forma fiable; la edición de 2026 la sigue situando en primer lugar. Los controles que funcionan son viejas ideas de sistemas operativos aplicadas a los agentes: mínimo privilegio, una separación estricta entre el contenido externo y las instrucciones, y aprobación humana para las acciones irreversibles.

IMPLICACIONES PARA EL NEGOCIO

**Seguridad.** Una **empresa regulada** debería tratar cada agente de IA como un usuario privilegiado. Darle su propia identidad, los permisos más estrechos que funcionen y ninguna autoridad para aprobar por sí solo pagos, firmas o concesiones de acceso. Registrar lo que leyó además de lo que hizo, para poder rastrear después una instrucción inyectada.

**Costo.** Un **operador de una flota de dispositivos** paga dos veces la defensa con IA si no puede desplegar: una vez por las herramientas que encuentran fallas en la imagen del quiosco, y otra por la emergencia cuando una corrección queda pendiente de despliegue. Conviene gastar primero en la ruta de actualización, el despliegue escalonado y la reversión en toda la flota, y después en detección. Los presupuestos de seguridad crecen (Gartner pronosticó 213 000 millones de dólares de gasto mundial en seguridad de la información para 2025), pero la mayor parte de ese dinero compra herramientas que se alquilan, y nada de él acorta la brecha entre una corrección publicada y su flota.

**Valor.** Un **fabricante de software** cuyas pantallas son estables, están bien etiquetadas y son accesibles resulta más fácil de usar para los agentes de sus clientes, y para los usuarios ciegos, con el mismo trabajo. Publicar una política de cambios para la semántica de la interfaz, como ya hace con su API, convierte un riesgo oculto en una característica en la que los clientes pueden confiar.

**Esta semana:** para cada fila de su lista de materiales, anotar el tiempo desde una corrección publicada hasta que esa corrección se ejecuta en sus máquinas, y acortar el más largo antes de comprar otra herramienta.

CAPÍTULO CATORCE

# Controlar el bit y la cadena de herramientas




Una empresa mediana no escribirá su propio kernel, y no debería intentarlo. Pero cada bit que publica pasa por un compilador, una compilación, una firma y un canal de actualización antes de llegar a un cliente. Ese camino es la capa que un equipo pequeño puede controlar más a fondo. Se siguen alquilando el compilador, las tiendas de aplicaciones, la notarización y los instaladores del sistema operativo, pero la cadena de compilación y publicación (pipeline), las claves de firma y la decisión de publicar pueden ser propias, y juntos deciden con qué rapidez llega cualquier otra corrección. Si de este libro se controla una sola cosa, que sea el camino desde el código fuente hasta la máquina del cliente en funcionamiento.

El capítulo 8 describió la cadena de herramientas (toolchain) como una capa y una frontera de confianza, y el capítulo 13 sostuvo que, a medida que las fallas se encuentran y se explotan más rápido, la velocidad de despliegue es la parte de la seguridad que decide el resultado. Este capítulo convierte esas conclusiones en un plan. Examina qué incluye la cadena de herramientas, qué se alquila y qué se controla en ella, empresas que eligieron ser dueñas de una capa y alquilar el resto, y lo que los agentes de IA cambian, y no cambian, en el tamaño del equipo necesario.

## Por qué la cadena de herramientas va primero

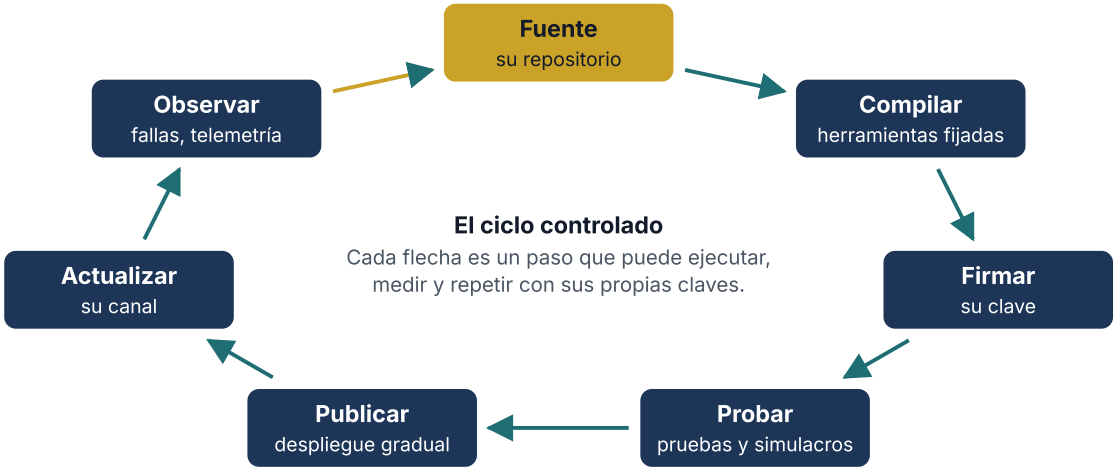
La **cadena de herramientas**, en el sentido amplio que usa este capítulo, es todo lo que hay entre la idea de un desarrollador y un binario en ejecución en la máquina de un cliente:

- **el compilador y la versión del lenguaje**, que deciden qué errores se detectan antes de que el programa se ejecute;
- **el editor o IDE y sus asistentes**, que ahora incluyen agentes de IA que escriben y modifican código;
- **la compilación**, que convierte el código fuente y las dependencias en artefactos, idealmente con los mismos bytes cada vez;
- **el empaquetado y la firma**, que deciden en qué confiará una máquina;
- **la publicación y la actualización**, que deciden con qué rapidez llega una corrección a cada máquina, y si se puede detener una mala;
- **la observación**, los informes de fallas y la telemetría que indican qué corregir a continuación.

Tres razones ponen esta capa en primer lugar. Es **lo bastante pequeña**: un equipo de pocos ingenieros puede operarla, a diferencia de un kernel o un motor de navegador. Tiene un **efecto multiplicador**: cada corrección de una dependencia que se distribuye, como un runtime, un motor o una biblioteca empaquetados, llega a los clientes solo a través de ella, así que mejorarla mejora la respuesta para todos los clientes. Y **ya está, en gran medida, en manos de la empresa**, así que la verdadera decisión es si se opera bien o mal.

FIGURA 14.1

### El ciclo controlado de la cadena de herramientas




Cada paso tiene un dueño y un tiempo. Un equipo es dueño del ciclo cuando puede ejecutar cada paso por sí mismo, con sus propias claves, y decir cuánto tarda una vuelta completa. La observación alimenta el siguiente cambio, así que el ciclo nunca termina.

El ciclo tiene una velocidad, y esa velocidad es la cifra que hay que gestionar. Un equipo que puede completar una vuelta entera, desde una línea modificada hasta una corrección ejecutándose en cada máquina, en horas, puede absorber una corrección de seguridad upstream el mismo día. Un equipo cuya vuelta tarda semanas queda expuesto durante semanas, compre las herramientas que compre.

## Qué se alquila y qué se controla

El camino tiene partes que se alquilan y partes que se controlan, y la división es distinta en cada sistema. Se controla una parte cuando se decide cómo se compone, se firma y se publica, aunque no se haya escrito.

| Pregunta | Windows | Linux | macOS |
|---|---|---|---|
| **Qué se alquila (detalle en el capítulo 8)** | El compilador; la Microsoft Store, si se usa | El compilador; la distribución o una tienda Flatpak o Snap | Apple Clang; el App Store o la notarización |
| **Qué queda bajo su control** | La cadena de compilación (build pipeline), las claves de firma, su propio servicio de actualización donde la plataforma lo permite, y la decisión de publicar | | |

Las capas alquiladas también están cambiando sus propias cadenas de herramientas. El capítulo 2 da el estado de Rust en los kernels de Windows y Linux a noviembre de 2026. Para una empresa que alquila el kernel, la lección es breve: a los proveedores de controladores y agentes se les preguntará por su hoja de ruta de lenguajes, y un equipo que construye su propio kernel para una flota de dispositivos ahora lleva un compilador de Rust en esa compilación.

## Poseer una capa, alquilar el resto

La evidencia más sólida a favor de la propiedad selectiva viene de empresas que eligieron una capa, la construyeron y siguieron alquilando todo lo que la rodea. Cada caso de abajo es una decisión de ingeniería pública y fechada.

| Capa propia | Dueño | Lenguaje y base | Hechos fechados | Lo que sigue alquilado |
|---|---|---|---|---|
| Firecracker, un monitor de microVM | AWS | Rust, sobre el hipervisor KVM de Linux | Anunciado el 26 de noviembre de 2018; la publicación decía que impulsaba AWS Lambda y Fargate | El kernel Linux y KVM |
| Pingora, un framework de proxy HTTP | Cloudflare | Rust | Descrito el 14 de septiembre de 2022; liberado como código abierto el 28 de febrero de 2024 bajo Apache 2.0 | El sistema operativo, el compilador, el hardware |
| workerd, el runtime de Workers | Cloudflare | C++, sobre instancias aisladas de V8 (isolates) | Liberado como código abierto el 27 de septiembre de 2022 bajo Apache 2.0 | El motor JavaScript V8 del proyecto Chromium |
| GPUI, el framework de interfaz del editor Zed | Zed Industries | Rust, dibujando con la GPU | Zed liberado como código abierto el 24 de enero de 2024; versión para Linux en julio de 2024; versión para Windows en octubre de 2025 | Los controladores de GPU de cada plataforma; las pilas de texto del sistema operativo |
| gVisor, un kernel de aplicación para contenedores | Google | Go | Liberado como código abierto el 2 de mayo de 2018 | El kernel Linux del host |

Vale la pena ver por qué construyó cada equipo. La publicación de Cloudflare sobre Pingora dice que "a medida que Cloudflare ha crecido, NGINX se nos ha quedado pequeño", y sopesa tres opciones: quedarse con NGINX y quizá bifurcarlo, pasar a un proxy de terceros o construir. Advierte que pasar al proxy de otro proveedor significaba que "el mismo ciclo podría repetirse en unos años". Informa que, con el mismo tráfico, Pingora usaba aproximadamente un 70 % menos de CPU y un 67 % menos de memoria que el servicio anterior. La decisión se refería a una capa en la que las necesidades de Cloudflare habían dejado atrás al producto alquilado.

Zed es la versión a nivel de píxel de la misma decisión. Su equipo construyó GPUI para renderizar el editor "como un videojuego", con shaders en la GPU, Metal en macOS y, en su versión para Windows de octubre de 2025, DirectX 11 para dibujar y DirectWrite para el texto. Ser dueño del renderizador dio a Zed el control de cada fotograma. También le dio todos los costos que el capítulo 6 enumeró para un renderizador propio: su propio trabajo de accesibilidad, sus propios casos límite de texto y un nuevo back end gráfico para cada plataforma que añade. Obsérvese que incluso aquí Zed alquila DirectWrite y Core Text para el texto. Ser dueño de una capa rara vez significa ser dueño de las capas que tiene debajo.

Y vale la pena ver lo que cada uno siguió alquilando. Firecracker depende de la virtualización del kernel Linux. gVisor se apoya en un kernel del host. workerd integra V8. Cada equipo es dueño de una capa que puede defender como su elemento diferenciador, y alquila las capas grandes y compartidas que tiene debajo.

**CÓMO LLEGÓ A SER ASÍ**

Rust 1.0 se anunció el 15 de mayo de 2015. Más tarde se convirtió en el lenguaje de tres de las capas propias de arriba: Firecracker en 2018, Pingora en 2022 y GPUI de Zed. gVisor, publicado el mismo año que Firecracker, se escribió en Go, elegido, según su anuncio, por su seguridad de memoria y de tipos. El patrón es más antiguo que ambos lenguajes. Las empresas de infraestructura llevan mucho tiempo reconstruyendo una parte de su pila cuando la escala convertía la versión alquilada en el cuello de botella. Lo que cambió en este periodo es que un lenguaje de sistemas más seguro hizo menos probable que esas reconstrucciones añadieran nuevas fallas de seguridad de memoria, lo que a su vez hizo la decisión más fácil de defender ante un equipo de seguridad.

> *Ser dueño de la capa que diferencia el producto. Alquilar las capas grandes y compartidas que tiene debajo, y ser dueño del camino que publica ambas.*

## Los agentes de IA y el tamaño del equipo

Las herramientas de programación con IA abaratan parte de este trabajo. La pregunta honesta es qué parte, y la evidencia apunta en ambas direcciones.

**Donde la evidencia apunta a un menor costo.** El experimento controlado de GitHub, publicado el 7 de septiembre de 2022, pidió a 95 desarrolladores que escribieran un servidor HTTP; quienes usaban Copilot terminaron un 55 % más rápido en promedio. El programa de investigación DORA de Google

encuestó a casi 5000 profesionales de tecnología para su informe de septiembre de 2025 y encontró que el 90 % usa IA en el trabajo; su resumen describe el papel principal de la IA como "el de un amplificador". El 22 de abril de 2026 el director ejecutivo de Google escribió que el 75 % de todo el código nuevo en Google lo genera ahora la IA y lo aprueban ingenieros, frente al 50 % del otoño anterior. Y el programa TRACTOR de DARPA, descrito en el capítulo 5, financia investigación para automatizar la traducción de C a Rust.

**Donde la evidencia pide cautela.** Un ensayo aleatorizado del grupo de investigación METR, publicado el 10 de julio de 2025, siguió a 16 desarrolladores experimentados de código abierto en 246 tareas reales de sus propios proyectos. Con herramientas de IA tardaron un 19 % más, aunque esperaban una aceleración del 24 % y después seguían creyendo que habían sido más rápidos. En febrero de 2026 METR publicó una actualización con un grupo más grande y dijo que sus nuevos resultados eran una señal poco fiable, probablemente una subestimación del beneficio, y que el resultado de 2025 debía considerarse desactualizado. Esa honestidad es lo importante. Las ganancias medidas dependen mucho de la tarea, del código base y del año. El proyecto curl muestra ambas caras en un mismo lugar: su mantenedor escribió en julio de 2025 que alrededor del 20 % de los informes de seguridad recibidos ese año eran ruido generado por IA, y en septiembre de 2025 elogió un conjunto de informes asistidos por IA que llevaron a decenas de correcciones reales.

Mi valoración, que es un juicio y no una medición: la IA reduce el costo de **escribir, migrar y revisar** código, y eso hace que construir una pequeña capa propia sea más asequible que antes. No reduce, y puede aumentar, otros cuatro costos:

- **Respuesta de seguridad.** Alguien debe clasificar los informes, muchos de ellos generados por máquinas, decidir cuáles son reales y publicar correcciones con un plazo de respuesta que, como mostró el capítulo 11, es cada vez más corto.
- **Compatibilidad.** Cada capa propia debe seguir funcionando en cada versión del sistema operativo, dispositivo y configuración de cliente soportados. Eso es probar en hardware real, no escribir.
- **Guardia.** Una capa propia necesita personas que respondan a las 2 de la madrugada y que entiendan el código, incluido el código que escribió un agente.
- **Contratación y continuidad.** El equipo debe sobrevivir a la salida de personas. El código que solo entienden un agente y un ingeniero es un punto único de falla.

Así que la regla para el tamaño del equipo no es "la IA permite que tres personas sean dueñas de un kernel". Es más acotada: la IA permite que un equipo pequeño sea dueño de una capa pequeña, y hace más importante que el equipo sea dueño del ciclo que la publica.

### Capacidad de recuperación, capa por capa

El capítulo 8 presentó el tiempo de recuperación probado: el tiempo medido para recompilar, firmar, desplegar, revertir y restaurar una capa después de que falle un proveedor. Para la cadena de herramientas se convierte en un conjunto de simulacros, cada uno con una cifra. ¿Se puede recompilar la versión del trimestre pasado desde una máquina limpia y obtener los mismos bytes? ¿Se puede firmar

con una clave guardada en hardware que uno controla, y revocarla? ¿Se puede detener un despliegue a mitad de camino y restaurar la versión anterior en las máquinas que ya recibieron la actualización? ¿Se puede instalar una nueva versión del compilador y publicar en un día si la anterior tiene una falla?

Hay que ejecutar cada simulacro una vez por trimestre y registrar el tiempo. Para una capa alquilada, la cifra honesta puede ser "hasta que el proveedor publique, más nuestro tiempo de despliegue"; también hay que anotarla. El objetivo no es que cada cifra sea pequeña. Es que cada cifra se conozca, para que un consejo de administración, un auditor o un cliente reciba una respuesta medida en lugar de una esperanza.

**EJEMPLO**

Una empresa regulada hipotética distribuye un cliente de escritorio a sus sucursales. Alquila el sistema operativo, el compilador y el framework de interfaz, y es dueña de la compilación, la firma y el servicio de actualización. Su primer simulacro descubre que una recompilación limpia tarda tres días, porque dos dependencias se descargan de un espejo público durante la compilación y un paso de firma necesita la computadora portátil de una persona concreta. Incorpora las dependencias al repositorio (vendoring), traslada la clave de firma a un módulo de seguridad de hardware con dos aprobadores, y el segundo simulacro tarda cuatro horas. Nada del producto cambió. Su exposición a cada falla upstream acaba de reducirse en días.

## Esta capa en la lista de materiales del píxel

| Componente | Quién lo escribe | Quién paga | Quién puede actualizarlo | Plazo de soporte | ¿Expuesto a entradas no confiables? | Su decisión |
|---|---|---|---|---|---|---|
| Compilador y cadena de herramientas del lenguaje (MSVC, GCC, Clang, rustc) | Proveedores y proyectos abiertos | Proveedores, patrocinadores, colaboradores corporativos | El proyecto; la empresa elige la versión | Calendario de versiones del proyecto | En tiempo de compilación, si compila código externo como dependencias o contribuciones; uno comprometido contamina cada compilación | Alquilar; fijar versiones y practicar la actualización en un día |
| Agentes de programación con IA en el editor y la cadena de compilación | El proveedor de la herramienta y el proveedor del modelo | La empresa, por suscripción | El proveedor | Los términos del proveedor | Sí: leen incidencias, páginas y archivos | Alquilar la herramienta; ser dueño de la revisión y los permisos |
| Sistema de compilación y espejo de dependencias | Su equipo, sobre herramientas abiertas | La empresa | La empresa | El suyo | Sí, a través de lo que descarga | Controlar (construir solo las piezas críticas para la confianza); hacer compilaciones reproducibles |
| Claves de firma y servicio de actualización | Su equipo | La empresa | La empresa | El suyo | Sí: el canal de actualización está expuesto a cada máquina | Controlar (construir solo las piezas críticas para la confianza); claves guardadas en hardware |
| Una capa diferenciadora (un proxy, un runtime o un renderizador) | Su equipo, quizá sobre código abierto | La empresa | La empresa | Mientras la publique | A menudo sí, por diseño | Construir o bifurcar, solo con un tiempo de recuperación probado |

LENTE DE SEGURIDAD

**¿Quién puede cambiarlo?** Todo el que tenga acceso de escritura al código fuente, las máquinas de compilación, las fuentes de dependencias, las claves de firma y el servicio de actualización de la empresa. Los agentes de IA que abren pull requests o editan archivos de compilación pertenecen a esa lista, con sus propias identidades y sin capacidad de fusionar ni firmar por sí solos.

**¿Qué tan rápido se puede corregir?** Tan rápido como una vuelta completa del ciclo de la figura 14.1, medida en un simulacro. La mayor parte de esa respuesta depende de la empresa; la revisión de las tiendas, la notarización y los compiladores que se alquilan añaden su propio tiempo. Acortar la parte que se controla mejora la respuesta para todas las capas alquiladas a la vez.

**¿Qué cambia la IA aquí?** Abarata escribir y migrar código, lo que hace más asequible una pequeña capa propia, y añade nuevos colaboradores con acceso de escritura, además de un flujo mayor de informes de errores que clasificar. La revisión, las compilaciones reproducibles, la procedencia y la firma se aplican al cambio de un agente exactamente igual que al de una persona.

IMPLICACIONES PARA EL NEGOCIO

**Seguridad.** Un **fabricante de software** debería tratar su sistema de compilación y firma como producción, con revisiones de acceso, claves guardadas en hardware y aprobación de publicación por dos personas. Este sistema permite que un atacante llegue a todos los clientes a la vez.

**Costo.** Un **operador de una flota de dispositivos** obtiene más de un servicio de actualización con despliegue escalonado y reversión que de ser dueño de cualquier otra capa. Requiere dedicar un equipo pequeño y se amortiza la primera vez que una corrección tiene que llegar a todos los quioscos en un día. Hay que presupuestar las personas, no solo las herramientas: ser dueño del ciclo implica guardias.

**Valor.** Una **empresa regulada** que puede mostrar una compilación reproducible, una versión firmada y un tiempo de recuperación probado para cada capa responde a las preguntas de auditoría con evidencia. Eso acorta auditorías y ciclos de venta, y convierte "¿de qué es dueña la empresa?" de una pregunta vaga en una lista con cifras.

**Cada trimestre:** medir una vuelta completa del ciclo de su cadena de herramientas, desde un cambio de una línea hasta ese cambio ejecutándose en cada máquina, y hacerla más corta que la anterior.

CAPÍTULO QUINCE

# Qué poseer y qué alquilar




Catorce capítulos han nombrado a los dueños que hay entre su código y la pantalla de su cliente. Ahora hay que decidir de cuáles de esas capas debería ser dueña la empresa. Este capítulo convierte el libro en un método: nueve preguntas para cada capa, cuatro opciones reales, una lista de materiales del píxel consolidada y un plan que se puede empezar el lunes. El método no recomienda poseer más. Indica dónde ser dueño de una capa acelera la corrección lo suficiente como para que valga su costo.

## Nueve preguntas para cada capa

La mayoría de los debates entre poseer y alquilar empiezan con una preferencia tecnológica y terminan en una pelea de presupuesto. Conviene empezar, en cambio, con las mismas nueve preguntas para cada fila de la lista. Cada una tiene una respuesta sencilla, y cada respuesta descarta algunas decisiones.

| Pregunta | Qué evalúa | Qué descarta una respuesta débil |
|---|---|---|
| **1. Diferenciación** | ¿Lo notaría un cliente, o pagaría más, si esta capa fuera mejor que la versión de sus competidores? | Si no, poseerla necesita otra razón: continuidad, soberanía o costo, o que no haya ninguna opción para alquilar. |
| **2. Riesgo** | ¿Qué le pasa al negocio si esta capa falla, se ve comprometida o desaparece? | Si la respuesta es "poco", no gastar nada más allá de alquilarla bien. |
| **3. Plazo de soporte** | ¿Cuándo dejan de llegar las actualizaciones, o vence una clave o un certificado? | Si el plazo termina antes de que termine la vida útil de su producto, alquilar necesita un plan de salida. |
| **4. Exposición** | ¿Puede llegar a este código un archivo, una página, un paquete, una imagen o una fuente tipográfica del exterior? | Si es así, poseerla significa ser dueño de un flujo de correcciones de seguridad. |
| **5. Plazo de parche** | Cuando se encuentra una falla, posiblemente con una herramienta de IA antes que el mantenedor, ¿cuánto tarda una corrección en ejecutarse en sus máquinas? | Poseer solo es aceptable con un riesgo residual que se haya evaluado y aceptado de forma explícita. Si la seguridad es la razón para poseer, debe dejar menos riesgo que alquilar: parchear con la misma fiabilidad, eliminar funciones o privilegios que no se necesitan, o seguir parcheando cuando el proveedor deja de dar soporte. |
| **6. Tiempo de recuperación probado** | Si esta capa se rompe en todas las máquinas a la vez, ¿cuánto tarda en recuperarse, medido en un simulacro y no estimado? | Si nunca se ha medido, no se sabe si se puede poseer. |
| **7. Talento** | ¿Puede la empresa contratar y retener al menos a dos personas que entiendan esta capa a fondo? | Una sola persona es un punto único de falla, no una capacidad. |
| **8. Costo de salida** | ¿Cuánto costaría, en dinero y en meses, pasar a otro dueño? | Un costo de salida alto en una capa alquilada es un riesgo que hay que gestionar, no ignorar. |
| **9. Responsable** | ¿Qué persona concreta de su empresa responde por esta capa cuando falla? | Si no hay nadie, ninguna decisión que se tome es real todavía. |

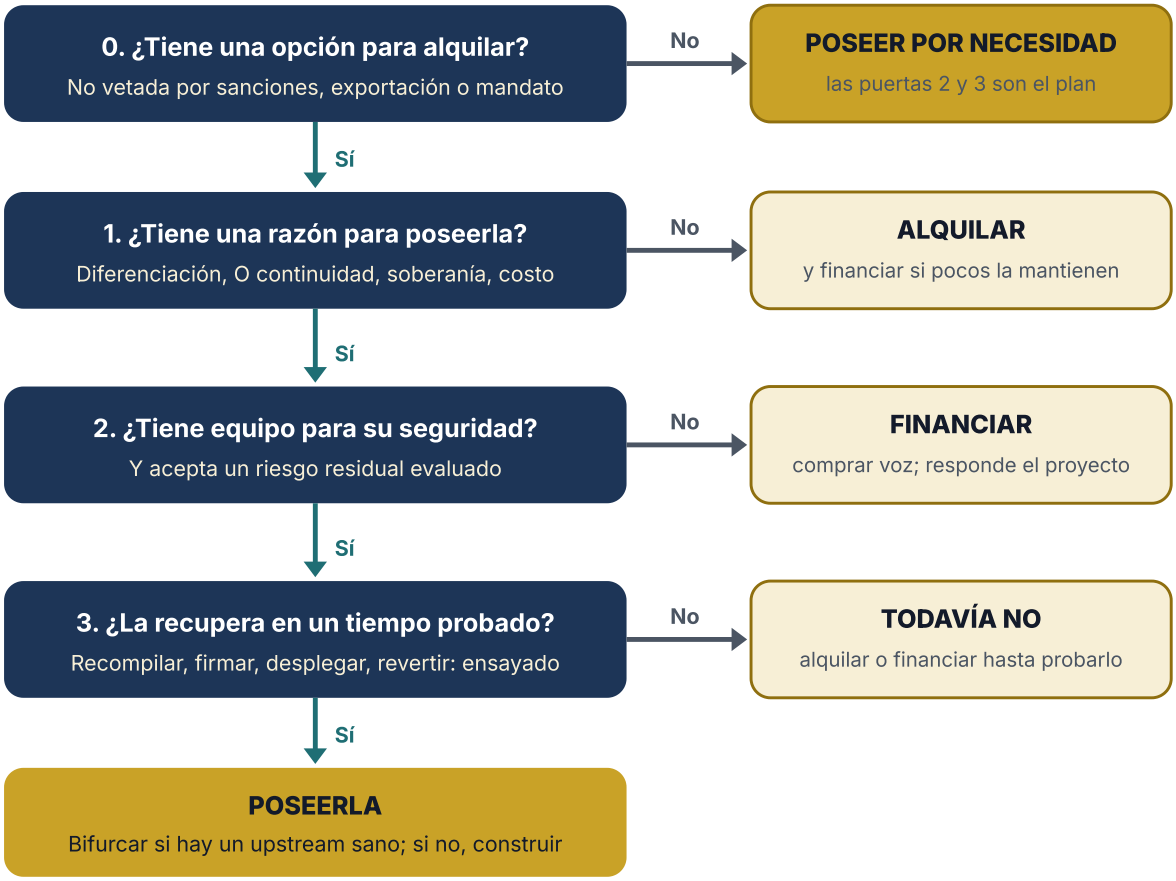
Las preguntas forman tres grupos. Las dos primeras preguntan si hay una razón para poseer la capa. Las preguntas tres a seis preguntan si se puede responder cuando se rompe. Las tres últimas preguntan si la empresa puede sostener la decisión durante años. En la primera puerta basta con cumplir una de las condiciones: poseer puede justificarse por diferenciación, **o** por continuidad, soberanía o costo, por ejemplo cuando el soporte de un proveedor termina antes que el producto, cuando quizá no se permita alquilar, o cuando alquilar cuesta más a lo largo de la vida del producto. Las dos puertas siguientes no son opcionales: sea cual sea la razón, solo hay que poseer una capa cuando se puede contar con personal para su respuesta de seguridad y su recuperación.

La puerta de seguridad no es una carrera contra el plazo de parche del dueño actual. Pide una evaluación explícita del riesgo que queda una vez que se posee la capa, y una decisión de que ese riesgo residual es aceptable. Cuando la razón para poseer es la diferenciación, la continuidad, la soberanía o el costo, basta con que sea aceptable. Cuando la razón es la propia seguridad, el riesgo residual debe ser menor que al alquilar. Poseer puede reducirlo parcheando más rápido, pero también eliminando funciones que no se usan, ejecutando la capa con menos privilegios o parcheándola después de que su proveedor deje de dar soporte.

Hay un caso que va antes de las puertas. Cuando no hay ninguna opción de alquiler disponible, porque controles de exportación, sanciones o un mandato de soberanía dejan sin acceso al proveedor, poseer no es una elección. El capítulo 12 mostró ese caso con HarmonyOS. Las otras dos puertas dejan entonces de ser una prueba y se convierten en el plan: contar con personal para la respuesta de seguridad y ensayar la recuperación antes de depender de la capa. La mayoría de las empresas medianas nunca se enfrentan a esta rama.

FIGURA 15.1

## El flujo de decisión para una capa




Una comprobación de necesidad y luego tres puertas, en orden. Si no hay ninguna opción de alquiler disponible, poseer es obligado y las puertas posteriores se convierten en el plan de personal. Si la hay, la primera puerta se supera con cualquier razón: diferenciación, o continuidad, soberanía o costo. La mayoría de las capas no pasan de esa puerta, y esa es la respuesta correcta para ellas. Una capa llega a "poseerla" solo cuando también supera las otras dos: su riesgo de seguridad residual se ha evaluado de forma explícita y es aceptable (y menor que al alquilar, si la seguridad es la razón para poseer), y se ha demostrado que se puede recuperar.

### Cuatro opciones, cuatro reglas

**Alquilar** significa usar la capa tal como la publica su dueño y aceptar la hoja de ruta del dueño. La regla: **alquilar cuando no hay ninguna razón de diferenciación, continuidad, soberanía o costo para poseer la capa, o cuando poseerla dejaría un riesgo residual que no se puede aceptar.** Eso cubre la mayor parte de la pila para la mayoría de las empresas: el kernel, el compositor, el controlador de la GPU, la vista web del sistema. Alquilar bien es una habilidad. Significa fijar versiones en las flotas, escalonar las actualizaciones, seguir las fechas de soporte y escribir cláusulas de salida en los contratos.

**Financiar** significa pagar a las personas que mantienen una capa abierta, directamente, a través de una fundación o a través de un proveedor que las emplea. La regla: **financiar cuando se depende de una capa que mantienen pocas personas y a la que pueden llegar entradas no confiables.** Financiar no compra control. Compra una voz, alertas más tempranas y un proyecto más sano. La puerta trasera de xz utils, revelada el 29 de marzo de 2024 y contada en detalle en el capítulo 8, mostró lo que una biblioteca de compresión con pocos mantenedores puede introducir en toda una distribución. Los gobiernos ahora también financian este trabajo: la Sovereign Tech Agency de Alemania gestiona el Sovereign Tech Fund, que invierte en los componentes de software abierto que, en sus palabras, sostienen la competitividad de Alemania y de Europa.

**Bifurcar (fork)** significa tomar una copia de una capa abierta y mantener en ella los cambios propios. La regla: **bifurcar solo cuando se necesitan cambios que el upstream no aceptará y se puede pagar la importación continua de las correcciones upstream durante todo el tiempo que se publique.** Un fork cuesta poco el primer año y más cada año siguiente, porque el upstream sigue avanzando. BoringSSL de Google, anunciado en junio de 2014, es el caso instructivo: tras años de mantener parches sobre OpenSSL en varias bases de código, Google pasó a importar cambios de OpenSSL en lugar de hacer rebase sobre ellos, y dijo claramente que no pretendía reemplazar a OpenSSL como proyecto de código abierto.

**Construir** significa escribir y poseer la capa. La regla: **construir solo cuando hay una razón para poseer la capa (diferenciación, o continuidad, soberanía o costo) o no hay opción de alquiler, cuando su riesgo de seguridad residual se ha evaluado de forma explícita y es aceptable (menor que al alquilar, si la razón es la seguridad), y cuando se tiene un tiempo de recuperación probado.** El proxy Pingora de Cloudflare, contado en el capítulo 14, es un desarrollo propio que superó esas puertas. Pocas empresas tienen ese perfil. Las pantallas del propio producto son la capa que la mayoría de las empresas debería construir; la ruta de publicación y las claves de firma son las capas que la mayoría debería controlar (capítulo 14).

> **CÓMO LLEGÓ A SER ASÍ**
>
> Las cuatro decisiones no son nuevas; lo que está en juego sí. BoringSSL en 2014, Edge en 2018 y Pingora en 2022 fueron tres grandes empresas dando tres respuestas distintas. Lo que ha cambiado desde entonces es el reloj: el capítulo 11 muestra que los tiempos de explotación bajaron de semanas a días, lo que da a las preguntas de plazo de parche y de recuperación más peso del que tenían.

| Capa | Windows | Linux | macOS |
|---|---|---|---|
| **Kernel** | Solo alquilar: Microsoft lo actualiza | Tomarlo de una distribución (alquilar), ajustarlo o bifurcarlo | Solo alquilar: Apple lo actualiza |
| **Cadena de arranque** | Alquilar: firmware del fabricante de la computadora, certificados de Microsoft | Alquilar shim y GRUB a una distribución; Microsoft firma shim | Solo alquilar: Boot ROM, LLB e iBoot de Apple |
| **Compositor** | Alquilar: DWM | Elegir: Mutter, KWin u otro; una flota controla la elección | Alquilar: WindowServer |
| **Vista web del sistema** | Alquilar WebView2, o empaquetar Chromium y controlar su ruta de parches | Tomarlo de la distribución (alquilar), o empaquetarlo y controlar la ruta de parches | En macOS de escritorio: alquilar WKWebView, o empaquetarlo y controlar la ruta de parches |
| **Lo que se puede poseer o controlar en los tres** | Poseer: su aplicación y sus pantallas de aprobación. Controlar: su ruta de compilación y publicación, sus claves de firma y su política. | | |

Estas reglas son para macOS de escritorio, no para iOS. En iOS y iPadOS las aplicaciones usan WebKit, salvo que, a noviembre de 2026, Apple permite otros motores con autorizaciones estrictas para usuarios de la UE (iOS 17.4 y posteriores) y de Japón (iOS 26.2 y posteriores).

## La lista de materiales del píxel completa

Aquí están, en un solo lugar, filas seleccionadas de los capítulos 1 a 8 y las tres filas de IA del capítulo 13, como inventario inicial y no como lista completa: los sistemas de archivos, los runtimes de C, los compiladores, los runtimes de aislamiento y otros también pertenecen a su copia, con las columnas finales. La lista se divide en dos para que siga siendo legible en una página. La primera tabla responde a **quién**: quién escribe, paga, actualiza y responde por cada componente. La segunda responde a **cuán expuesto**: el plazo de soporte, la exposición, el plazo de parche cuando la IA de otro encuentra primero una falla, el tiempo de recuperación probado y la decisión. Las columnas de parche y recuperación describen lo que normalmente fija el tiempo; sus propias cifras van en la copia en blanco al final de este capítulo.

| Capa | Componente | Quién lo escribe | Quién paga | Quién puede actualizarlo | Responsable |
|---|---|---|---|---|---|
| **Silicio** | Microcódigo de la CPU | Intel, AMD; Apple para sus chips | Incluido en el precio del chip | El fabricante del chip, a través del fabricante de la computadora o del sistema operativo | Responsable de la plataforma de endpoints |
| **Firmware** | Firmware UEFI | Proveedor de firmware más el fabricante de la computadora | Incluido en el precio del dispositivo | Solo el fabricante de la computadora | Responsable de la plataforma de endpoints |

| | | | | | |
|---|---|---|---|---|---|
| **Firmware** | Certificados de Secure Boot | Microsoft; el fabricante de la computadora para la PK | Microsoft | Microsoft, mediante actualizaciones de firmware y de Windows | Responsable de la plataforma de endpoints |
| **Kernel** | Kernel NT, Linux o XNU | Microsoft, la comunidad upstream, Apple | Licencias del proveedor; empresas que contribuyen | El proveedor del sistema operativo o la distribución; en Linux, también la empresa | Responsable de infraestructura |
| **Kernel** | Agente de seguridad de los equipos (endpoints) | El proveedor de seguridad | La empresa, por suscripción | El proveedor, a menudo muchas veces al día | Responsable de operaciones de seguridad |
| **Controladores** | Controladores de GPU y de red | Fabricantes de hardware, Mesa, Apple | El precio del hardware | El fabricante o el proveedor del sistema operativo | Responsable de la plataforma de endpoints |
| **Seguridad** | Avisos de consentimiento y de elevación | El proveedor del sistema operativo o el proyecto de escritorio | El proveedor del sistema operativo | Solo el proveedor del sistema operativo | Arquitecto de seguridad |
| **Seguridad** | Sus pantallas de aprobación | La empresa | La empresa | La empresa | Responsable de producto del flujo |
| **Runtimes** | .NET, Node.js en Electron, Python | Proveedor o proyecto abierto | Proveedor, patrocinadores, fundaciones | El proveedor del sistema operativo, o cada aplicación que empaqueta una copia | Responsable de ingeniería de cada aplicación |
| **Gráficos** | Compositor del sistema | El proveedor del sistema operativo o el proyecto de escritorio | El proveedor del sistema operativo, las distribuciones | El proveedor del sistema operativo o de la distribución | Responsable de la plataforma de endpoints |
| **Gráficos** | HarfBuzz, FreeType, decodificadores de imagen | Pequeños equipos abiertos | Sobre todo patrocinadores | Cada producto que empaqueta una copia | Arquitecto de seguridad |
| **Navegador** | Chromium empaquetado en su aplicación | El proyecto Chromium, el equipo de Electron | Google, patrocinadores, la empresa | La empresa, al publicar una versión | Responsable de ingeniería de cada aplicación |
| **IA** | Modelo alojado detrás de su agente o funcionalidad | El proveedor del modelo | La empresa, por uso | Solo el proveedor | Responsable de producto de la funcionalidad de IA |
| **IA** | Permisos y lista de herramientas del agente | Su equipo, sobre un framework alquilado | La empresa | La empresa | Arquitecto de seguridad |
| **IA** | Semántica de interfaz que leen los agentes (etiquetas, roles, árbol de accesibilidad) | Su equipo de aplicación, sobre el toolkit del sistema operativo | La empresa | La empresa, dentro de los límites del toolkit | Responsable de producto de las pantallas |
| **Cadena de herramientas (toolchain)** | Cadena de compilación (build pipeline) y claves de firma | Su equipo | La empresa | La empresa | Responsable de ingeniería de publicación |
| **Cadena de herramientas** | Ruta de tienda o de notarización | El dueño de la plataforma | El dueño de la plataforma | El dueño de la plataforma | Responsable de ingeniería de publicación |

| Componente | Plazo de soporte | ¿Expuesto a entradas no confiables? | Plazo de parche si la IA lo encuentra primero | Tiempo de recuperación probado | Decisión |
|---|---|---|---|---|---|

| | | | | | |
|---|---|---|---|---|---|
| **Microcódigo de la CPU** | La lista de soporte del fabricante del chip | Indirectamente, como mostró Spectre | El fabricante del chip, luego el fabricante de la computadora o el sistema operativo, luego la empresa | Revertir mediante firmware o sistema operativo; ensayarlo | Alquilar; seguirlo por modelo |
| **Firmware UEFI** | El soporte del fabricante de la computadora para el modelo | Sí: LogoFAIL usó un logotipo de arranque | Lo fija el fabricante de la computadora, luego su despliegue | Regrabar el firmware de cada dispositivo; a menudo, una visita a las instalaciones | Alquilar; comprar soporte largo |
| **Certificados de Secure Boot** | El conjunto de 2011 venció en junio y octubre de 2026 | No, pero deciden qué arranca | Actualizaciones de Microsoft y del fabricante de la computadora | Verificar que los certificados de 2023 estén registrados ( `db` y KEK) en cada modelo y luego probar el arranque con medios firmados por la CA de 2023, antes del despliegue | Alquilar; mantener el calendario |
| **Kernel NT, Linux o XNU** | Fechas de mantenimiento de cada versión | Sí | El ciclo del proveedor; en Linux, puede ser el suyo | Volver a instalar desde una imagen verificada | Alquilar; las flotas Linux pueden ajustarlo |
| **Agente de seguridad de los equipos** | Su contrato | Sí, por diseño | El proveedor, sin ventana de cambios | Arrancar en modo seguro dispositivo por dispositivo, salvo despliegue escalonado | Alquilar; exigir despliegue escalonado |
| **Controladores de GPU y de red** | El soporte del fabricante para el dispositivo | A menudo | El fabricante, luego el proveedor del sistema operativo, luego la empresa | Fijar y revertir a la versión probada | Alquilar; estandarizar modelos |
| **Avisos de consentimiento y de elevación** | La vida de soporte del sistema operativo | Indirecto: superposiciones e interfaz suplantada, no análisis sintáctico (parsing) | Solo el proveedor del sistema operativo | No es suyo; usar un segundo canal | Alquilar |
| **Sus pantallas de aprobación** | El suyo | Sí: superposiciones, captura, agentes | Su próxima versión | La reversión de su versión | Construir; vincular el monto y el beneficiario a la solicitud mediante la firma |
| **.NET, Node.js en Electron, Python** | Las ventanas LTS de cada proyecto | Sí | Un parche por cada copia empaquetada | La reversión de su aplicación | Alquilar; seguir cada copia |
| **Compositor del sistema** | La versión del sistema operativo | Indirectamente | El proveedor del sistema operativo | Reinstalar la imagen | Alquilar |
| **HarfBuzz, FreeType, decodificadores de imagen** | Las versiones del proyecto; la antigüedad de su copia | Sí: fuentes tipográficas e imágenes | Upstream, luego cada producto que lo empaqueta, por separado | La reversión de cada producto | Alquilar y financiar; documentar la exposición |
| **Chromium empaquetado en su aplicación** | Las tres últimas versiones mayores de Electron | Sí, si carga contenido externo | Upstream, Electron, luego su versión | La reversión de su aplicación | Alquilar; controlar la ruta de parches |
| **Modelo alojado detrás de su agente o funcionalidad** | El calendario de retirada de modelos del proveedor | Sí: prompts, documentos y páginas que lee | El plazo del proveedor; solo se puede limitar lo que el modelo puede hacer | Pasar a un modelo de respaldo probado o desactivar la funcionalidad; ensayarlo | Alquilar; pactar por contrato el aviso y las condiciones de datos |
| **Permisos y lista de herramientas del agente** | El suyo | Sí: cada página y archivo que lee el agente | Su próximo cambio de configuración, a menudo horas | Revocar la identidad y las herramientas del agente | Construir la política; alquilar el framework |

| | | | | | |
|---|---|---|---|---|---|
| **Semántica de interfaz que leen los agentes** | Su ciclo de publicaciones | Indirectamente: los agentes actúan según lo que dice | Su ciclo de publicaciones; el del proveedor del sistema operativo para el toolkit | La reversión de su versión | Construir; versionarla como una API |
| **Cadena de compilación y claves de firma** | El suyo | Sí, a través de las dependencias | El suyo | Recompilar y volver a firmar desde fuentes limpias; ensayarlo | Controlar (construir solo las piezas críticas para la confianza) |
| **Ruta de tienda o de notarización** | Las reglas de la plataforma | Indirectamente | El tiempo de revisión de la plataforma | Un canal alternativo donde se permita | Alquilar; mantener una alternativa |

Dos de las columnas nuevas merecen un comentario. **Plazo de parche si la IA lo encuentra primero** pregunta qué ocurre cuando una herramienta automatizada encuentra una falla antes de que el mantenedor sepa de ella. Eso ya no es hipotético: el capítulo 13 cuenta cómo el agente Big Sleep de Google encontró una falla de SQLite que unos atacantes se preparaban para usar. La columna pregunta por cuántas manos debe pasar una corrección antes de llegar a sus máquinas. **Tiempo de recuperación probado** pide una cifra salida de un simulacro. Ser dueño del código fuente no significa poder recompilar, firmar, desplegar y restaurar un sistema en funcionamiento después de que falle un proveedor. Solo un simulacro lo dice.

## Tres empresas, tres listas

EL FABRICANTE DE SOFTWARE

| DECISIÓN | CAPAS | POR QUÉ |
|---|---|---|
| POSEER | INTERFAZ DEL PRODUCTO Y FLUJOS DE APROBACIÓN | LOS CLIENTES LOS VEN, Y SOLO LA EMPRESA PUEDE CORREGIRLOS |
| CONTROLAR (CONSTRUIR SOLO LAS PIEZAS CRÍTICAS PARA LA CONFIANZA) | CADENA DE COMPILACIÓN SOBRE HERRAMIENTAS ALQUILADAS, CLAVES DE FIRMA, CANAL DE ACTUALIZACIÓN | CADA CORRECCIÓN QUE PUBLICA PASA POR ELLOS |
| FINANCIAR | LAS BIBLIOTECAS DE MODELADO DE TEXTO Y DE DECODIFICACIÓN DE IMÁGENES QUE EMPAQUETA | EQUIPOS DE MANTENEDORES REDUCIDOS; EXPUESTAS A ARCHIVOS DE CLIENTES |
| BIFURCAR | NINGUNA, EN PRINCIPIO | NINGÚN UPSTREAM RECHAZA LOS CAMBIOS QUE NECESITA |
| ALQUILAR | SISTEMA OPERATIVO, KERNEL, COMPOSITOR, CONTROLADORES DE GPU, CHROMIUM MEDIANTE ELECTRON O LA VISTA WEB DEL SISTEMA | LOS PROVEEDORES LOS PARCHEAN MÁS RÁPIDO DE LO QUE PODRÍA UN EQUIPO DE PRODUCTO |

Su decisión más delicada es el motor de navegador en el escritorio. Si empaqueta Chromium mediante Electron, ha alquilado el motor pero controla la ruta de parches: cada corrección de Chromium llega a los clientes solo cuando publica una versión. Si usa la vista web del sistema, renuncia al control de la versión y gana la velocidad de parche del proveedor. Ambas opciones pueden ser correctas. Lo que no es correcto es elegir por costumbre y descubrir la ruta de parches durante un incidente.

**LA EMPRESA REGULADA**

| DECISIÓN | CAPAS | POR QUÉ |
|---|---|---|
| POSEER | POLÍTICA DE ACCESO, VINCULACIÓN DE APROBACIONES PARA PAGOS Y FIRMAS, REGISTRO DE EVIDENCIAS, INVENTARIO | LOS AUDITORES PIDEN PRUEBAS, Y SOLO ELLA PUEDE PRODUCIRLAS |
| CONTROLAR (CONSTRUIR SOLO LAS PIEZAS CRÍTICAS PARA LA CONFIANZA) | CADENA DE COMPILACIÓN, CLAVES DE FIRMA Y SERVICIO DE ACTUALIZACIÓN PARA SUS PROPIAS APLICACIONES | CADA CORRECCIÓN QUE LLEGA A SUS APLICACIONES CLIENTE PASA POR ELLOS, Y LOS AUDITORES PREGUNTAN QUIÉN PUEDE FIRMAR UNA VERSIÓN |
| FINANCIAR | COMPONENTES ABIERTOS DE LOS QUE DEPENDE A TRAVÉS DE SU DISTRIBUCIÓN LINUX | UNA VOZ Y ALERTA TEMPRANA SOBRE SU EXPOSICIÓN |
| BIFURCAR | NINGUNA | UN FORK AÑADE SU PROPIO ALCANCE DE AUDITORÍA |
| ALQUILAR | SISTEMA OPERATIVO, FIRMWARE, AGENTE DE SEGURIDAD DE LOS EQUIPOS, AVISOS DE CONSENTIMIENTO, INICIO DE SESIÓN | ALQUILAR, CON CLÁUSULAS CONTRACTUALES SOBRE DESPLIEGUE ESCALONADO Y FECHAS DE SOPORTE |

Casi no posee código por debajo de sus aplicaciones, y sin embargo posee una gran responsabilidad. Controla su ruta de publicación como las otras dos. El resto de su trabajo es alquilar bien y demostrarlo: cláusulas de despliegue escalonado para cualquier proveedor que escriba en el kernel, un calendario de fechas de soporte y aprobaciones vinculadas a datos firmados para que una disputa no dependa de una captura de pantalla.

**EL OPERADOR DE UNA FLOTA DE DISPOSITIVOS**

| DECISIÓN | CAPAS | POR QUÉ |
|---|---|---|
| POSEER | SHELL DEL QUIOSCO Y PANTALLA DE PAGO | LOS CLIENTES LOS TOCAN EN CADA UBICACIÓN |
| CONTROLAR (CONSTRUIR SOLO LAS PIEZAS CRÍTICAS PARA LA CONFIANZA) | IMAGEN DEL SISTEMA OPERATIVO (COMPUESTA CON PIEZAS DEL PROVEEDOR O DE LA DISTRIBUCIÓN Y FIJADA), RUTA DE ACTUALIZACIÓN Y REVERSIÓN | UNA MALA ACTUALIZACIÓN LLEGA A TODAS LAS UBICACIONES A LA VEZ; DEBE CONTROLAR EL DESPLIEGUE |
| FINANCIAR | EL SOPORTE DE LA DISTRIBUCIÓN O DEL PROVEEDOR PARA SU IMAGEN DE LARGA VIDA | SUS MÁQUINAS SOBREVIVEN A LAS VENTANAS DE SOPORTE HABITUALES |
| BIFURCAR | NINGUNA, EN PRINCIPIO | COMPONER Y FIJAR UNA IMAGEN MÍNIMA NO ES UN FORK; BIFURCAR SOLO UN COMPONENTE CUYO UPSTREAM NO ACEPTE UN CAMBIO QUE NECESITA |
| ALQUILAR | FIRMWARE, KERNEL, CONTROLADORES DE GPU, COMPOSITOR, MOTOR DE NAVEGADOR | FIJAR VERSIONES PROBADAS; NO ACEPTAR ACTUALIZACIONES SIN PROBAR |

Es la única de las tres empresas para la que controlar más suele ser lo correcto, porque su mayor costo es una visita a las instalaciones y su mayor riesgo es una falla en toda la flota. Componer y fijar la imagen, y controlar la ruta de reversión, convierte una caída de toda la flota en una reversión. Aun así debería alquilar el kernel y los controladores, y fijarlos.

## El píxel de confianza: la decisión más delicada

El capítulo 4 expuso los modelos de amenaza del píxel de confianza y un control para cada uno. Para la lista, lo importante es que el píxel de consentimiento y de pago es una decisión dividida. El proveedor del sistema operativo suministra y controla el aviso del sistema, el escritorio seguro y la pantalla de inicio de sesión, y los sistemas de escritorio no ofrecen a las aplicaciones de terceros ninguna forma general de atestiguar lo que mostró la pantalla. La empresa es dueña de la pantalla de aprobación dentro de su producto y de lo que se firma cuando la persona presiona el botón. Así que conviene **alquilar** las superficies protegidas del proveedor donde existan, **construir** un flujo de aprobación que vincule el monto, el beneficiario y la identidad a la solicitud mediante la firma, confirmar las acciones irreversibles por un **segundo canal** y decidir de forma deliberada si los agentes que actúan en nombre de un usuario pueden aprobar algo.

> *No se puede ser dueño del cristal. Se puede ser dueño de lo que la solicitud firmada dice que la persona aprobó.*

### Un dueño para cada costura

Una **costura** es el lugar donde la capa de un dueño pasa el relevo a la de otro. Los incidentes viven ahí. En muchas organizaciones el camino del píxel se reparte entre varios equipos: ingeniería de endpoints es dueña de la imagen y los controladores, seguridad es dueña del agente, los equipos de aplicación son dueños de los motores empaquetados y otro grupo es dueño del quiosco o de la ficha en la tienda. Cada equipo puede ser competente y la costura puede seguir sin pertenecer a nadie.

La columna **responsable** lo resuelve. Nombra a una persona, por su rol, para cada fila. Esa persona no hace todo el trabajo. Responde a tres preguntas cuando la fila falla: a quién llamamos, qué puede corregir y cuánto tardará. Una fila sin responsable es un riesgo que la empresa ha aceptado sin decidirlo.

| Costura | Normalmente repartida entre | Nombrar un responsable para |
|---|---|---|
| Firmware a sistema operativo | Compras de hardware, ingeniería de endpoints | Fechas de soporte y calendario de certificados |
| Kernel a agente de seguridad | Operaciones de seguridad, ingeniería de endpoints | Despliegue escalonado y recuperación cuando falla una actualización |
| Aplicación a motor empaquetado | Cada equipo de producto | Contar las copias y publicar las correcciones upstream |
| Aplicación a pantalla de consentimiento | Producto, seguridad, legal | Qué se firma y qué se le mostró a la persona |
| Imagen a dispositivo en campo | Operaciones de campo, ingeniería de endpoints | Reversión sin visita a las instalaciones |

**CLÁUSULA DE EJEMPLO: DESPLIEGUE ESCALONADO (UNA PLANTILLA, NO ASESORAMIENTO LEGAL)**

La costura entre el kernel y un agente de seguridad falló a gran escala en julio de 2024 (capítulo 2). Este es un punto de partida en lenguaje sencillo para sus asesores legales; los valores entre corchetes los fija la empresa.

- **1. Etapas.** Las actualizaciones que se ejecutan con privilegios de kernel, firmware o administrador, incluidas las actualizaciones de contenido y de configuración, se publican en al menos [tres] etapas: la propia flota del proveedor, luego no más del [1] % de nuestros dispositivos, luego el resto, con al menos [24] horas de separación.
- **2. Nuestro control.** Podemos retrasar o pausar las actualizaciones por grupo de dispositivos. Una actualización de emergencia requiere aviso previo a nuestro contacto designado, con el motivo.
- **3. Reversión.** El proveedor detiene una etapa cuando fallan [señales de salud definidas] y ofrece una reversión remota, probada [una vez al año].
- **4. Remedio.** El incumplimiento de las cláusulas 1 a 3 nos da [créditos de servicio / derecho a rescindir].

### La economía en palabras sencillas

El **costo total de propiedad (TCO)** es todo lo que cuesta una capa a lo largo de su vida, no solo su precio. Para una capa alquilada incluye la licencia, el tiempo del personal para probar y desplegar actualizaciones, el trabajo de incidentes cuando falla, y la memoria y el disco que ocupa en cada máquina. Para una capa propia añade los ingenieros, su respuesta de seguridad, la infraestructura de compilación y firma, y el costo de mantener un fork al día. Hay que comparar ambos a lo largo de la vida del producto, no del primer año, porque los forks y las construcciones propias son más baratos al principio.

El **costo de cambio** es lo que cuesta pasar una capa a otro dueño: reescribir código, volver a capacitar al personal, volver a probar y migrar datos y dispositivos. Es el precio de un cambio en la hoja de ruta de un proveedor. Microsoft lo pagó en 2018 cuando pasó Edge a Chromium. La empresa lo paga cuando un runtime, un motor o una línea de dispositivos se descontinúa mientras todavía depende de él.

Las **opciones de salida** son los caminos de cambio que se preparan antes de necesitarlos: un segundo proveedor calificado, una cláusula contractual sobre fechas de soporte y depósito del código fuente (escrow), una exportación de datos probada, un canal de distribución alternativo. Una opción de salida es barata si se compra pronto y cara si se improvisa. Para cada fila alquilada con un costo de salida alto, hay que anotar la salida que se tomaría y probar una parte de ella cada año.

**ILUSTRACIÓN: UNA FLOTA DE QUIOSCOS A 7 AÑOS, PLAZO DE FIRMWARE FRENTE A PRECIO DE LISTA**

Esta es una ilustración con supuestos explícitos, no datos. Un **operador de una flota de dispositivos** planea 2000 quioscos de registro en 400 ubicaciones, para funcionar durante 7 años. El modelo A cuesta 900 USD e incluye 3 años de actualizaciones de firmware. El modelo B cuesta 1200 USD e incluye 7 años. Supongamos que si una falla de firmware necesita una corrección del proveedor después de que termine el soporte del modelo A, el operador reemplaza la flota antes de tiempo, a 900 USD por quiosco más una visita de 400 USD por ubicación.

| CONCEPTO | MODELO A | MODELO B |
|---|---|---|
| COMPRA, 2000 QUIOSCOS | 1 800 000 USD | 2 400 000 USD |
| LAS ACTUALIZACIONES DE FIRMWARE CUBREN | AÑOS 1 A 3 | AÑOS 1 A 7 |
| RENOVACIÓN ANTICIPADA SI SE NECESITA UNA CORRECCIÓN DE FIRMWARE EN LOS AÑOS 4 A 7 | 1 800 000 USD + 400 × 400 USD = 1 960 000 USD | NO ES NECESARIA: EL PROVEEDOR PUBLICA LA CORRECCIÓN |
| PROBABILIDAD A PARTIR DE LA CUAL SE IGUALAN LOS COSTOS | 600 000 USD ÷ 1 960 000 USD = ALREDEDOR DEL 31 % | |

Si se estima que la probabilidad de al menos una falla de firmware que necesite una corrección del proveedor en los años 4 a 7 supera el 31 % aproximadamente, el plazo de soporte más largo es la opción más barata. Los plazos de firmware sí se agotan dentro de la vida de los dispositivos: los certificados de Secure Boot de 2011 de la lista anterior vencieron en 2026. El modelo omite el valor de reventa, el beneficio de un hardware más nuevo y el costo de operar expuesto mientras se decide, así que conviene poner cifras propias en cada celda. Lo importante es el método: poner precio al plazo de soporte, no solo al precio de lista.

## La misma lista en una pila del lado del servidor

El método no es solo para escritorios y quioscos. Un servicio que se ejecuta en Kubernetes en una nube pública tiene los mismos tipos de dueños, con otros nombres: el plano de control gestionado de la nube hace el papel del proveedor del sistema operativo, el proveedor de identidad hace el papel de la pantalla de consentimiento y el servicio de gestión de claves (KMS) guarda las claves como lo hace su configuración de firma. El "píxel" es la página o la respuesta de API que recibe su cliente. Las filas de abajo usan las siete columnas de los capítulos; conviene añadir las tres columnas adicionales de la lista completa en la copia propia.

| Componente | Quién lo escribe | Quién paga | Quién puede actualizarlo | Plazo de soporte | ¿Expuesto a entradas no confiables? | Su decisión |
|---|---|---|---|---|---|---|
| Plano de control de la nube (servidor de API de Kubernetes gestionado) | El proveedor de nube, sobre Kubernetes upstream | La empresa, por hora de clúster | El proveedor; en Amazon EKS, actualiza él mismo el plano de control cuando termina el soporte extendido | En EKS: 14 meses de soporte estándar por versión, luego 12 meses de soporte extendido con costo adicional | Sí: la API está expuesta a sus pipelines y a su personal | Alquilar; actualizar según su calendario, antes de la actualización forzada |
| Nodos de trabajo y su imagen de sistema operativo | Una distribución o el proveedor | La empresa, como costo de cómputo | La empresa, en los grupos de nodos administrados y autoadministrados de EKS, que no se actualizan con el plano de control; con EKS Auto Mode, AWS reemplaza los nodos después de una actualización del plano de control | Las fechas de soporte de la imagen | Sí, a través de las cargas de trabajo que ejecutan | Alquilar las piezas; componer y fijar la imagen |
| Imágenes base de contenedores y dependencias de la aplicación | Proyectos abiertos y proveedores | Patrocinadores, proveedores | La empresa, al recompilar y volver a desplegar | Las versiones de cada proyecto | Sí: las solicitudes llegan a este código | Alquilar y financiar; recompilar con cada corrección upstream |
| Proveedor de identidad (personas y cargas de trabajo) | El proveedor de identidad | La empresa, por suscripción | El proveedor | Su contrato | Sí: el inicio de sesión está expuesto a internet | Alquilar; ser dueño de las políticas y de las cuentas de emergencia (break-glass) |

| | | | | | | |
|---|---|---|---|---|---|---|
| Servicio de gestión de claves | El proveedor de nube | La empresa, por clave y por solicitud | El proveedor para el servicio; la empresa para la política y la rotación de claves | Los términos de servicio del proveedor | Indirectamente: a través de quien pueda invocarlo | Alquilar; ser dueño de la política de claves y ensayar un día con una clave desactivada |
| Cadena de compilación, registro y despliegue | Su equipo, sobre herramientas alquiladas | La empresa | La empresa | El suyo | Sí, a través de las dependencias | Controlar (construir solo las piezas críticas para la confianza; capítulo 14) |

Dos datos de calendario muestran por qué la columna del plazo de soporte también importa aquí. El proyecto Kubernetes da soporte a cada versión menor durante unos catorce meses. Amazon EKS, a noviembre de 2026, ofrece 14 meses de soporte estándar y 12 meses de soporte extendido de pago por versión, y después actualiza el plano de control por su cuenta. AWS publica un aviso de la fecha de fin de soporte con al menos 60 días de antelación, pero dice que no puede dar un plazo para la actualización automática y no envía ninguna notificación antes de ella. Una actualización in situ puede revertirse en 7 días; una automática, no. Es una capa alquilada cuyo dueño la cambiará en lugar de la empresa, en una fecha que hoy ya se puede leer. Conviene ponerla en el calendario, nombrar a un responsable y medir en un simulacro cuánto tarda una actualización completa del clúster.

**LENTE DE SEGURIDAD**

**¿Quién puede cambiarlo?** En toda la lista, la mayoría de las filas las cambia otro: fabricantes de chips, fabricantes de computadoras, proveedores de sistemas operativos, proveedores de seguridad, proyectos abiertos. La empresa cambia su aplicación, sus flujos de aprobación, su política, su ruta de compilación y sus claves. La lista hace visible esa división, fila por fila.

**¿Qué tan rápido se puede corregir?** Tan rápido como la mano más lenta de la cadena. Una corrección en un decodificador compartido debe pasar del upstream a cada producto que lo empaqueta; una corrección de firmware pasa por el fabricante de la computadora; su propio código solo pasa por la empresa. Ser dueño de una capa ayuda a la seguridad cuando quita manos de esa cadena, elimina funciones o privilegios que no se necesitan, o mantiene las correcciones después de que un proveedor deja de dar soporte.

**¿Qué cambia la IA aquí?** Añade filas que la lista antigua no tenía: el modelo alojado, los permisos del agente y la semántica de interfaz que leen los agentes (capítulo 13). Y como ahora una falla puede encontrarla primero la herramienta de otro, las columnas de plazo de parche y de tiempo de recuperación son las primeras que hay que medir.

IMPLICACIONES PARA EL NEGOCIO

**Seguridad.** La **empresa regulada** es la que más gana con la columna del responsable: la pregunta de un auditor sobre el código del kernel o sobre las aprobaciones de pago tiene ahora una respuesta con nombre y un calendario. El **operador de una flota de dispositivos** es el que más gana con el tiempo de recuperación probado, porque su peor día es que fallen todas las ubicaciones a la vez.

**Costo.** El **fabricante de software** debería comparar el TCO a lo largo de la vida del producto antes de empaquetar un runtime o un motor más, ya que cada copia trae su propia ruta de parches. Financiar unas pocas bibliotecas abiertas con pocos mantenedores suele costar menos que un solo incidente en ellas.

**Valor.** Las tres empresas poseen la misma lista corta, las pantallas de su producto y su vinculación de aprobaciones, y controlan dos cosas más: su ruta de publicación y sus claves. Ahí es donde poseer cambia quién puede corregir un defecto, y con qué rapidez. Todo lo demás es un alquiler que hay que gestionar bien.

**La regla para la lista:** ninguna fila sin un responsable con nombre y una fecha de soporte.

## Su lista de materiales del píxel en blanco

Conviene copiar estas dos tablas para el propio producto y completar una fila por cada componente que esté entre su código y la pantalla de su cliente. Se empieza con las filas de este capítulo y luego se añade lo que sea específico de cada caso.

| Capa | Componente | Quién lo escribe | Quién paga | Quién puede actualizarlo | Responsable |
|---|---|---|---|---|---|
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |

| Componente | Plazo de soporte | ¿Expuesto a entradas no confiables? | Plazo de parche si la IA lo encuentra primero | Tiempo de recuperación probado | Decisión |
|---|---|---|---|---|---|
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |

## El plan del lunes

1. **Listar el camino y nombrar a sus responsables.** Para su producto principal, anotar cada componente entre su código y la pantalla del cliente, en cada sistema en el que publica, y dar a cada fila un responsable por rol.
2. **Poner las fechas en un calendario.** Fechas de fin de soporte, fechas de vencimiento de certificados y renovaciones de contratos, cada una con un recordatorio con un año de antelación.
3. **Ejecutar un simulacro de recuperación.** Romper, en un entorno de pruebas, la fila cuya falla más dolería, y anotar en la lista el tiempo de recuperación medido.
4. **Tomar una decisión al mes.** Aplicar las nueve preguntas a una fila, elegir alquilar, financiar, bifurcar o construir, y llevar la decisión y sus razones al equipo directivo.

EPÍLOGO

# La última capa es el ser humano




Este libro siguió una pulsación de tecla desde un interruptor en un teclado hasta un punto encendido. Se detuvo en el panel, porque ahí termina la pila. El camino no termina ahí. Todo camino del bit al píxel termina en un ojo humano, y el ojo es una capa que ningún proveedor entrega, que ninguna actualización parchea y que ningún contrato transfiere.

## La pantalla actúa sobre una persona

La luz puede hacer daño, y las normas ya lo dicen. Las Pautas de Accesibilidad para el Contenido Web (WCAG), en el criterio de conformidad 2.3.1, un requisito de nivel A, dicen que el contenido no debe destellar más de tres veces en un segundo salvo que los destellos queden por debajo de ciertos umbrales, porque los destellos pueden provocar convulsiones. Muchas pantallas reducen el brillo encendiendo y apagando su luz muy rápido (modulación por ancho de pulso), y TÜV Rheinland advierte que ese parpadeo puede provocar dolores de cabeza y migrañas en personas sensibles; se elige al comprar el hardware, así que debe figurar en los requisitos de compra. La norma de alarmas médicas IEC 60601-1-8 fija cómo se muestran las alarmas visuales en monitores que funcionan toda la noche junto a los pacientes. Y el derecho europeo trata la pantalla como lugar de trabajo desde la Directiva 90/270/CEE del Consejo, del 29 de mayo de 1990, cuyo anexo dice que "la imagen en la pantalla debe ser estable, sin destellos ni otras formas de inestabilidad". Para un **operador de una flota de dispositivos**, el brillo por la noche, lo que muestra un quiosco inactivo y si algo destella son decisiones de diseño con una persona del otro lado.

> **LA ÚNICA FILA QUE NO SE PUEDE ALQUILAR**
>
> Cualquier otra fila de la lista de materiales del píxel se puede alquilar, financiar, bifurcar o construir. Esta no. Ninguna lista de materiales cubre una retina. El deber de cuidado es la única capa que no se puede alquilar.

## Qué significa "cada píxel importa"

El lema de esta serie ha significado tres cosas en este libro, y cada una se apoyó en la anterior.

Primero, significó **control del renderizado**: quién dibuja el producto. Entre el código propio y la pantalla hay un entorno de ejecución, un motor de texto, un motor de navegador, un compositor, un controlador y una GPU, y cada uno tiene un dueño que puede cambiar lo que ve el cliente.

Segundo, significó **seguridad**. Quien puede entregar actualizaciones a esas capas puede cambiar código en las máquinas de los clientes, y muchas de las capas están expuestas desde fuera con un archivo, una fuente, una imagen o una página web. El camino del píxel es un camino de ataque, y ser dueño de una capa cambia quién puede corregirla y qué tan rápido.

Tercero, significó **confianza: lo que la persona vio es lo que el sistema hizo**. Un consentimiento, un pago o una firma no valen más que la pantalla en la que se dieron. El sistema puede probar mucho sobre la transacción y poco sobre el cristal.

El epílogo añade la razón de fondo de las tres. Cada píxel importa porque una persona lo está mirando.

> *Ser dueño de lo que se puede corregir. Alquilar aquello en lo que se puede confiar. Y nunca olvidar quién está mirando.*

APÉNDICES

APÉNDICE A

# Glosario

Estos son los términos que usa este libro, en palabras sencillas. Cada entrada indica el capítulo donde el término tiene más peso; ese capítulo da la historia completa, las fechas y las fuentes. Cuando una definición depende de una versión o de una fecha, está vigente a noviembre de 2026.

## A a C

| Término | Significado | Capítulo |
|---|---|---|
| Access control list, ACL (lista de control de acceso) | La lista asociada a cada objeto de Windows, como un archivo o una clave del registro, que indica qué usuarios y grupos pueden hacer qué con él. | Cap. 4 |
| Access token (token de acceso) | Un objeto del kernel de Windows que indica con qué usuario se ejecuta un proceso, a qué grupos pertenece y qué privilegios tiene. El kernel lo compara con la ACL. | Cap. 4 |
| Accessibility tree (árbol de accesibilidad) | La descripción estructurada de lo que hay en pantalla (roles, etiquetas, valores) que un sistema expone para los lectores de pantalla. Ahora también la leen los agentes de IA. | Cap. 13 |
| Accountable owner (responsable) | La única persona, identificada por su rol, que responde por una fila de la lista de materiales del píxel cuando falla: a quién llamar, qué puede corregir y cuánto tarda. | Cap. 15 |
| APFS | Apple File System, el sistema de archivos predeterminado de macOS desde High Sierra (2017). Diseñado para almacenamiento flash, con cifrado, instantáneas y clones de archivos baratos. | Cap. 3 |
| AppArmor | Un módulo de seguridad de Linux que confina programas según la ruta de los archivos. Se integró en el kernel principal en Linux 2.6.36 (2010). | Cap. 4 |
| ArkWeb | El motor web de HarmonyOS, que según su documentación se desarrolla sobre Chromium. Cada versión del sistema operativo está ligada a una versión principal de Chromium. | Cap. 12 |
| Blink | El motor de renderizado del proyecto Chromium, bifurcado de WebKit por Google en abril de 2013. | Cap. 7 |
| Boot ROM (ROM de arranque) | Código de arranque fijado en el chip al fabricarlo. En Apple silicon es la raíz de confianza del hardware, y nunca se puede actualizar. | Cap. 1 |
| Chromium | El proyecto de navegador de código abierto, liderado por Google, cuyos motores Blink y V8 se ejecutan dentro de Chrome, Edge, WebView2 y Electron. | Cap. 7 |

| | | |
|---|---|---|
| CNA | CVE Numbering Authority (autoridad de numeración de CVE): una organización autorizada para asignar registros CVE. El proyecto del kernel Linux se convirtió en una en febrero de 2024. | Cap. 11 |
| Compositor (compositor) | El programa del sistema que es dueño de toda la pantalla y combina el búfer de cada ventana en un solo fotograma, la última etapa del pipeline gráfico: DWM, Mutter o KWin, WindowServer. | Cap. 6 |
| Computer use (uso de la computadora por agentes) | Un modelo de IA que lee capturas de pantalla, mueve el cursor, hace clic y escribe. Anthropic lo lanzó en beta pública el 22 de octubre de 2024. | Cap. 4, Cap. 13 |
| Container (contenedor) | Un proceso con su propia vista de archivos, red e identificadores de proceso, construido con mecanismos del kernel. Todos los contenedores de un host comparten un solo kernel. | Cap. 8 |
| Control (controlar) | La palabra de este libro para decidir la composición, la firma y el despliegue de partes que uno no escribió, como una imagen de sistema operativo compuesta y fijada, o la cadena de publicación propia sobre servicios de compilación alquilados. Se es dueño de una capa cuando se bifurca o se construye su código; se controla cuando se decide cómo se ensambla y se entrega. Un proveedor controla lo que suministra, como el aviso de consentimiento del sistema operativo. En las listas, la decisión se lee "Controlar (construir solo las piezas críticas para la confianza)". | Resumen, Cap. 11, Cap. 14, Cap. 15 |
| CVE | Common Vulnerabilities and Exposures: un registro público y un identificador para una falla de seguridad conocida, como CVE-2023-4863. | Cap. 6, Cap. 11 |
| Cyber Resilience Act (Reglamento de Ciberresiliencia, CRA) | Reglamento (UE) 2024/2847, en vigor desde el 10 de diciembre de 2024. Exige para los productos dentro de su alcance un periodo de soporte de al menos cinco años, salvo que el uso previsto sea más corto. | Cap. 12 |

## D a G

| Término | Significado | Capítulo |
|---|---|---|
| db, dbx, KEK y PK | Las bases de datos de claves de Secure Boot: firmantes de confianza (db), firmantes e imágenes revocados (dbx), las claves autorizadas para actualizarlas (KEK) y la Platform Key (PK) del fabricante de la computadora. | Cap. 1 |
| Decoder (decodificador) | Código que convierte un formato de archivo, como una imagen, una fuente o un video, en píxeles o glifos. Hace el análisis sintáctico (parsing) de datos de desconocidos, por eso los atacantes apuntan a él. | Cap. 6 |
| Decoder reachability (exposición de los decodificadores) | Si bytes no confiables pueden llegar a un decodificador concreto en una máquina. El capítulo 11 lo mide con números separados: copias físicas (C), instalaciones (I), dueños (O), instalaciones expuestas (R), el retraso máximo en días transcurridos (L) y días-copia (E). | Cap. 6, Cap. 11 |

| | | |
|---|---|---|
| Deemed export (exportación presunta) | Según las normas de exportación de Estados Unidos (15 CFR 734.13), entregar tecnología o código fuente controlados a una persona extranjera dentro de Estados Unidos. La inspección visual, como compartir pantalla, puede ser una entrega (15 CFR 734.15(a)(1)). | Cap. 4 |
| Deepfake | Video o audio sintético que imita a una persona real. Una videollamada de enero de 2024 con colegas falsificados mediante deepfake llevó a 15 transferencias por un total de HK$200 millones. | Cap. 4, Cap. 13 |
| DRM y KMS | El Direct Rendering Manager del kernel Linux, que gestiona la memoria y los trabajos de la GPU, y Kernel Mode Setting, que fija el modo de pantalla y alterna los fotogramas terminados. | Cap. 6 |
| Driver (controlador) | El código que traduce entre el kernel y una pieza de hardware. Tradicionalmente se ejecuta en el kernel, así que un solo controlador defectuoso puede hacer caer la máquina. | Cap. 3 |
| DriverKit | El framework de Apple, presentado con macOS Catalina en 2019, cuando las extensiones del kernel fueron declaradas obsoletas (deprecated), para controladores que se ejecutan en el espacio de usuario como extensiones de controlador en lugar de en el kernel. | Cap. 2, Cap. 3 |
| DWM | Desktop Window Manager, el compositor de Windows. Está siempre activo desde Windows 8. | Cap. 6 |
| Dynamic linking, PSD2 (vinculación dinámica) | La norma de pagos de la UE según la cual al pagador se le muestran el monto y el beneficiario, y el código de autenticación queda ligado a ellos, de modo que cualquier cambio lo invalida. | Cap. 4 |
| eBPF | Programas pequeños que se cargan en el kernel Linux en tiempo de ejecución y que un verificador revisa antes de ejecutarlos; se usan para filtrar paquetes y para observabilidad. | Cap. 3 |
| EEVDF | Earliest Eligible Virtual Deadline First, el planificador de Linux desde Linux 6.6 (octubre de 2023), que reemplazó al Completely Fair Scheduler. | Cap. 2 |
| eIDAS 2.0 | Reglamento (UE) 2024/1183, en vigor desde el 20 de mayo de 2024, que establece la Cartera Europea de Identidad Digital. | Cap. 4 |
| Electron | Un framework que empaqueta (bundle) el motor Chromium y el runtime Node.js en el binario de cada aplicación, así que cada aplicación Electron lleva su propio motor. | Cap. 5, Cap. 7, Cap. 11 |
| Evergreen, WebView2 (modo siempre actualizado) | El modo de WebView2 en el que todas las aplicaciones comparten un runtime que se actualiza automáticamente, con las mismas actualizaciones que el navegador Edge. | Cap. 7 |
| Exception levels (niveles de excepción) | Los niveles de privilegio de Arm: EL0 para las aplicaciones, EL1 para el kernel, EL2 para un hipervisor y EL3 para el firmware de mayor confianza. | Cap. 1 |

| | | |
|---|---|---|
| Firmware | El programa guardado en un chip de la placa base que despierta el hardware y entrega el control a un cargador del sistema operativo. | Cap. 1 |
| Fixed Version, WebView2 (versión fija) | El modo de WebView2 en el que una aplicación entrega su propia copia empaquetada del runtime, que nunca se actualiza salvo que la aplicación entregue una nueva. | Cap. 7 |
| Fork upkeep (mantenimiento de la bifurcación) | El trabajo recurrente de mantener al día una copia modificada de un proyecto original: fusionar, retroportar (backport) correcciones, recompilar y volver a probar cada versión del proyecto original, para siempre. | Cap. 12, Cap. 15 |
| gVisor | Un sandbox de contenedores de Google, escrito en Go, cuyo kernel de aplicación (el Sentry) atiende él mismo las llamadas al sistema de un contenedor. | Cap. 8, Cap. 14 |

## H a M

| Término | Significado | Capítulo |
|---|---|---|
| HarfBuzz | El motor abierto de modelado de texto (shaping) que usan muchos navegadores y escritorios Linux. | Cap. 6 |
| Hybrid kernel (kernel híbrido) | Un kernel que toma ideas del microkernel pero ejecuta sus servicios en modo kernel. A Windows NT se le suele llamar así; Apple llama así a XNU. | Cap. 2 |
| Kernel (núcleo) | El núcleo del sistema operativo, que se ejecuta con todos los privilegios del hardware. Ejecuta procesos, gestiona la memoria, planifica el trabajo y aloja los controladores. | Cap. 2 |
| KEV | El catálogo Known Exploited Vulnerabilities de CISA: fallas que la agencia de ciberseguridad de Estados Unidos sabe que se usan en ataques reales. | Cap. 6, Cap. 11 |
| libwebp | La biblioteca abierta que decodifica imágenes WebP. Su falla CVE-2023-4863 llegó a navegadores, aplicaciones Electron y bibliotecas de imágenes en septiembre de 2023. | Cap. 6, Cap. 7, Cap. 11 |
| LogoFAIL | Fallas, presentadas en diciembre de 2023, en el análisis sintáctico (parsing) de imágenes que hace el firmware UEFI para mostrar un logotipo al arrancar. Se ejecutan demasiado pronto para que Secure Boot las detenga. | Cap. 1 |
| Memory safety (seguridad de memoria) | Una propiedad de un lenguaje o de un programa que impide leer y escribir fuera de la memoria que debe tocar, por ejemplo más allá de un búfer o después de liberarla. | Cap. 5 |
| Microcode (microcódigo) | Pequeños programas internos que ejecutan muchas instrucciones del procesador. El fabricante del chip puede cambiarlo después de la venta, y se carga en cada arranque. | Cap. 1 |

| | | |
|---|---|---|
| Microkernel (micronúcleo) | Un kernel que mantiene solo un núcleo pequeño en modo kernel y ejecuta los demás servicios como programas separados, de modo que un servicio que falla se puede aislar, a costa de más cruces entre programas. La seguridad y la velocidad reales dependen del diseño. | Cap. 2 |
| MicroVM (micro máquina virtual) | Una máquina virtual pequeña con su propio kernel y solo los dispositivos que necesita. Firecracker de AWS, anunciado en noviembre de 2018, es el ejemplo principal. | Cap. 8 |
| MMU | Memory management unit (unidad de gestión de memoria): el hardware que traduce las direcciones virtuales de cada programa a memoria real, con tablas que escribe el kernel. | Cap. 1 |
| Monolithic kernel (kernel monolítico) | Un kernel que ejecuta su núcleo y sus servicios, incluidos los sistemas de archivos, la red y los controladores, como un solo programa en modo kernel. Linux es uno, con módulos cargables. | Cap. 2 |

## N a R

| Término | Significado | Capítulo |
|---|---|---|
| OWASP Top 10 for LLM Applications | Una lista ordenada de riesgos para aplicaciones construidas sobre modelos de lenguaje. La inyección de prompts ocupa el primer lugar en las ediciones de 2025 y de 2026. | Cap. 13 |
| Patch lag (retraso de parches) | Los días entre una corrección integrada (commit) en el proyecto original y el momento en que el último dispositivo de la flota la ejecuta. Cada dueño de la cadena lo aumenta. | Cap. 11 |
| Pixel Bill of Materials (lista de materiales del píxel) | La lista de este libro con cada componente entre el código propio y la pantalla del cliente, con quién lo escribe, quién paga, quién puede actualizarlo, su plazo de soporte, su exposición y la decisión que se toma. | Prólogo, Cap. 15 |
| Pluton | Un procesador de seguridad de Microsoft integrado en la CPU junto con AMD, Intel y Qualcomm, anunciado en noviembre de 2020. Su firmware se actualiza mediante Windows Update. | Cap. 1 |
| Prompt injection (inyección de prompts) | Esconder instrucciones en contenido que lee un modelo, como una página web, un correo o un documento, para que el modelo las siga. OWASP dice que no está claro si algún método lo impide por completo. | Cap. 13 |
| Rasterization (rasterización) | Convertir contornos de glifos, u otras formas, en píxeles a un tamaño dado. | Cap. 6 |
| Recall | Una función de Windows en los equipos Copilot+ PC, anunciada en mayo de 2024, que guarda instantáneas periódicas de la pantalla para que el usuario pueda buscar lo que vio. Se rediseñó para ser opcional y cifrada. | Cap. 4, Cap. 13 |

| | | |
|---|---|---|
| Release path (ruta de publicación) | La cadena de compilación (build pipeline), las claves de firma y el servicio de actualización propios: la ruta por la que el código y cada corrección empaquetada llegan a las máquinas de los clientes. Su decisión en cada lista es Controlar: se ejecuta sobre herramientas alquiladas y se construyen solo las piezas críticas para la confianza. | Prólogo, Cap. 11, Cap. 14, Cap. 15 |
| Rent, fund, fork, build (alquilar, financiar, bifurcar, construir) | Las cuatro opciones para cada capa: usarla tal como se entrega, pagar a sus mantenedores para tener voz, mantener una copia propia o escribir una propia. | Prólogo, Cap. 15 |
| Reproducible build (compilación reproducible) | Una compilación en la que cualquier parte, con el mismo código fuente, entorno e instrucciones, puede recrear artefactos idénticos bit a bit. | Cap. 8, Cap. 14 |
| Ring 0 y ring 3 (anillo 0 y anillo 3) | Niveles de privilegio de x86. El kernel se ejecuta en el anillo 0, el modo de CPU con más privilegios, aunque los hipervisores, el firmware y las protecciones del hardware todavía pueden limitarlo; las aplicaciones se ejecutan en el anillo 3. | Cap. 1 |
| Runtime (entorno de ejecución) | El código que debe estar en la máquina para que un programa se ejecute, aunque el equipo no lo haya escrito: una biblioteca de C, un runtime administrado como la JVM o .NET, o una plataforma como Electron. | Cap. 5 |
| Rust | Un lenguaje de sistemas con seguridad de memoria, sin recolector de basura y sin máquina virtual. Aun así tiene un pequeño runtime propio. Rust 1.0 se anunció en mayo de 2015. | Cap. 2, Cap. 5, Cap. 14 |

## S a T

| Término | Significado | Capítulo |
|---|---|---|
| SBOM | Software bill of materials (lista de materiales del software): una lista formal de los componentes dentro de un software. Dice qué está instalado, no a qué código pueden llegar las entradas no confiables. | Cap. 6, Cap. 8 |
| Seam (costura) | El lugar donde la capa de un dueño entrega el trabajo a la capa de otro dueño. Ahí viven los incidentes. | Cap. 11, Cap. 15 |
| Seam tax (sobrecosto de las costuras) | El nombre que da este libro al costo de una pila con muchos dueños, contado en copias, canales de actualización, retraso de parches y exposición. | Cap. 11 |
| Secure Boot | La función de UEFI que comprueba la firma de cada cargador de arranque contra las bases de datos de claves antes de ejecutarlo. Obligatoria para las computadoras certificadas para Windows 8 en 2012. | Cap. 1 |
| Secure desktop (escritorio seguro) | Un escritorio de Windows separado, usado de forma predeterminada para los avisos de UAC, al que solo pueden acceder procesos de Windows. Otros programas no pueden dibujar en él ni enviarle entradas. | Cap. 4 |

| | | |
|---|---|---|
| Secure Enclave | Un procesador de seguridad separado en los chips de Apple, con su propia ROM de arranque y su propio sistema operativo (sepOS), que guarda claves y datos biométricos. | Cap. 1, Cap. 4 |
| SELinux | Un módulo de seguridad de Linux que etiqueta cada proceso y cada archivo y comprueba cada acceso contra una política central. Publicado por la NSA en diciembre de 2000. | Cap. 4 |
| Shaping (modelado de texto) | Convertir caracteres en los glifos correctos, en el orden y la posición correctos: unir letras árabes, reordenar vocales índicas, colocar marcas. | Cap. 6 |
| shim | El pequeño cargador de arranque de Linux de primera etapa que firma Microsoft. Lleva la clave de la distribución y comprueba GRUB, que comprueba el kernel. | Cap. 1 |
| SIP | System Integrity Protection (protección de la integridad del sistema): la función de macOS, presentada en 2015, que impide incluso al usuario root cambiar archivos protegidos del sistema. | Cap. 4 |
| Spectre y Meltdown | Fallas de procesador en la ejecución especulativa, publicadas el 3 de enero de 2018. Corregirlas exigió a fabricantes de chips, kernels, compiladores y navegadores. | Cap. 1 |
| Support clock (plazo de soporte) | La fecha en la que un componente deja de recibir actualizaciones, o en la que vence una clave o un certificado. | Resumen, Cap. 1, Cap. 15 |
| TCC | Transparency, Consent and Control (transparencia, consentimiento y control): la capa de macOS que pregunta al usuario antes de que una aplicación pueda usar la cámara, el micrófono, carpetas protegidas o la grabación de pantalla. | Cap. 4 |
| Tested recovery time (tiempo de recuperación probado) | El tiempo medido, en un simulacro que alguien realmente ejecutó, para recompilar, firmar, desplegar, revertir y restaurar una capa después de que falla un proveedor. | Cap. 8, Cap. 14, Cap. 15 |
| Time-to-exploit (tiempo hasta la explotación) | La métrica de Mandiant: "el tiempo promedio que se tarda en explotar una vulnerabilidad antes o después de que se publique un parche", promediado entre días cero y días n. Mandiant midió 5 días para 2023, excluyendo 15 valores atípicos (47 con ellos). | Cap. 11, Cap. 13 |
| Toolchain (cadena de herramientas) | Todo lo que hay entre la idea de un desarrollador y un binario en ejecución: compilador, editor y sus asistentes, compilación, empaquetado, firma, publicación y observación. | Cap. 8, Cap. 14 |
| TPM | Trusted Platform Module: un chip de seguridad, o una función protegida del procesador, que guarda claves y registra lo que arrancó. Windows 11 exige TPM 2.0. | Cap. 1 |

| | | |
|---|---|---|
| Trusted Pixel (píxel de confianza) | El nombre que da este libro a la aplicación del principio WYSIWYS a todo el camino del píxel: lo que la persona vio debe ser lo que el sistema hizo. En los sistemas de escritorio, las aplicaciones de terceros no tienen una forma general de certificar lo que mostró la pantalla; los registros prueban la transacción, no lo que vio la persona. | Cap. 4, Cap. 15 |

## U a Z

| Término | Significado | Capítulo |
|---|---|---|
| UAC | User Account Control (control de cuentas de usuario): desde Windows Vista, da a un administrador un token filtrado para el trabajo diario y pide aprobación antes de usar el completo. | Cap. 4 |
| UEFI | Unified Extensible Firmware Interface: el estándar de firmware actual en computadoras y servidores, mantenido por el UEFI Forum desde 2005. | Cap. 1 |
| UI semantics (semántica de la interfaz) | Los nombres, roles y orden de los controles en una pantalla. Como los agentes ahora dependen de ellos, este libro los trata como una superficie de integración con versiones. | Cap. 13 |
| Update channel (canal de actualización) | Cualquier ruta por la que llega código nuevo a una máquina: el actualizador del sistema operativo, un servicio de un proveedor, una tienda de aplicaciones o un actualizador dentro de una aplicación. | Cap. 11 |
| Update topology (topología de actualización) | La forma de la ruta de parches: cuántas copias de un componente hay en una máquina y quién debe actuar antes de que cada copia quede corregida. | Cap. 7 |
| V8 | El motor de JavaScript y WebAssembly de código abierto de Google, escrito en C++, que usan Chromium, Node.js y workerd de Cloudflare. | Cap. 7, Cap. 14 |
| Wayland | El protocolo de pantalla de Linux en el que el compositor es el servidor de pantalla y las aplicaciones le entregan búferes terminados. La versión 1.0 se anunció en octubre de 2012. | Cap. 6 |
| WDDM | Windows Display Driver Model, presentado con Vista. Divide el controlador de la GPU en partes de modo usuario y de modo kernel, y puede reiniciar un controlador colgado. | Cap. 3, Cap. 6 |
| WebGPU | Una interfaz web que da a las páginas acceso moderno a la GPU, activada por defecto en Chrome 113. Es una superficie de ataque nueva que llega al controlador de la GPU. | Cap. 7 |
| WebKit | El motor de navegador de Apple, que empezó como una bifurcación de KHTML de KDE para Safari en 2003. En iOS, fuera de la UE y de Japón, todo navegador debe usarlo. | Cap. 7 |
| Web view (vista web) | Un control que muestra contenido web dentro de la ventana de una aplicación nativa: WebView2 en Windows, WKWebView en los sistemas de Apple, WebKitGTK o Qt WebEngine en Linux. | Cap. 7 |

| | | |
|---|---|---|
| Windows Resiliency Initiative | El plan de Microsoft, anunciado en noviembre de 2024, que incluye permitir que los productos de seguridad se ejecuten fuera del modo kernel. | Cap. 2 |
| WOW64 | La capa de Windows que ejecuta aplicaciones de 32 bits en Windows de 64 bits. No ejecuta aplicaciones de 16 bits. | Cap. 10 |
| WYSIWYS | "What you see is what you sign" (lo que se ve es lo que se firma): el principio establecido de las firmas electrónicas, analizado por Landrock y Pedersen en 1998, según el cual quien firma debe ver exactamente lo que firma. El píxel de confianza lo aplica a aprobaciones y pagos; el libro no lo inventó. | Cap. 4 |
| XNU | El kernel de macOS: un híbrido del kernel Mach de Carnegie Mellon, componentes de FreeBSD y la API de controladores IOKit. | Cap. 2 |
| xz backdoor (puerta trasera de xz) | Una puerta trasera en los tarballs de las versiones 5.6.0 y 5.6.1 de la biblioteca de compresión xz, revelada el 29 de marzo de 2024 antes de que llegara a la mayoría de las distribuciones estables. | Cap. 8 |
| Zero-day (día cero) | Una falla explotada antes de que exista una corrección. Cerca del 70 % de las 138 fallas del análisis de Mandiant de 2023 fueron días cero. | Cap. 11, Cap. 13 |

APÉNDICE B

# Línea de tiempo

Setenta años de la pila en dos figuras. Cada fecha de este póster aparece en un capítulo de este libro, donde se cuenta completa y con sus fuentes. Solo se dibujan los años con algún evento, así que el espaciado vertical no está a escala.

FIGURA B.1

## De 1956 a 1998: un solo proveedor, luego un mercado de capas

| | Sistemas operativos | Navegadores y runtimes | Eventos de seguridad | IA y seguridad |
|---|---|---|---|---|
| 1956 | GM-NAA I/O en el IBM 704 | | | Sin entradas antes de 2017 |
| 1961 | CTSS se presenta en el MIT | | | |
| 1964 | IBM System/360 | | | |
| 1965 | Artículos de Multics | | | |
| 1969 | Unix nace en Bell Labs<br>IBM separa el software | | | |
| 1972 | Se anuncia IBM VM/370 | | | |
| 1973 | Unix reescrito en C | | | |
| 1974 | Demostración de CP/M | | | |
| 1981 | IBM PC con PC DOS | | | |
| 1983 | | Sockets de 4.2BSD | | |
| 1984 | | | Conferencia Trusting Trust | |
| 1985 | Windows 1.0<br>Project Mach nace en CMU | | | |
| 1986 | Artículo de Mach en USENIX | | | |
| 1987 | OS/2 anunciado; sale la 1.0<br>X Window System versión 11 | GCC 1.0 | | |
| 1988 | | POSIX, IEEE 1003.1 | | |
| 1990 | Windows 3.0 | | | |
| 1991 | Se anuncia Linux | | | |
| 1992 | Linux pasa a la GPL<br>Plan 9, primera edición | UTF-8, diseñado para Plan 9 | | |
| 1993 | Windows NT 3.1 | Sale Mosaic | | |
| 1995 | Windows 95 | | PAM propuesto, OSF RFC 86.0 | |
| 1996 | Apple archiva Copland<br>Apple comprará NeXT, no Be | Java 1.0 | | |
| 1998 | Intel Boot Initiative | Netscape abre su código | | |

Del primer sistema operativo escrito por un cliente a la pila del PC y el kernel libre. Detalles en los capítulos 1 a 10 y 12.

FIGURA B.2

## De 2000 a 2026: vuelve la integración y el ritmo se acelera

| | Sistemas operativos | Navegadores y runtimes | Eventos de seguridad | IA y seguridad |
|---|---|---|---|---|
| 2000 | IBM: $1000 millones a Linux | | La NSA publica SELinux | |
| 2001 | Mac OS X 10.0<br>Be vende activos a Palm | | | |
| 2002 | | .NET Framework | | |
| 2003 | Artículo de Xen en SOSP | Safari sobre WebKit | | |
| 2004 | | Firefox 1.0 | | |
| 2005 | Nace el Unified EFI Forum<br>IBM retira OS/2 | | | |
| 2006 | Amazon EC2 en beta | | | |
| 2007 | iPhone; se anuncia Android<br>Vista: WDDM, UAC | | | |
| 2008 | Abre la App Store | Chrome en beta | | |
| 2010 | Se presenta Windows Phone 7<br>Llega systemd | | AppArmor entra al kernel | |
| 2012 | Wayland 1.0<br>Apple se disculpa por Maps | | Secure Boot para Windows 8 | |
| 2013 | Docker se muestra en PyCon | Blink se bifurca de WebKit | | |
| 2014 | Se anuncian Swift y Metal | BoringSSL, fork de OpenSSL | | |
| 2015 | Kubernetes 1.0<br>Windows Phone: pérdida contable<br>Fin de teléfonos Firefox OS | Rust 1.0 | SIP en OS X El Capitan | |
| 2016 | Fedora 25: Wayland de serie<br>Google revela la TPU<br>BlackBerry deja sus teléfonos | | | |
| 2017 | APFS por defecto; AWS Nitro<br>Linux en todo el TOP500<br>Canonical abandona Unity8 | | | Rico (datos de UI), UIST |
| 2018 | Firecracker; gVisor<br>AWS Graviton | Edge pasa a Chromium | Spectre y Meltdown | |
| 2019 | DriverKit; io_uring en 5.1<br>Huawei presenta HarmonyOS | WebAssembly, estándar W3C | Huawei en la Entity List<br>MSRC: 70 % son de memoria | |
| 2020 | Apple M1, memoria unificada<br>Se anuncia Pluton | Nuevo Edge; WebView2 GA | BootHole en GRUB2 | |
| 2021 | | Fin del soporte Edge Legacy | EO 14028; elementos SBOM<br>Vence raíz de Let's Encrypt | UiPath sale a bolsa (RPA) |
| 2022 | Soporte de Rust en Linux 6.1<br>BlackBerry 10 sin servicios | Pingora descrito; workerd | Guía NSA de memoria segura | Copilot: 55 % más rápido |
| 2023 | Rust en win32k; EEVDF<br>Azure Maia y Cobalt | | libwebp CVE-2023-4863<br>LogoFAIL | |
| 2024 | Plan Windows Resiliency | iOS en la UE: otros motores<br>Pingora, código abierto | Puerta trasera de xz<br>Caída del kernel, 19 de julio<br>El kernel Linux, nueva CNA<br>Cyber Resilience Act (UE) | Llamada deepfake a Arup<br>DARPA TRACTOR<br>Mandiant: explotar en 5 días<br>Computer use en beta<br>Big Sleep: falla en SQLite |
| 2025 | Fin del experimento Rust<br>GNOME 49: sin sesión X11<br>Fin de Windows 10 Home, Pro | | CVE de FreeType en KEV<br>CISA, NSA: memoria segura | Operator, vista previa<br>Final de AIxCC en DEF CON<br>Informe campaña GTG-1002 |
| 2026 | | | Vencen certificados de 2011 | OWASP LLM Top 10 2026 |

"Caída del kernel, 19 de julio" es la actualización defectuosa de un agente de seguridad que se cuenta en el capítulo 2. La entrada de Mandiant es su análisis de octubre de 2024 sobre datos de 2023 (capítulos 11 y 13). Los anillos rojos después de 2010 son los dueños que abandonaron, de los capítulos 10 y 12. Los certificados de Secure Boot de 2011 vencieron en junio y octubre de 2026 (capítulo 1).

LEYENDA

## Cómo leer el póster

- Ganador, plataforma o sistema entregado
- Sistema fallido, o un paso de su declive
- Sistema de investigación, artículo o estudio
- Evento de seguridad, falla o guía
- Bandas sombreadas: décadas alternas. Punto verde azulado con anillo rojo: investigación que perdió su mercado.

Los colores indican el tipo de evento; los carriles indican la capa. Una fila leída de lado a lado muestra lo que pasó el mismo año en distintas capas.

### El patrón en un párrafo

Al leer el carril de sistemas operativos de arriba abajo aparece el péndulo del capítulo 9. Un solo proveedor fue dueño de toda la máquina hasta 1969. De los años setenta a los noventa, la pila se dividió en capas, cada una un mercado. Desde 2007, con el iPhone, y luego con los chips propios de los proveedores de nube, volvió la integración. Al leer los dos carriles de la derecha aparece el tema de la Parte III: los eventos de seguridad y de IA se amontonan en la última década, y son la razón por la que el tiempo que tarda una empresa en entregar una corrección ahora importa tanto como quién es dueño de cada capa.

APÉNDICE C

# Tabla comparativa completa

Este apéndice reúne en una sola referencia las comparaciones de los tres sistemas de cada capítulo, ordenadas desde el silicio hasta el píxel y luego hacia las capas de negocio. Cada fila dice lo mismo que su capítulo; el capítulo cuenta la historia y da las fuentes. Cuando un dato depende del momento, está vigente a noviembre de 2026.

## La pulsación de tecla, de principio a fin (Prólogo)

| Paso | Windows | Linux (escritorio Wayland) | macOS |
|---|---|---|---|
| **Informe de la tecla** | El teclado envía un informe USB HID que lleva el estado de los modificadores y el código de uso 0x04, la posición de la tecla "a y A"; el sistema operativo aplica la distribución del teclado para decidir el carácter | | |
| **Entrada en el kernel** | kbdhid.sys, kbdclass.sys | usbhid, evdev | IOHIDFamily |
| **Evento hacia la aplicación** | win32k, WM_KEYDOWN y WM_CHAR | libinput, xkbcommon, evento de tecla de Wayland | WindowServer, NSEvent |
| **Texto** | DirectWrite | HarfBuzz da forma al texto, FreeType dibuja | Core Text da forma al texto y lo compone; Core Graphics rasteriza |
| **Componer la pantalla** | Desktop Window Manager | Mutter o KWin | WindowServer, Core Animation |
| **GPU y pantalla** | Direct3D, controlador WDDM | Mesa, DRM y KMS | Metal |
| **Luz** | El nuevo fotograma se muestra en la siguiente actualización de la pantalla | | |

## Capa 1: Silicio y arranque (Capítulo 1)

| Pregunta | PC con Windows | PC con Linux | Mac con Apple silicon |
|---|---|---|---|
| **Dónde se ejecuta el kernel** | Anillo 0 en x86; aplicaciones en el anillo 3 | Anillo 0 en x86; aplicaciones en el anillo 3 | EL1 en Arm; aplicaciones en EL0 |
| **Memoria de CPU y GPU** | En una computadora con tarjeta gráfica dedicada, los datos suelen copiarse a su memoria de video; las GPU integradas comparten la memoria principal | Lo mismo que Windows en el mismo hardware | Memoria unificada, un solo grupo para CPU y GPU, desde el M1 (2020) |
| **Primera etapa de firmware del host para arrancar el sistema operativo** | Firmware UEFI del fabricante de la computadora, normalmente sobre un núcleo de AMI, Insyde o Phoenix | El mismo firmware UEFI | Boot ROM, fijada en el chip al fabricarlo |
| **Quién firma la siguiente etapa** | Microsoft (administrador de arranque de Windows) | Microsoft firma `shim`; la distribución firma GRUB y el kernel | Apple: la Boot ROM comprueba el Low-Level Bootloader (LLB), que comprueba iBoot |
| **Almacén de claves y políticas** | UEFI `db`, `dbx`, `KEK`, `PK` | Claves UEFI, más la lista Machine Owner Key de `shim` | LocalPolicy, firmada por el Secure Enclave |
| **Custodio de claves en hardware** | TPM 2.0 o Pluton | TPM 2.0 | Secure Enclave, con su propia ROM de arranque y sepOS |
| **Cómo llega el microcódigo** | Sobre todo dentro de las actualizaciones de firmware del fabricante de la computadora; Microsoft también publica paquetes de microcódigo de Intel desde marzo de 2018 | El kernel lo carga temprano desde el disco RAM inicial, a partir de paquetes de la distribución; la carga tardía está desactivada por defecto desde 5.19 | Dentro de las actualizaciones de macOS |
| **Quién entrega las correcciones de firmware** | El fabricante de la computadora; algunas mediante Windows Update | El fabricante de la computadora; algunas mediante el Linux Vendor Firmware Service | Apple, dentro de las actualizaciones de macOS |

## Capa 2: El kernel (Capítulo 2)

| Pregunta | Windows (kernel NT) | Linux | macOS (XNU) |
|---|---|---|---|
| **Diseño** | Híbrido: kernel más ejecutivo, controladores en modo kernel | Monolítico con módulos cargables | Híbrido: Mach más BSD más IOKit |
| **Quién controla el código fuente** | Microsoft (cerrado) | Los mantenedores del proyecto original, liderados por Linus Torvalds (GPLv2) | Apple (código publicado para partes de Darwin) |
| **Modelo del planificador** | Apropiativo, 32 niveles de prioridad (0 a 31), turno rotatorio dentro de cada nivel | EEVDF desde Linux 6.6 (octubre de 2023), en sustitución de CFS | Hilos de Mach, guiados por las clases de calidad de servicio de las aplicaciones |
| **Código de terceros en el kernel** | Común: controladores, agentes de seguridad, anti-cheat | Común: módulos de fabricantes de hardware y de proveedores de seguridad | Restringido: extensiones del kernel declaradas obsoletas (deprecated) desde 2019 |
| **Hacia dónde se mueve el código externo** | Modo usuario, dentro de la Windows Resiliency Initiative | No hay una política única; el proyecto original prefiere controladores integrados en el árbol | System Extensions y DriverKit, en el espacio de usuario |
| **Rust en el kernel** | Una implementación en Rust de las regiones GDI (`win32kbase_rs.sys`) desde la compilación Insider 25905, julio de 2023 | Soportado desde 6.1 (diciembre de 2022); el experimento se declaró concluido en diciembre de 2025 | No anunciado para el kernel XNU |
| **Controladores en Rust en productos comerciales** | Surface dijo en julio de 2025 que había entregado varios equipos Copilot+ PC con controladores escritos en Rust | Linux 6.18 (noviembre de 2025) añadió el controlador Rust Binder que usa Android | No anunciado |
| **Cómo se construyen controladores en Rust** | `windows-drivers-rs`, un proyecto de código abierto de Microsoft | Enlaces de Rust mantenidos dentro del árbol del kernel | No aplica |
| **Quién decide el ritmo** | Microsoft | Los mantenedores del kernel, subsistema por subsistema | Apple |

## Capa 3: Controladores, almacenamiento y red (Capítulo 3)

| Pregunta | Windows | Linux | macOS |
|---|---|---|---|
| **Interfaz de controladores** | WDM, con KMDF y UMDF encima; estable entre versiones | Sin interfaz estable dentro del kernel; los controladores pertenecen al árbol del kernel | DriverKit en el espacio de usuario; extensiones del kernel declaradas obsoletas |
| **Quién escribe la mayoría de los controladores** | Los fabricantes de hardware | Fabricantes y desarrolladores del kernel, en el proyecto original | Apple, más los fabricantes de dispositivos adicionales |
| **Quién permite que un controlador se ejecute** | Microsoft firma los controladores del kernel (Windows 10, 1607 y posteriores) | La distribución que compila el kernel, y el dueño de la máquina | Apple, mediante la firma de código y la aprobación de extensiones del sistema |
| **Ruta de corrección de un error en un controlador** | El fabricante la entrega; Windows Update puede llevarla | Parche en el proyecto original, luego el kernel de cada distribución | La actualización del sistema de Apple, o la extensión del fabricante |
| **Dueño del controlador de la GPU** | El fabricante de la GPU, en dos mitades bajo WDDM | Controlador DRM del kernel más Mesa en el espacio de usuario; los módulos abiertos de NVIDIA se entregan fuera del árbol | Apple |
| **Respaldo por software cuando falla la GPU** | WARP | llvmpipe de Mesa | Documentado solo para el framework OpenGL declarado obsoleto, no para Metal |

| Función | Windows | Linux | macOS |
|---|---|---|---|
| **Sistema de archivos predeterminado** | NTFS | ext4 o XFS, según la distribución; Btrfs en algunas | APFS |
| **Sumas de verificación contra la corrupción silenciosa** | ReFS (no arrancable, a noviembre de 2026) | Btrfs, para datos y metadatos | No se trata en este libro |
| **Instantáneas** | Volume Shadow Copy; instantáneas de archivos de ReFS en Server 2022 y posteriores | Btrfs, o el gestor de volúmenes | Instantáneas de APFS |
| **Cifrado en reposo** | BitLocker, debajo del sistema de archivos | dm-crypt, debajo del sistema de archivos | Integrado en APFS; FileVault liga la clave a la contraseña del usuario |
| **Interfaz de red para los programas** | Winsock, basado en los sockets de Berkeley | Sockets BSD; io_uring desde Linux 5.1 (2019) | Sockets BSD; Network.framework para desarrolladores de aplicaciones desde 2018 |
| **Programas dentro del kernel** | eBPF for Windows, en desarrollo a noviembre de 2026 | eBPF, revisado por un verificador | No se trata en este libro |

## Capa 4: Seguridad, identidad y el píxel de confianza (Capítulos 4 y 13)

| Frontera | Windows | Linux | macOS |
|---|---|---|---|
| **Identidad de un proceso** | Token de acceso | Identificadores de usuario y de grupo, más capacidades | Identificador de usuario, más firma de código y entitlements |
| **Quién puede acceder a un objeto** | ACL en cada objeto | Bits de permisos; política de SELinux o AppArmor | Bits de permisos; perfil de sandbox |
| **Reglas que ni el administrador puede saltarse** | Procesos protegidos, firma de controladores | SELinux o AppArmor en modo de aplicación (enforcing) | System Integrity Protection |
| **Preguntar a la persona** | UAC en el escritorio seguro | Avisos propios de cada escritorio (polkit, portales) | Avisos de TCC |
| **Gestor de servicios** | Service Control Manager | systemd en la mayoría de las distribuciones | launchd |
| **Framework de inicio de sesión** | Windows Hello y proveedores de credenciales | PAM | Touch ID verificado en el Secure Enclave, mediante Authorization Services y LocalAuthentication; PAM para `sudo` y algunos servicios |
| **Interfaz de accesibilidad que un agente puede leer** | UI Automation | ATK y AT-SPI | AX API (NSAccessibility) |
| **Quién define la semántica de los controles** | Los frameworks de interfaz de Microsoft, luego cada equipo de aplicación | Proyectos de toolkits como GTK y Qt, luego cada equipo de aplicación | AppKit y SwiftUI de Apple, luego cada equipo de aplicación |
| **Cómo pide una aplicación capturar la pantalla** | Windows.Graphics.Capture: el usuario elige en la interfaz del sistema, y un borde amarillo marca lo que se captura | En escritorios Wayland, el portal ScreenCast de xdg-desktop-portal | El permiso de grabación de pantalla de TCC |

## Capa 5: Entornos de ejecución y lenguajes (Capítulo 5)

| Pregunta | Windows | Linux | macOS |
|---|---|---|---|
| **Runtime base de C** | Universal CRT, parte de Windows 10 y posteriores | glibc (musl en algunas distribuciones pequeñas) | libSystem |
| **Framework nativo de aplicaciones** | Win32, con .NET encima | Ninguno desde el kernel; GTK o Qt elegidos por cada aplicación | Cocoa, con Swift y Objective-C |
| **Runtime administrado del proveedor** | .NET CLR | Ninguno por defecto | Ninguno con recolector de basura; Swift usa conteo de referencias |
| **Quién parchea el runtime base** | Microsoft, mediante Windows Update | El proveedor de la distribución | Apple, mediante las actualizaciones del sistema |
| **¿Puede una aplicación sustituirlo?** | No la UCRT en Windows 10 y 11 | Sí, con paquetes autocontenidos o contenedores | Rara vez, y solo para destinos antiguos de Swift |

## Capa 6: Gráficos, texto y el píxel (Capítulo 6)

| Parte | Windows | Linux | macOS |
|---|---|---|---|
| **Interfaz gráfica** | Direct3D 11 y 12 | OpenGL, Vulkan (Mesa) | Metal (OpenGL declarado obsoleto desde Mojave, 2018) |
| **Compositor** | DWM, siempre activo desde Windows 8 | Mutter, KWin y otros, bajo Wayland | WindowServer con Core Animation |
| **Modelo de controladores** | WDDM, desde Vista | DRM y KMS en el kernel | Controladores propios de Apple |
| **Motor de texto** | DirectWrite | HarfBuzz da forma al texto, FreeType rasteriza | Core Text da forma al texto y lo compone; Core Graphics (Quartz) rasteriza |
| **Quién entrega las correcciones** | Microsoft y el fabricante de la GPU | Kernel, Mesa, proyecto de escritorio, distribución | Apple |

## Capa 7: El motor de navegador (Capítulos 7 y 11)

| Pregunta | Windows | Linux | macOS |
|---|---|---|---|
| **Navegador y motor del proveedor** | Edge: Chromium (Blink, V8) | No hay un navegador único del proveedor; la distribución elige | Safari: WebKit |
| **Vista web que una aplicación puede tomar prestada** | WebView2 (Chromium) | WebKitGTK (WebKit) o Qt WebEngine (Chromium) | WKWebView (WebKit del sistema) |
| **Quién parchea el motor prestado** | Microsoft, mediante el runtime Evergreen; o la aplicación, con Fixed Version | Los mantenedores de paquetes de la distribución | Apple, mediante las actualizaciones del sistema y de Safari |
| **Motor empaquetado en la aplicación** | Electron y frameworks similares: una copia de Chromium por aplicación, parcheada por cada proveedor de la aplicación | | |
| **De dónde vienen las copias adicionales del motor** | De cada aplicación Electron y de cada aplicación con WebView2 Fixed Version | De cada aplicación Electron y de los navegadores que empaquetan sus propias bibliotecas | De cada aplicación Electron y de cada navegador de terceros |
| **Quién corrige una falla en una biblioteca compartida** | Microsoft, para las bibliotecas del sistema y el runtime Evergreen de WebView2; cada producto, para un runtime Fixed Version u otra copia privada | La distribución, una vez por cada paquete fuente que lleve una copia | Apple para las copias del sistema; cada proveedor para las suyas |
| **Quién fija la hoja de ruta del motor** | Google, con Microsoft como gran contribuidor | El proyecto original del motor que sea | Apple |

## Capa 8: Cadena de herramientas y distribución (Capítulos 8 y 11)

| Pregunta | Windows | Linux | macOS |
|---|---|---|---|
| **Compilador nativo principal** | MSVC (Clang disponible) | GCC y LLVM/Clang | Apple Clang en Xcode |
| **Formatos de paquete de aplicaciones** | MSIX, firmado; instaladores MSI y EXE más antiguos | deb, rpm; Flatpak y Snap entre distribuciones | Paquetes de aplicación (bundles) y paquetes de instalación |
| **Canales de actualización que la plataforma ofrece a las aplicaciones** | Microsoft Store, WinGet o el actualizador propio de la aplicación | Los repositorios de la distribución, Flatpak, Snap | Mac App Store o el actualizador propio de la aplicación |
| **Quién está en la ruta de publicación** | La Microsoft Store, si se usa; la firma para MSIX | Los mantenedores de la distribución, o la tienda de Flatpak o Snap | Apple: la App Store, o la notarización fuera de ella |
| **Quién construye el sistema operativo a partir de piezas** | Microsoft | La distribución, a partir del kernel de la comunidad del kernel y de miles de proyectos | Apple |

## Historia y ecosistema (Capítulos 9 y 10)

| Pregunta | Windows | Linux | macOS |
|---|---|---|---|
| **En qué oscilación nació** | Horizontal: un sistema operativo licenciado a muchos fabricantes de computadoras | Horizontal: un kernel que no pertenece a ninguna empresa | Integrada: una sola empresa, Mach y BSD bajo su propio diseño |
| **Quién fabrica el hardware** | Muchos fabricantes de computadoras | Cualquiera; los proveedores de nube, con sus propios chips | Solo Apple |
| **Quién decide la capa de abajo** | Intel, AMD, Qualcomm y el fabricante de la computadora | El fabricante del chip y la distribución | Apple, desde su propio silicio |
| **Dirección actual** | Atraído hacia la integración: el procesador de seguridad Pluton está dentro de la CPU y se actualiza mediante Windows Update | Ambas: proyecto original abierto, integrado dentro de cada nube | Totalmente integrado, del chip a la App Store |
| **Quién lo instala en la máquina** | Los fabricantes de computadoras, en la fábrica | Algunos fabricantes de computadoras; sobre todo lo instala TI, y es el predeterminado en los servidores de nube | Apple, en cada Mac |
| **De dónde vienen las aplicaciones** | Descargas directas, instaladores y la Microsoft Store | Repositorios de la distribución, Flatpak, Snap y contenedores | La Mac App Store y descargas directas |
| **Promesa de compatibilidad** | Fuerte: décadas de aplicaciones Win32, 32 bits mediante WOW64 | Fuerte para la interfaz del kernel hacia el espacio de usuario; más débil para las bibliotecas de escritorio | Apple avanza cuando cambia de chips o de API |
| **Qué alquila a otros** | El hardware y la mayoría de las aplicaciones | Casi todo: cualquiera puede distribuirlo | Solo la capa de aplicaciones |

## De qué puede ser dueña una empresa (Capítulo 15)

| Capa | Windows | Linux | macOS |
|---|---|---|---|
| **Kernel** | Solo alquilar: Microsoft lo actualiza | Tomarlo de una distribución (alquilar), ajustarlo o bifurcarlo | Solo alquilar: Apple lo actualiza |
| **Cadena de arranque** | Alquilar: firmware del fabricante de la computadora, certificados de Microsoft | Tomar shim y GRUB de una distribución (alquilar); Microsoft firma shim | Solo alquilar: Boot ROM, LLB e iBoot de Apple |
| **Compositor** | Alquilar: DWM | Elegir: Mutter, KWin u otro; una flota controla la elección | Alquilar: WindowServer |
| **Vista web del sistema** | Alquilar WebView2, o empaquetar Chromium y controlar su ruta de parches | Tomarla de la distribución (alquilar), o empaquetarla y controlar la ruta de parches | En macOS de escritorio: alquilar WKWebView, o empaquetarla y controlar la ruta de parches |
| **De qué se puede ser dueño, o qué se puede controlar, en los tres** | Ser dueño: la aplicación propia y las pantallas de aprobación propias. Controlar: la ruta de compilación y publicación, las claves de firma y la política propias. | | |

**CÓMO USAR ESTA TABLA**

Una fila leída de lado a lado muestra quién es dueño de la misma tarea en cada sistema. Una columna leída de arriba abajo muestra cuántos dueños pone un sistema entre el código propio y la pantalla. En macOS, la mayoría de las filas terminan en Apple. En Windows, la mayoría terminan en Microsoft o en un fabricante de hardware. En Linux, la mayoría terminan en una distribución o en un proyecto original, y por eso "¿qué Linux?" es siempre la primera pregunta. Cada fila que termina en otra parte es una fila para la lista de materiales del píxel (Pixel Bill of Materials).

APÉNDICE D

# Lecturas y fuentes

La parte 1 es un estante corto con los artículos y libros que dieron forma a los sistemas de este libro. La parte 2 enumera las principales fuentes primarias detrás de cada capítulo, para que el lector pueda comprobar cualquier afirmación en su origen. Cada fuente de la parte 2 se comprobó el 17 de noviembre de 2026; las páginas web cambian, así que conviene comprobar la fecha de todo aquello en lo que se confíe.

## Parte 1: Lecturas fundamentales

| Obra | Año | Por qué leerla |
|---|---|---|
| F. J. Corbató and V. A. Vyssotsky, "Introduction and Overview of the Multics System", Fall Joint Computer Conference | 1965 | Los objetivos de diseño del "computer utility" (la informática como servicio público), muchos de los cuales todo sistema operativo moderno sigue persiguiendo: protección, uso compartido y operación continua. |
| E. W. Dijkstra, "The Structure of the 'THE'-Multiprogramming System", Communications of the ACM 11(5) | 1968 | El argumento clásico para construir un sistema operativo en capas, cada una apoyada solo en las de abajo. Es breve, y es la raíz de la visión capa por capa de este libro. |
| D. M. Ritchie and K. Thompson, "The UNIX Time-Sharing System", Communications of the ACM 17(7) | 1974 | El artículo que defendió con números un sistema operativo pequeño y portable. Su modelo de procesos y archivos sigue en la base de Linux y macOS (capítulos 2 y 9). |
| K. Thompson, "Reflections on Trusting Trust", Communications of the ACM 27(8) | 1984 | Tres páginas sobre por qué leer el código fuente no prueba lo que hace un binario. El punto de partida de los capítulos 8 y 14. |
| M. Accetta, R. Baron, W. Bolosky, D. Golub, R. Rashid, A. Tevanian and M. Young, "Mach: A New Kernel Foundation for UNIX Development", Summer USENIX Conference | 1986 | El microkernel que pasó a formar parte de XNU, el kernel de macOS. Explica los compromisos detrás de la palabra "híbrido" del capítulo 2. |
| F. P. Brooks Jr., The Mythical Man-Month: Essays on Software Engineering (Addison-Wesley) | 1975 | Lecciones de dirigir OS/360. Sigue siendo la mejor explicación de por qué ser dueño de una capa grande cuesta más personas y más tiempo de lo previsto (capítulos 9 y 12). |
| R. H. Arpaci-Dusseau and A. C. Arpaci-Dusseau, Operating Systems: Three Easy Pieces, gratis en línea en `pages.cs.wisc.edu/~remzi/OSTEP/` | En línea, actualizado | El libro de texto moderno más claro sobre procesos, memoria, concurrencia y almacenamiento. Conviene leerlo cuando un capítulo de este libro deja con ganas de conocer el mecanismo. |
| A. Agache et al., "Firecracker: Lightweight Virtualization for Serverless Applications", USENIX NSDI | 2020 | Un relato reciente y legible de una empresa que es dueña de una capa estrecha, un monitor de microVM, y alquila el kernel que hay debajo (capítulos 8 y 14). |

## Parte 2: Fuentes principales por capítulo

Cada entrada indica la fuente, lo que respalda y su dirección. La lista completa de afirmaciones y fuentes de cada capítulo se conserva con el manuscrito; estas son las fuentes primarias más importantes.

### Resumen para el lector

- **Microsoft, "Helping our customers through the CrowdStrike outage"** (20 de julio de 2024): 8,5 millones de dispositivos afectados. `blogs.microsoft.com/blog/2024/07/20/helping-our-customers-through-the-crowdstrike-outage/`

### Capítulo 1, Silicio y arranque

- **Microsoft Learn, "User mode and kernel mode"**: un solo espacio de direcciones en modo kernel. `learn.microsoft.com/en-us/windows-hardware/drivers/gettingstarted/user-mode-and-kernel-mode`
- **Apple Newsroom, M1 announcement** (10 de noviembre de 2020): memoria unificada. `apple.com/newsroom/2020/11/apple-unleashes-m1/`
- **Microsoft Learn, Secure Boot for OEMs**: db, dbx, KEK y PK. `learn.microsoft.com/en-us/windows-hardware/design/device-experiences/oem-secure-boot`
- **Microsoft Security blog, Pluton** (17 de noviembre de 2020). `microsoft.com/en-us/security/blog/2020/11/17/meet-the-microsoft-pluton-processor-the-security-chip-designed-for-the-future-of-windows-pcs/`
- **Apple Platform Security, boot process for a Mac with Apple silicon**: Boot ROM, LLB, iBoot. `support.apple.com/guide/security/boot-process-for-a-mac-with-apple-silicon-secac71d5623/web`
- **Linux kernel documentation, microcode loading**. `kernel.org/doc/html/latest/arch/x86/microcode.html`
- **Google Project Zero, Meltdown and Spectre** (3 de enero de 2018). `projectzero.google/2018/01/reading-privileged-memory-with-side.html`
- **CERT/CC note on LogoFAIL** (diciembre de 2023). `kb.cert.org/vuls/id/811862`
- **Microsoft Support KB5062710**: vencimiento de los certificados de Secure Boot y actualizaciones de las CA. `support.microsoft.com/kb/5062710`

### Capítulo 2, El kernel

- **CrowdStrike, technical details of the Falcon update**: Channel File 291, de 04:09 a 05:27 UTC. `crowdstrike.com/en-us/blog/falcon-update-for-windows-hosts-technical-details/`
- **Apple XNU source repository**: kernel híbrido, Mach, FreeBSD e IOKit. `github.com/apple-oss-distributions/xnu`
- **Microsoft Learn, scheduling priorities**. `learn.microsoft.com/en-us/windows/win32/procthread/scheduling-priorities`
- **Kernel Newbies, Linux 6.6**: EEVDF. `kernelnewbies.org/Linux_6.6`
- **Windows Experience blog, Windows Resiliency Initiative** (19 de noviembre de 2024). `blogs.windows.com/windowsexperience/2024/11/19/windows-security-and-resiliency-protecting-your-business/`

- **Windows Experience blog, Windows security at Ignite 2026** (17 de noviembre de 2026): la plataforma de seguridad de los equipos (endpoints) de Windows alcanza la disponibilidad general en Windows 11. `blogs.windows.com/windowsexperience/2026/11/17/windows-security-at-ignite-2026-start-secure-protect-what-matters-prepare-for-ai/`
- **Apple Developer, kernel extensions**: obsolescencia y sustitutos. `developer.apple.com/support/kernel-extensions/`
- **Windows Insider build 25905** (12 de julio de 2023): Rust en win32k. `blogs.windows.com/windows-insider/2023/07/12/announcing-windows-11-insider-preview-build-25905/`
- **Check Point Research, Rust in the Windows kernel** (2025). `research.checkpoint.com/2025/denial-of-fuzzing-rust-in-the-windows-kernel/`
- **LWN, the Rust experiment concluded** (diciembre de 2025). `lwn.net/Articles/1050174/`

### Capítulo 3, Controladores, almacenamiento y red

- **Linux kernel documentation, "stable-api-nonsense"**. `kernel.org/doc/html/latest/process/stable-api-nonsense.html`
- **Microsoft, driver signing changes in Windows 10, version 1607**. `techcommunity.microsoft.com/blog/windowshardwarecertification/driver-signing-changes-in-windows-10-version-1607/364894`
- **Microsoft Learn, WDDM design guide**. `learn.microsoft.com/en-us/windows-hardware/drivers/display/windows-vista-display-driver-model-design-guide`
- **NVIDIA, open-source GPU kernel modules** (mayo de 2022). `forums.developer.nvidia.com/t/nvidia-releases-open-source-gpu-kernel-modules/214128`
- **Microsoft Learn, ReFS overview**. `learn.microsoft.com/en-us/windows-server/storage/refs/refs-overview`
- **Apple Platform Security, APFS encryption**. `support.apple.com/guide/security/seca6147599e/web`
- **Apple WWDC 2018 session 715, Network.framework**. `developer.apple.com/videos/play/wwdc2018/715/`
- **eBPF for Windows repository**. `github.com/microsoft/ebpf-for-windows`
- **Cloudflare, "How we built Pingora"** (14 de septiembre de 2022). `blog.cloudflare.com/how-we-built-pingora-the-proxy-that-connects-cloudflare-to-the-internet/`

### Capítulo 4, Seguridad, identidad y el píxel de confianza

- **Microsoft Learn, how User Account Control works**: el escritorio seguro. `learn.microsoft.com/en-us/windows/security/application-security/application-control/user-account-control/how-it-works`
- **Linux man page, capabilities(7)**. `man7.org/linux/man-pages/man7/capabilities.7.html`
- **Lennart Poettering, "Rethinking PID 1"** (abril de 2010). `0pointer.net/blog/projects/systemd.html`

- **Commission Delegated Regulation (EU) 2018/389, Article 5**: vinculación dinámica. `legislation.gov.uk/eur/2018/389/article/5`
- **European Commission, eIDAS regulation**: carteras de identidad digital. `digital-strategy.ec.europa.eu/en/policies/eudi-regulation`
- **Android Developers, tapjacking**. `developer.android.com/privacy-and-security/risks/tapjacking`
- **Microsoft Learn, SetWindowDisplayAffinity**. `learn.microsoft.com/en-us/windows/win32/api/winuser/nf-winuser-setwindowdisplayaffinity`
- **Windows Experience blog, Recall security and privacy architecture** (27 de septiembre de 2024). `blogs.windows.com/windowsexperience/2024/09/27/update-on-recall-security-and-privacy-architecture/`
- **15 CFR 734.13 and 734.15**: exportación presunta (deemed export) y entrega. `law.cornell.edu/cfr/text/15/734.15`


## Capítulo 5, Entornos de ejecución y lenguajes


- **Microsoft Learn, Universal CRT deployment**. `learn.microsoft.com/en-us/cpp/windows/universal-crt-deployment`
- **Swift.org, ABI stability and Apple platforms**. `swift.org/blog/abi-stability-and-apple/`
- **Node.js, previous releases and LTS policy**. `nodejs.org/en/about/previous-releases`
- **.NET support policy**. `dotnet.microsoft.com/en-us/platform/support/policy/dotnet-core`
- **MSRC, "A proactive approach to more secure code"** (julio de 2019). `microsoft.com/en-us/msrc/blog/2019/07/a-proactive-approach-to-more-secure-code`
- **Google Security blog, memory safety in Android** (septiembre de 2024). `security.googleblog.com/2024/09/eliminating-memory-safety-vulnerabilities-Android.html`
- **CISA and partners, "The Case for Memory Safe Roadmaps"** (6 de diciembre de 2023). `cisa.gov/resources-tools/resources/case-memory-safe-roadmaps`
- **CISA and NSA, memory safe languages guidance** (24 de junio de 2025). `cisa.gov/news-events/alerts/2025/06/24/new-guidance-released-reducing-memory-related-vulnerabilities`
- **DARPA, TRACTOR announcement** (31 de julio de 2024). `darpa.mil/news/2024/memory-safety-vulnerabilities`


## Capítulo 6, Gráficos, texto y el píxel


- **Microsoft Learn, Desktop Window Manager overview**. `learn.microsoft.com/en-us/windows/win32/dwm/dwm-overview`
- **Wayland 1.0 announcement** (22 de octubre de 2012). `lists.freedesktop.org/archives/wayland-devel/2012-October/005967.html`
- **GNOME 49 release notes for developers**. `release.gnome.org/49/developers/`
- **NVD record for CVE-2025-27363** (FreeType). `nvd.nist.gov/vuln/detail/CVE-2025-27363`

- **Chrome Releases, stable channel update of 11 September 2023**. chromereleases.googleblog.com/2023/09/stable-channel-update-for-desktop_11.html
- **Apple, macOS Mojave 10.14 release notes**: OpenGL declarado obsoleto. developer.apple.com/documentation/macos-release-notes/macos-mojave-10_14-release-notes
- **ITU-R Recommendation BT.2100-0** (julio de 2016). itu.int/rec/R-REC-BT.2100
- **Google Workspace Updates, canvas-based rendering in Docs** (mayo de 2021). workspaceupdates.googleblog.com/2021/05/Google-Docs-Canvas-Based-Rendering-Update.html


### Capítulo 7, El navegador, el segundo sistema operativo


- **Chromium blog, Blink announcement** (3 de abril de 2013). blog.chromium.org/2013/04/blink-rendering-engine-for-chromium.html
- **Apple Newsroom, Safari unveiled** (7 de enero de 2003). apple.com/newsroom/2003/01/07Apple-Unveils-Safari/
- **Windows Experience blog, Edge adopts Chromium** (6 de diciembre de 2018). blogs.windows.com/windowsexperience/2018/12/06/microsoft-edge-making-the-web-better-through-more-open-source-collaboration/
- **Microsoft Learn, WebView2 distribution**: Evergreen y Fixed Version. learn.microsoft.com/en-us/microsoft-edge/webview2/concepts/distribution
- **Electron documentation and release timelines**. electronjs.org/docs/latest/tutorial/electron-timelines
- **GitHub advisory GHSA-j7hp-h8jx-5ppr** (libwebp). github.com/advisories/GHSA-j7hp-h8jx-5ppr
- **Chromium, site isolation design**. chromium.org/developers/design-documents/site-isolation/
- **Apple App Review Guidelines**, directriz 2.5.6. developer.apple.com/app-store/review/guidelines/
- **Apple Developer, alternative browser engines**. developer.apple.com/support/alternative-browser-engines/


### Capítulo 8, Cadena de herramientas, distribución y los bordes de un sistema operativo


- **GCC releases**. gcc.gnu.org/releases.html
- **Microsoft Learn, MSIX overview**. learn.microsoft.com/en-us/windows/msix/overview
- **Apple Developer news, notarization for macOS Catalina** (3 de septiembre de 2019). developer.apple.com/news/?id=09032019a
- **Andres Freund, xz backdoor report** (29 de marzo de 2024). openwall.com/lists/oss-security/2024/03/29/4
- **Tukaani project, xz backdoor notice**. tukaani.org/xz-backdoor/
- **AWS News blog, Firecracker** (26 de noviembre de 2018). aws.amazon.com/blogs/aws/firecracker-lightweight-virtualization-for-serverless-computing/
- **gVisor documentation**. gvisor.dev/docs/

- **Reproducible Builds, definition**. reproducible-builds.org/docs/definition/
- **NTIA, minimum elements for an SBOM** (12 de julio de 2021). ntia.gov/report/2021/minimum-elements-software-bill-materials-sbom


## Capítulo 9, Setenta años de sistemas operativos


- **Computer History Museum, GM-NAA I/O** (Robert L. Patrick). softwarepreservation.computerhistory.org/os/gm.html
- **Multicians, Multics history**. multicians.org/history.html
- **IBM, System/360 history**. ibm.com/history/system-360
- **Computer History Museum, IBM unbundling** (1969). computerhistory.org/revolution/mainframe-computers/7/172
- **Computer History Museum, fifty years of the PC operating system**. computerhistory.org/blog/fifty-years-of-the-personal-computer-operating-system/
- **AWS, the Nitro System journey**. docs.aws.amazon.com/whitepapers/latest/security-design-of-aws-nitro-system/the-nitro-system-journey.html
- **Microsoft, Azure Maia and Cobalt** (15 de noviembre de 2023). news.microsoft.com/source/features/ai/in-house-chips-silicon-to-service-to-meet-ai-demand/
- **Open Handset Alliance press release** (5 de noviembre de 2007). openhandsetalliance.com/press_110507.html


## Capítulo 10, Por qué ganaron los ganadores


- **Microsoft, history of Windows**. learn.microsoft.com/en-us/archive/blogs/uktechnet/history-of-windows
- **Plan 9 from Bell Labs, about**. 9p.io/plan9/about.html
- **Rob Pike, history of UTF-8**. cl.cam.ac.uk/~mgk25/ucs/utf-8-history.txt
- **Microsoft, restructuring of the phone hardware business** (8 de julio de 2015). news.microsoft.com/2015/07/08/microsoft-announces-restructuring-of-phone-hardware-business/
- **Microsoft Learn, running 32-bit applications (WOW64)**. learn.microsoft.com/en-us/windows/win32/winprog64/running-32-bit-applications
- **Linux 0.12 release notes**: el paso a la GPL. cdn.kernel.org/pub/linux/kernel/Historic/old-versions/RELNOTES-0.12
- **European Commission, Android decision** (18 de julio de 2018). ec.europa.eu/commission/presscorner/detail/en/IP_18_4581
- **Apple Newsroom, iPhone SDK** (6 de marzo de 2008). apple.com/newsroom/2008/03/06Apple-Announces-iPhone-2-0-Software-Beta/

## Capítulo 11, Las costuras, en números

- **Electron release v26.2.1**: tamaños de los archivos y versión de Chromium. `github.com/electron/electron/releases/tag/v26.2.1`
- **libwebp fix commit and NEWS file**. `github.com/webmproject/libwebp/blob/main/NEWS`
- **Mozilla Foundation Security Advisory 2023-40**. `mozilla.org/en-US/security/advisories/mfsa2023-40/`
- **CISA Known Exploited Vulnerabilities catalog**. `cisa.gov/known-exploited-vulnerabilities-catalog`
- **Debian security tracker, CVE-2023-4863**. `security-tracker.debian.org/tracker/CVE-2023-4863`
- **CVE records for CVE-2023-4863 and CVE-2023-5129**. `cveawg.mitre.org/api/cve/CVE-2023-4863`
- **Mandiant, time-to-exploit trends 2023** (15 de octubre de 2024). `cloud.google.com/blog/topics/threat-intelligence/time-to-exploit-trends-2023`
- **Linux kernel documentation, CVEs**. `docs.kernel.org/process/cve.html`
- **Jerry Gamblin, 2024 and 2025 CVE data reviews** (recuentos independientes). `jerrygamblin.com/2026/01/01/2025-cve-data-review/`

## Capítulo 12, El precio de ser dueño

- **Chromium, release cycle**. `chromium.googlesource.com/chromium/src/+/main/docs/process/release_cycle.md`
- **OpenHarmony documentation, web component overview** (ArkWeb). `gitcode.com/openharmony/docs`
- **BlackBerry, end of life for BlackBerry 10 services**. `blackberry.com/us/en/support/devices/end-of-life`
- **Mozilla, update on Firefox OS smartphones** (5 de febrero de 2016). `blog.mozilla.org/press-uk/2016/02/05/update-on-firefox-os-smartphones-and-next-steps-for-connected-devices/`
- **Canonical, "Growing Ubuntu for cloud and IoT"** (5 de abril de 2017). `canonical.com/blog/growing-ubuntu-for-cloud-and-iot-rather-than-phone-and-convergence`
- **Apple, letter from Tim Cook on Maps** (28 de septiembre de 2012). `apple.com/letter-from-tim-cook-on-maps/`
- **Federal Register, Huawei added to the Entity List** (21 de mayo de 2019). `federalregister.gov/documents/2019/05/21/2019-10616/addition-of-entities-to-the-entity-list`
- **Let's Encrypt, DST Root CA X3 expiration**. `letsencrypt.org/docs/dst-root-ca-x3-expiration-september-2021/`
- **Microsoft Learn, Windows 10 IoT Enterprise LTSC 2021 lifecycle**. `learn.microsoft.com/en-us/lifecycle/products/windows-10-iot-enterprise-ltsc-2021`
- **European Commission, Cyber Resilience Act**. `digital-strategy.ec.europa.eu/en/policies/cyber-resilience-act`

## Capítulo 13, La IA mueve las costuras


- **Anthropic, disrupting AI-orchestrated espionage** (13 de noviembre de 2025). anthropic.com/news/disrupting-AI-espionage
- **OWASP, LLM01 prompt injection**. genai.owasp.org/llmrisk/llm01-prompt-injection/
- **OWASP, Top 10 for LLM Applications 2026**. genai.owasp.org/resource/owasp-genai-llm-top-10-2026/
- **Google Project Zero, "From Naptime to Big Sleep"** (1 de noviembre de 2024). projectzero.google/2024/10/from-naptime-to-big-sleep.html
- **DARPA, AIxCC results** (agosto de 2025). darpa.mil/news/2025/aixcc-results
- **Anthropic, computer use announcement** (22 de octubre de 2024). anthropic.com/news/3-5-models-and-computer-use
- **W3C, Core Accessibility API Mappings**. w3.org/TR/core-aam-1.2/
- **Microsoft Playwright MCP repository**. github.com/microsoft/playwright-mcp


## Capítulo 14, Controlar el bit y la cadena de herramientas


- **Microsoft Surface, "Safer drivers, stronger devices"** (julio de 2025). techcommunity.microsoft.com/blog/surfaceitpro/safer-drivers-stronger-devices/4431411
- **Kernel Newbies, Linux 6.18**: controlador Rust Binder. kernelnewbies.org/Linux_6.18
- **Cloudflare, workerd open source** (27 de septiembre de 2022). blog.cloudflare.com/workerd-open-source-workers-runtime/
- **Zed blog on GPUI** (7 de marzo de 2023): renderizar "como un videojuego". zed.dev/blog/videogame
- **Google Cloud, open-sourcing gVisor** (2 de mayo de 2018). cloud.google.com/blog/products/identity-security/open-sourcing-gvisor-a-sandboxed-container-runtime
- **Rust blog, Rust 1.0** (15 de mayo de 2015). blog.rust-lang.org/2015/05/15/Rust-1.0/
- **GitHub, research on Copilot productivity** (7 de septiembre de 2022). github.blog/news-insights/research/research-quantifying-github-copilots-impact-on-developer-productivity-and-happiness/
- **METR, early-2025 study of experienced developers** (10 de julio de 2025) y su actualización de febrero de 2026. metr.org/blog/2025-07-10-early-2025-ai-experienced-os-dev-study/
- **Daniel Stenberg, "Death by a thousand slops"** (14 de julio de 2025). daniel.haxx.se/blog/2025/07/14/death-by-a-thousand-slops/


## Capítulo 15, Qué poseer y qué alquilar


- **Sovereign Tech Agency**. sovereign.tech/
- **Adam Langley, "BoringSSL"** (20 de junio de 2014). imperialviolet.org/2014/06/20/boringssl.html
- **Google, summer 2025 cybersecurity updates**: Big Sleep y CVE-2025-6965. blog.google/technology/safety-security/cybersecurity-updates-summer-2025/

- **Cloudflare, Pingora open source** (28 de febrero de 2024). blog.cloudflare.com/pingora-open-source/
- **CISA alert on xz utils** (29 de marzo de 2024). cisa.gov/news-events/alerts/2024/03/29/reported-supply-chain-compromise-affecting-xz-utils-data-compression-library-cve-2024-3094
- **Amazon EKS User Guide, Update the Kubernetes version of an EKS Auto Mode cluster**: Auto Mode reemplaza los nodos después de una actualización del plano de control. docs.aws.amazon.com/eks/latest/userguide/auto-upgrade.html


## Epílogo, La última capa es el ser humano


- **W3C, Understanding WCAG success criterion 2.3.1**. w3.org/WAI/WCAG22/Understanding/three-flashes-or-below-threshold.html
- **TÜV Rheinland, eye comfort certification brochure**. origin.tuv.com/content-media-files/master-content/services/products/1553-tuv-rheinland-eye-comfort-certification/tuv-rheinland-eye-comfort-certification_brochure_2023v2.pdf
- **EU-OSHA, Directive 90/270/EEC**: equipos con pantallas de visualización. osha.europa.eu/en/legislation/directives/5
- **Directive 90/270/EEC, Annex**. legislation.gov.uk/eudr/1990/270/annex
- **ASUS, flicker-free monitors**. asus.com/support/faq/1044285/

LA SERIE

# Cada píxel importa

Un libro para toda la pila y luego un libro para cada capa que decide lo que ven sus clientes. La serie complementaria, GDCD en código, trata cómo gobernar las capas que decida poseer.

| | |
|---|---|
| **Libro 1** | **Del bit al píxel**: toda la pila, la historia, poseer o alquilar (este libro) |
| **Libro 2** | **Del código al cristal**: la capa de renderizado, del código al cristal |
| **Libro 3** | **Del silicio a la pantalla**: seguridad desde el chip hasta la pantalla |
| **Libro 4** | **Datos en movimiento**: almacenamiento, redes, sistemas distribuidos |
| **Libro 5** | **La pila propia**: cadena de herramientas, IDE, construir un sistema propio |
| **Libro 6** | **The Pixel Runtime**: construir nuestro propio runtime (previsto) |

**Juan Cardena**

Arquitecto de IA empresarial y sistemas agénticos · blog.jcardena.com

Descargue la edición más reciente en blog.jcardena.com/es/ebooks/from-bit-to-pixel.

Cada píxel importa · Libro 1 · Del bit al píxel.