

LordOS: un escritorio GNU/Linux privado y rápido, construido sobre Debian y Wayland

Documento técnico: diseño, modelo de seguridad y estado actual

Versión **0.8.0 Beta** Autores: **Pablo Miranda, Federico Pereira** · iPERFEX

Fecha: **18 de octubre de 2026**

Resumen

LordOS es una distribución GNU/Linux libre basada en Debian 13 que trae un entorno de escritorio propio, construido desde sus componentes fundamentales sobre Debian y Wayland. En lugar de GNOME o KDE, tiene su propio servidor gráfico y compositor Wayland escritos en C (`lord-server`), su propio motor de interfaz (React corriendo sobre el motor de JavaScript QuickJS, con layout, pintado y texto en código C nativo), sus propias apps nativas y sus propios servicios de seguridad: un lanzador de apps en caja de arena, un broker de permisos, un registro de permisos por app, un firewall por app que aplica el kernel, firma de paquetes y una bóveda de contraseñas cifrada. Reutiliza lo que está maduro y muy revisado: el kernel Linux, Debian y sus paquetes, Mesa, PipeWire, el protocolo Wayland y unas pocas bibliotecas (FFmpeg, Poppler, libarchive, OpenSSL, Monocypher) que quedan encerradas en pequeños procesos ayudantes aislados.

Este documento describe el diseño y el estado actual de la versión **0.8.0 Beta** (octubre de 2026), el primer lanzamiento público, para un lector técnico que quiere verificar el proyecto en lugar de creer en él a ciegas. Cada afirmación remite a un archivo del repositorio público. Los números son los medidos en los registros de prueba del proyecto, e indican dónde se midieron. Lo que está diseñado pero no construido aparece marcado como tal.

1. Visión

1.1 El problema

Un escritorio Linux moderno se arma con muchos proyectos independientes: un servidor gráfico o compositor, un toolkit, un gestor de sesiones, un demonio de configuración, un llavero, un gestor de red, una interfaz de firewall y las apps. Cada uno es bueno en lo suyo, pero las costuras entre ellos son donde las cosas se vuelven lentas, inconsistentes o inseguras. Tres costuras en particular motivaron LordOS:

- **Peso.** Un escritorio que dibuja su interfaz con un motor web, o que carga varios toolkits a la vez, necesita cientos de megabytes antes de que el usuario abra nada. LordOS quiere andar cómodo en máquinas modestas y en una Raspberry Pi 5.
- **Confianza en las apps.** En un escritorio Linux tradicional, cada programa que corrés puede leer toda tu carpeta personal, grabar el micrófono, abrir cualquier conexión de red y leer el portapapeles. El aislamiento

existe, pero suele ser un agregado que el escritorio mismo no entiende.

- **Visibilidad.** Rara vez ves qué app está hablando con qué servidor, y menos todavía podés decir "esta app no puede usar internet" de una forma que la app no pueda esquivar.

1.2 Principios

El proyecto sigue una lista corta de reglas, escritas en [AGENTS.md](#) y [docs/ARCHITECTURE.md](#) :

1. **Ser dueños de lo que el usuario toca; reutilizar los cimientos.** El servidor gráfico, el compositor, el motor de interfaz, las apps y los servicios de seguridad son código de LordOS. El kernel, los paquetes de Debian, Mesa, PipeWire y el protocolo Wayland se reutilizan.
2. **Una política de dependencias estricta.** Dentro de LordOS no se usan X11, wlroots, libdrm, libinput, WebKit, GTK ni Qt. Las bibliotecas que leen datos no confiables las enlazan solo ayudantes aislados, nunca el compositor.
3. **La interfaz no puede ejecutar comandos.** El JavaScript de LordOS no tiene ninguna API para ejecutar programas arbitrarios. Hay tres excepciones angostas y sin argumentos (la shell de inicio de la Terminal, lanzar una app `.desktop` instalada y arrancar una app `.lpk` instalada en su caja de arena).
4. **Root solo a través de puertas cerradas.** Todo lo que necesita root pasa por una tabla fija de acciones validadas o por un servicio pequeño y dedicado con un protocolo cerrado.
5. **El camino por CPU siempre tiene que funcionar.** La composición en GPU es una optimización, nunca un requisito.
6. **El compositor nunca espera.** Ayudantes, apps y servicios hablan con `lord-server` por sockets no bloqueantes; una app colgada se cuelga solo a sí misma.
7. **Verificar en un sistema real.** Cada hito se comprueba en una máquina virtual con capturas reales antes de marcarlo como terminado, y no se declara compatibilidad que no se probó.

Archivos fuente

- [AGENTS.md](#)
- [docs/ARCHITECTURE.md](#)

2. Arquitectura

2.1 Capas

Las capas de LordOS

👤 VOS

Apps

Apps incluidas y el tema · apps .lpk aisladas · apps de Linux (GTK, Qt, navegadores)



🏠 PROPIO

Motor de interfaz de LordOS

React sobre QuickJS, layout, pintado y texto nativos en C. Lo usan las apps de LordOS; las apps de Linux no pasan por acá



🏠 PROPIO

Servidor gráfico y compositor Wayland de LordOS

lord-server, en C: KMS sin libdrm, evdev sin libinput, composición en CPU o GPU. Las apps de Linux y las .lpk aisladas se conectan acá como clientes Wayland



≡ REUTILIZADO

Espacio de usuario de Debian 13

dpkg y paquetes, Mesa (GPU opcional), PipeWire, libwayland, libxkbcommon, systemd



⚙️ KERNEL

Kernel Linux

DRM/KMS, evdev, Landlock, seccomp, cgroups v2, eBPF

👤 Vos

🏠 Propio

≡ Reutilizado

⚙️ Kernel

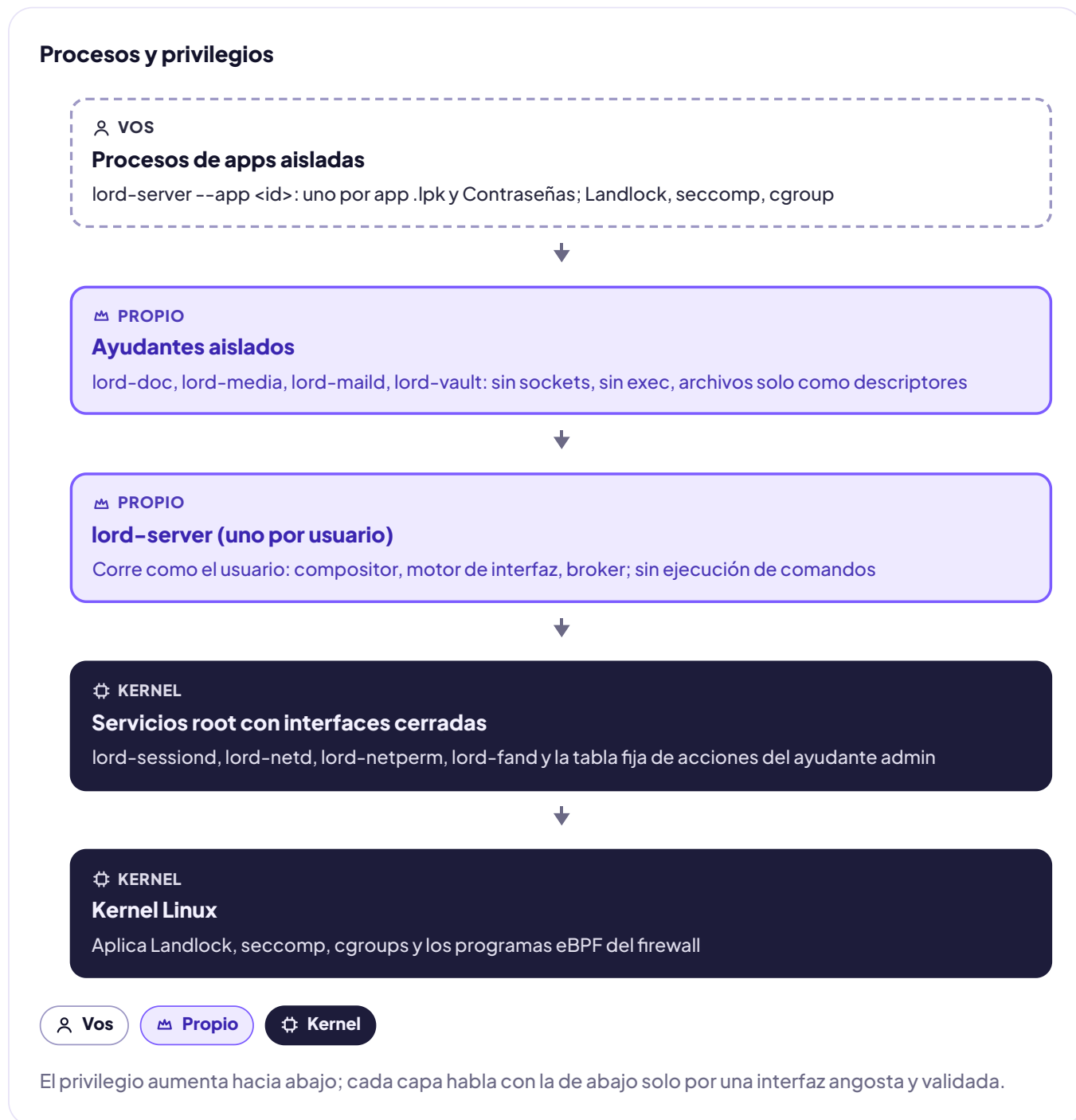
Las capas violetas son código propio de LordOS; las de abajo se reutilizan de Debian y Linux.

De abajo hacia arriba:

- **El kernel Linux** provee DRM/KMS para las pantallas, evdev para la entrada y las funciones de seguridad sobre las que se apoya LordOS: Landlock, seccomp-bpf, cgroups v2 y eBPF. En PC es el kernel de Debian; en la Raspberry Pi 5 es la compilación de LordOS del kernel de Raspberry Pi con Landlock activado (§7.4).
- **Debian 13** provee el espacio de usuario: **dpkg** y el archivo de paquetes, systemd, Mesa, PipeWire, **libwayland**, **libxkbcommon**, CUPS, BlueZ y el resto.
- **lord-server** es el servidor gráfico y compositor Wayland. Maneja las pantallas y la entrada directamente, compone y sirve el protocolo Wayland a las apps de Linux.

- **El motor de interfaz** vive dentro de **lord-server** : QuickJS corre la interfaz en React, y código C nativo hace el layout, el pintado y el texto.
- **Las apps** son de tres tipos: las apps incluidas y el tema, que corren en el contexto de JavaScript de **lord-server** ; las apps **.lpk** instaladas, cada una en su propio proceso aislado con su propia instancia del motor; y las apps de Linux, que se conectan al compositor como clientes Wayland comunes.

2.2 Procesos y privilegios



Un LordOS en marcha tiene pocos procesos, con roles bien separados:

Proceso	Corre como	Rol
<code>lord-sessiond</code>	root	pantalla de inicio de sesión, sesiones, prestar la pantalla y la entrada a la sesión visible, escritorio remoto
<code>lord-netd</code>	root	la red del sistema: DHCP, Wi-Fi, VPN; solo las acciones <code>net-*</code>
<code>lord-netperm</code>	root (capacidades limitadas)	reglas de red por app, y las restricciones de dispositivos y Bluetooth de las apps de Linux, aplicadas con programas eBPF de cgroup y de LSM
<code>lord-fand</code>	root (sin capacidades)	control de ventiladores según la temperatura
<code>lord-server</code>	el usuario	uno por sesión: compositor, motor de interfaz, broker
<code>lord-server --app <id></code>	el usuario, aislado	uno por app <code>.lpk</code> en ejecución (y Contraseñas)
<code>lord-doc</code> , <code>lord-media</code> , <code>lord-maild</code> , <code>lord-vault</code>	el usuario, aislados	ayudantes que procesan datos no confiables o secretos
pantalla de inicio de sesión	<code>lordos-greeter</code>	dibuja la interfaz de inicio; nunca verifica contraseñas por su cuenta

Cada proceso root tiene una interfaz angosta y documentada. La §6.3 las describe.

2.3 Qué es propio y qué se reutiliza

Propio (en este repositorio)	Reutilizado
Servidor gráfico y compositor Wayland (C)	Kernel Linux
Motor de interfaz: nodos nativos, layout flexbox, rasterizador, shaping de texto, bidi, métodos de entrada	Debian 13 y sus paquetes; <code>dpkg</code>
Renderizador de React (configuración de host para <code>react-reconciler</code>)	React 19.3.1, <code>react-reconciler</code> 0.34 (empaquetados en el tema)
Apps nativas y temas	QuickJS 2025-04-26 (incluido como código fuente)
Caja de arena, broker, permisos, firma, servicio de firewall, bóveda	Mesa (opcional), PipeWire, <code>libwayland</code> , <code>libxkbcommon</code>
Gestor de sesiones, servicio de red, servicio de ventiladores	systemd, systemd-resolved, BlueZ, CUPS, wpa_supplicant, OpenVPN, el demonio de strongSwan
Motor de paquetes Debian (OpenPGP, xz, HTTP, resolución de dependencias)	FFmpeg, Poppler, libarchive, OpenSSL, Monocypher (solo en ayudantes)
Driver de touchpad, emulador de terminal, intérprete de WebAssembly	stb (fuentes e imágenes), fuentes bajo la OFL

LordOS habla con varios servicios de Debian por sus propios protocolos, con clientes propios: D-Bus (BlueZ, perfiles de energía, fecha y hora), IPP (CUPS), el protocolo de PulseAudio (PipeWire) y el Secret Service. No se usan libdbus, sd-bus, libcups, libpulse, libbpf ni libapt.

2.4 Servicios del sistema

- **Red (`lord-netd`).** La red pertenece al sistema, no a una sesión. `lord-sessiond` arranca `lord-netd` al iniciar, con una ruta fija y sin argumentos, así la máquina tiene dirección antes de que alguien inicie sesión. Solo ejecuta las acciones cerradas `net-*` : su propio cliente DHCP y código netlink, Wi-Fi a través del `wpa_supplicant` de Debian, WireGuard con el código netlink propio de LordOS, OpenVPN, IPsec/IKEv2 a través del `charon` de strongSwan manejado por VICI con un cliente propio de LordOS y, opcionalmente, túneles OpenConnect, PPTP y compatibles con Fortinet. Habla solo con los ayudantes root de las sesiones, por un socket que acepta únicamente uid 0.
- **Ventiladores (`lord-fand`).** Un servicio root que lee la temperatura de la CPU cada 2 s y escribe el ciclo de trabajo de los ventiladores según un perfil, desde temprano en el arranque. Maneja solo los ventiladores que encontró su propio escaneo: el conector de ventilador de la Raspberry Pi 5 y controladores de placa madre de una lista permitida. Nunca escribe ventiladores de notebooks, GPU ni desconocidos. Al detenerse, o después de un fallo, restaura la configuración original del kernel. No ejecuta programas, no tiene red, ni capacidades, ni nodos de dispositivo.
- **Configuración de hardware.** Pantallas (modos desde KMS con nombres EDID, una reversión de 15 segundos con "¿mantener esta configuración?", persistencia por pantalla, rotación, Luz nocturna), Bluetooth a través de

BlueZ, batería y perfiles de energía, inactividad y tapa, fecha y hora, notificaciones para apps de Linux y apps predeterminadas. Los cambios llegan como uevents del kernel y señales de D-Bus; nada consulta en bucle.

Archivos fuente

- [server/src/main.c](#)
- [server/src/admin.c](#)
- [server/src/net/netd.c](#)
- [server/src/fan/fand.c](#)
- [server/src/hw](#)

3. El compositor

3.1 Pantallas sin libdrm

`lord-server` configura los modos de pantalla con los ioctls de DRM del kernel directamente (`kms.c`): enumera conectores, CRTCs y planos, reserva buffers y cambia de página en el refresco vertical. Elige la placa que tiene conectores, no simplemente `card0` , algo importante en placas donde el controlador de pantalla y la GPU son dispositivos separados (en la Raspberry Pi 5, `vc4` y `v3d`).

3.2 Dos formas de armar un cuadro

- **Camino por CPU.** La interfaz y las ventanas de las apps se copian a un lienzo, y se presenta un buffer "dumb" con un cambio de página. No necesita nada más que el kernel. Es el predeterminado cuando no hay una placa con 3D, y `LORD_GPU=0` lo fuerza (el modo seguro del menú de arranque lo usa).
- **Camino por GPU.** GBM, EGL y GLES2 de Mesa (`server/src/gpu/gpu.c`). La interfaz pasa a ser una textura, y en cada cuadro se sube solo la región que cambió. Las ventanas de las apps no se copian al lienzo de la interfaz: la interfaz deja un hueco transparente donde va cada ventana, y las texturas de las ventanas se dibujan debajo, así una app que solo cambia su contenido no provoca un repintado de la interfaz. Las apps pueden entregar buffers de GPU (`linux-dmabuf`) sin copias, y una app a pantalla completa con un buffer adecuado se muestra directamente (scanout directo). El cursor usa el plano de cursor por hardware.

El camino se elige al arrancar: una placa sin 3D (simplifiedrm, bochs, virtio-gpu sin virgl y otras) usa la CPU sin siquiera cargar Mesa; un renderizador OpenGL por software también vuelve a la CPU, porque componer con él cuesta más que el camino por CPU propio de LordOS. Si un cuadro por GPU falla, la GPU se apaga y el camino por CPU toma el control.

Estado: el camino por GPU se probó con el renderizador por software de Mesa en máquinas virtuales. Falta probarlo en placas AMD y Nvidia reales.

3.3 Ventanas y entrada

- **Gestión de ventanas en C.** Mover y redimensionar corre en C, con un evento por cuadro y sin JavaScript, y la ventana que se mueve se dibuja desde su propia capa. Los cuatro bordes y las esquinas redimensionan, con cursores propios de LordOS.

- **Entrada sin libinput.** Teclados, mouses y tabletas se leen desde evdev. Las distribuciones de teclado vienen de `libxkbcommon` (todas las distribuciones XKB, variantes y compose).
- **Touchpad y pantalla táctil.** LordOS tiene su propio driver multitáctil (`server/src/tp`): aceleración del puntero, tocar para hacer clic y clic con dos dedos, rechazo de la palma, desplazamiento cinético y natural, pellizco y deslizamientos con tres dedos. Los gestos llegan tanto a la interfaz de LordOS como a las apps de Linux (`zwp_pointer_gestures_v1`). Se prueba con gestos grabados en `make test` .

3.4 Compatibilidad con Wayland

El compositor (`server/src/wl/wl.c`) usa solo `libwayland-server` y código generado a partir del XML de los protocolos. Ofrece: `wl_compositor` , `wl_subcompositor` , `wl_shm` , `wl_output` (uno por pantalla), `xdg_wm_base` con popups, `wl_seat` (puntero, teclado, táctil), `wl_data_device_manager` (un portapapeles compartido con LordOS), `zxdg_decoration_manager_v1` , el protocolo de decoración del servidor de KDE (para que las apps GTK 3 como Firefox tengan el marco del tema), `zxdg_output_manager_v1` , `wp_fractional_scale_v1` , `wp_viewporter` , `zwp_linux_dmabuf_v1` v4, `wp_presentation` , `zwp_relative_pointer_manager_v1` , `zwp_pointer_constraints_v1` , `zwp_pointer_gestures_v1` y `wp_cursor_shape_manager_v1` .

Las ventanas de Linux entran al gestor de ventanas del SDK como cualquier otra, con título, foco, maximizar, minimizar e ícono en el Dock. Las apps que piden decoración del lado del servidor reciben el marco del tema; las que dibujan el suyo (GTK 4) lo conservan, y se recorta su sombra. Las apps GTK 4 y libadwaita siguen el modo claro u oscuro, el color de acento y el orden de los botones a través de una implementación de `org.freedesktop.portal.Settings` que `lord-server` sirve él mismo en el bus de sesión; GTK 3 los sigue a través del archivo de claves de GSettings.

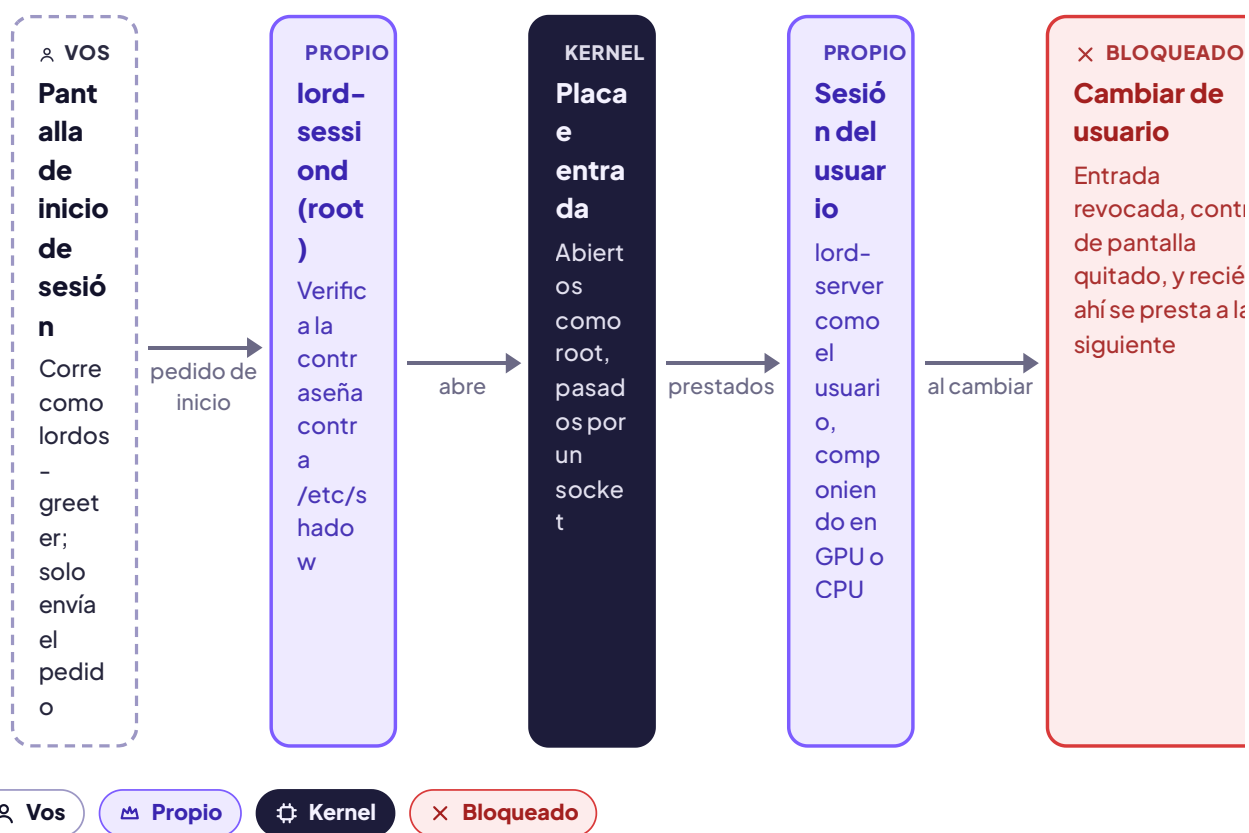
Xwayland todavía no está soportado: LordOS no corre apps que solo hablan X11 (Xwayland está planeado antes de la 1.0, §10.3). También quedan pendientes: arrastrar y soltar, la selección primaria, los métodos de entrada de LordOS dentro de apps de Linux (`text-input`), la activación (`xdg-activation`), la sincronización explícita y los íconos de app en SVG. No se ofrece a las apps ningún protocolo de captura de pantalla.

3.5 Varias pantallas y HiDPI

Todas las pantallas conectadas se manejan a la vez, extendidas o en espejo, en un único espacio lógico, con una disposición que se recuerda por identidad EDID y manejo de conexión en caliente. Cada pantalla se pinta a su propia escala (100, 125, 150, 175 o 200 %), así el texto y los vectores se rasterizan a su resolución real. Las apps de Linux reciben un `wl_output` por pantalla y escalado fraccional. Se probó con dos salidas virtuales (1280×800 y 2560×1600) en los dos caminos; no se probaron monitores reales, paneles de notebook ni hubs DisplayPort.

3.6 Sesiones e inicio de sesión

Prestarle la pantalla a una sesión



La sesión visible usa la GPU sin ser nunca root; al cambiar, se le revoca el acceso antes de que la siguiente reciba los dispositivos.

lord-sessiond es el gestor de sesiones propio de LordOS:

- **La pantalla de inicio de sesión** es un **lord-server** que corre como el usuario sin privilegios **lordos-greeter**, con un tema especial. Envía los pedidos de inicio de sesión a **lord-sessiond**; **las contraseñas siempre las verifica el proceso root** contra **/etc/shadow**, nunca el JavaScript de la pantalla de inicio. Los intentos fallidos esperan 1, 2, 4... segundos, hasta 5 minutos.
- **Una sesión por usuario**, y todas pueden vivir a la vez (cambio rápido de usuario). Cada una es un **lord-server** que arranca como root y suelta los privilegios enseguida.
- **Préstamo de la pantalla.** **lord-sessiond** abre la placa gráfica y los dispositivos de entrada como root y se los presta a la sesión que está en el monitor, por un socket Unix. Al cambiar, los revoca: **EVIOCREVOKE** hace que los descriptores de entrada de la sesión anterior dejen de funcionar, y **DROP_MASTER** le quita el control de la pantalla, antes de darle algo a la sesión siguiente. La sesión visible sigue componiendo en la GPU sin ser nunca root. Una sesión fuera del monitor dibuja en memoria, y solo si alguien la está mirando de forma remota.
- **Bloquear saca de verdad la sesión del monitor**; no es una capa dibujada encima. Si la sesión del usuario se cuelga, la pantalla de inicio sigue funcionando.
- **Arranque.** Un tema de GRUB de LordOS dibuja el menú de arranque, y un tema de Plymouth de LordOS dibuja la pantalla de carga (la corona con una barra de progreso). El traspaso no tiene ningún cuadro negro ni texto de consola: Plymouth deja de dibujar y suelta el control de la pantalla mientras su último cuadro queda visible,

`lord-sessiond` toma el control de la pantalla y pone el primer cuadro real de la pantalla de inicio de sesión, y recién entonces Plymouth termina, dejando la pantalla de carga hasta que se reemplaza.

Escritorio remoto. El mismo servicio ofrece el escritorio a un navegador de la red local en

`https://<dirección>:5673` (5673 deletrea "LORD" en un teclado de teléfono). Viene **desactivado por defecto** y se activa en Ajustes → Compartir. Usa TLS 1.2 o superior con un certificado P-256 autofirmado cuya huella muestra Ajustes para comparar, una política de seguridad de contenido estricta, controles de mismo origen, esperas por IP ante inicios fallidos y una cookie de sesión `Secure`, `HttpOnly` y `SameSite=Strict`. La página de inicio nunca lista las cuentas de usuario. El proceso que atiende una conexión suelta root apenas sabe de quién es la sesión. Las actualizaciones de pantalla se envían como PNG cuando está quieta y como JPEG mientras se mueve. El audio está diseñado pero no construido. El acceso desde internet no está previsto; se recomienda una VPN.

Archivos fuente

- [server/src/kms.c](#), [server/src/gpu/gpu.c](#), [server/src/outputs.c](#)
- [server/src/input.c](#), [server/src/tp](#), [server/src/ui/grab.c](#)
- [server/src/wl/wl.c](#), [server/src/ui/portal.c](#)
- [server/src/remote/sessiond.c](#), [server/src/remote/monitor.c](#), [server/src/remote/sess.c](#)
- [tools/boot/plymouth](#), [tools/iso/grub](#), [docs/SESSIONS.md](#)

4. El motor de interfaz

4.1 No es un navegador

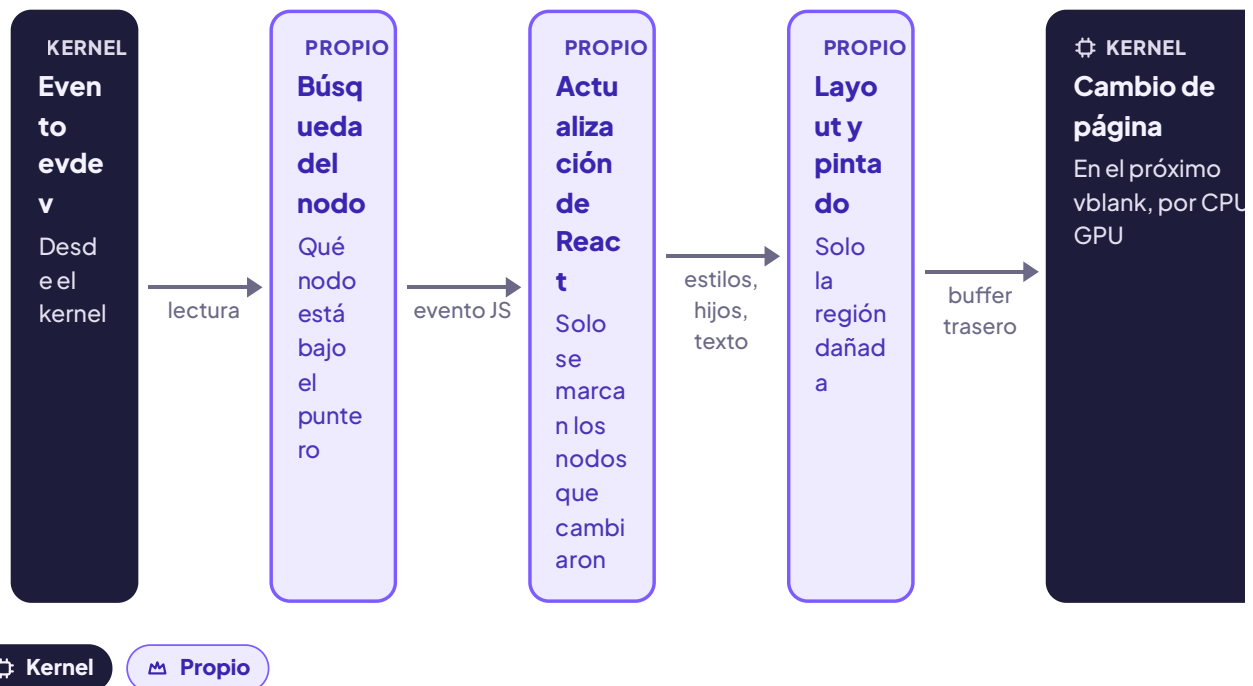
La interfaz de LordOS está escrita en JavaScript con React, pero no hay ningún motor web debajo. WebKit o Chromium costarían de 100 a 300 MB de RAM por proceso y no serían código de LordOS. En cambio, el tema y las apps usan un renderizador de React (`runtime/index.js`) que crea y actualiza **nodos nativos** en C: `View`, `Text`, `Image`, más superficies para ventanas Wayland y multimedia. Los estilos son un subconjunto de CSS (flexbox, colores, bordes redondeados y por lado, sombras, desenfoque, transformaciones, transiciones) que aplica código C controlado por el proyecto. El modelo se parece a cómo un framework móvil renderiza React a vistas nativas, salvo que acá el lado "nativo" es propio de LordOS.

4.2 Piezas

- **JavaScript:** QuickJS 2025-04-26, un motor ES2023 pequeño en C, incluido como código fuente y embebido en `lord-server`.
- **React:** React 19.3.1 con `react-reconciler` 0.34, empaquetados con esbuild dentro de cada tema. El tema predeterminado con todas las apps pesa 367 KB.
- **Layout:** implementación propia de flexbox (`layout.c`), con ajuste de línea, bordes lógicos de inicio y fin, y espejado de derecha a izquierda.
- **Pintado:** rasterizador 2D propio con rectángulos redondeados con antialiasing, sombras en caché, desenfoque a resolución reducida y seguimiento de daño: solo se repinta la región que cambió. En ARM, bucles NEON opcionales aceleran las copias y las máscaras, verificados bit a bit contra el código C.

- **Texto:** los glifos se rasterizan con `stb_truetype`. El shaping es código OpenType propio de LordOS (sustituciones GSUB y posicionamiento GPOS, emoji a color), con bidi Unicode completo, unión de letras árabes, shaping índico y el Universal Shaping Engine (409 de 409 oraciones de prueba idénticas a HarfBuzz en `tests/hb_compare.py`), corte de línea CJK, separación de palabras en tailandés y fuentes de respaldo por escritura. `lord-server` no enlaza HarfBuzz, FreeType, fontconfig, fribidi ni ICU.
- **Métodos de entrada:** coreano (hangul), japonés (kana y SKK) y chino (pinyin), en `server/src/ime.c`, sin ibus ni fcitx.

De un movimiento del mouse a un cuadro nuevo



El motor repinta solo lo que cambió y cambia de página en el siguiente refresco vertical.

4.3 Cómo llega el JavaScript al sistema

El lado JavaScript llama a servicios nativos a través de las APIs `lord.*` implementadas en C (`server/src/ui/js_*.c`): `lord.fs` (archivos), `lord.settings`, `lord.net`, `lord.pkg`, `lord.media`, `lord.mail`, `lord.vault`, `lord.fw`, `lord.monitor`, `lord.fan` y otras. Cada llamada valida sus argumentos en C. Las reglas son estrictas:

- `lord-server` corre sin root, y **ninguna API ejecuta comandos arbitrarios**.
- `lord.fs` escribe solo en la carpeta personal y en los montajes propios del usuario, no tiene un "borrar" general (solo mover a la Papelera) y rechaza las carpetas que el sistema lee para lanzar o instalar algo o que guardan decisiones de seguridad: `~/.local/share/applications`, `~/.config/autostart`, la carpeta de apps instaladas, `settings.json`, el archivo de permisos, las reglas del firewall, el archivo de cuentas de correo y la carpeta de la bóveda.
- Todo lo que necesita root pasa por `admin.c`: una tabla cerrada de acciones con argumentos validados del lado root, rutas de programa fijas y configuración generada por LordOS, nunca texto del usuario pasado a una shell. Las contraseñas pasan por la entrada estándar, nunca por la línea de comandos.

- El trabajo largo nunca bloquea un cuadro: ayudantes y servicios se consultan desde el bucle principal sin esperar.

4.4 Idiomas

La interfaz viene en inglés (el predeterminado), español y portugués, y se suman más idiomas como paquetes de idioma sin cambiar código. El texto pasa por `t('namespace.key')` del SDK, con plurales y formatos regionales. Toda la interfaz se espeja para los idiomas de derecha a izquierda, y la región (`LANG / LC_*`) se pasa a las apps de Linux. `node tools/i18n-check.mjs` verifica que cada clave exista en todos los idiomas y que no aparezcan marcas de terceros en la interfaz.

Archivos fuente

- [runtime/index.js](#), [sdk/index.js](#)
- [server/src/ui/js.c](#), [server/src/ui/node.c](#), [server/src/ui/layout.c](#), [server/src/ui/paint.c](#)
- [server/src/ui/shape.c](#), [server/src/ui/otf.c](#), [server/src/ui/bidi.c](#), [server/src/ime.c](#)
- [server/src/ui/js_sys.c](#), [server/src/fsops.c](#)

5. Aplicaciones

5.1 Apps nativas

LordOS trae alrededor de treinta apps nativas, todas apps React sobre el mismo motor y SDK: Archivos, Ajustes, Terminal, Tienda, Editor de texto, Notas, Calculadora, Monitor, Visor de PDF, Fuentes, Emoji y símbolos, Escáner, Fotos, Reproductor, Capturas, Cámara, Grabadora, Contraseñas, Discos, Copias de seguridad, Registros, Reloj, Ayuda, Mail, Calendario, Contactos, Recordatorios, Acerca de y el instalador. Algunas para destacar:

- **Terminal:** emulador propio compatible con xterm, en C, con pestañas, paneles divididos, perfiles, imágenes en línea y Sixel, y sesiones que sobreviven a un reinicio de `lord-server`.
- **Monitor:** CPU, memoria, discos, procesos, refrigeración y conexiones de red por app en un mapa del mundo, con búsqueda de país desde una base de datos local. Su hilo de muestreo existe solo mientras la app está abierta.
- **Discos:** lector SMART propio, formateo y pendrives de arranque verificados con SHA-256.
- **Copias de seguridad:** instantáneas con enlaces duros y cifrado opcional, que corren con prioridad baja.

5.2 Apps `.lpk` instalables

Las apps nativas de terceros se empaquetan como `.lpk`: un tar comprimido con gzip con un manifiesto, el `app.js` empaquetado, un ícono obligatorio y traducciones y recursos opcionales. Una app no trae React ni el SDK; usa los del sistema, y eso la mantiene chica (la app de ejemplo Fractal pesa 5 KB). Se instalan sin root en `~/.local/share/lordos/apps/<id>/`. El instalador rechaza enlaces, dispositivos, rutas absolutas, `..`, caracteres raros, más de 2.000 archivos o 64 MB, y reemplaza la versión anterior de forma atómica. Cada una corre aislada (§6.4).

5.3 La Tienda y los paquetes de Debian

La Tienda muestra las apps de Debian por categoría con búsqueda, las apps `.lpk` y los archivos `.deb` locales, y siempre muestra un plan antes de instalar o quitar. Los paquetes de Debian los maneja **el motor de paquetes propio de LordOS** (`server/src/pkg`): lee las fuentes de apt, verifica las firmas de `InRelease` con su propio código OpenPGP (RSA y Ed25519), rechaza índices vencidos, repetidos, de otra suite o con datos agregados al final, verifica cada índice y cada `.deb` contra los valores SHA-256 del índice firmado, y resuelve dependencias (planes idénticos a los de apt en 12 casos de prueba). Solo el último paso es de Debian: `dpkg` desempaqueta y configura, y sigue siendo el único que escribe su base de datos, así `apt` sigue funcionando. LordOS no tiene un cliente TLS propio, así que las fuentes `https://` se descargan por `http://`; la integridad descansa en las firmas y los hashes, pero los nombres de los paquetes descargados quedan visibles en la red.

5.4 Apps de Linux

Las apps de Linux con un archivo `.desktop` aparecen en el lanzador de apps. El lanzamiento pasa por `lord.apps.launch(id)`: la interfaz solo puede nombrar un archivo `.desktop` instalado; `lord-server` lo vuelve a leer, lo verifica, arma `argv` a partir de su línea `Exec` según la especificación y lo ejecuta con `execve`, sin shell ni argumentos del llamador, dentro de su propio scope de cgroup que pertenece a root. Chromium y Firefox vienen preinstalados en la ISO y se verificaron con el marco de LordOS. **LordOS no aísla las apps de Linux de forma predeterminada**: corren como el usuario, como en cualquier Debian. Tienen atribución, el firewall por app y las restricciones por app que fije el dueño (§6.6), incluida una vista opcional "sin acceso a tus archivos" que aplica el lanzador.

5.5 WebAssembly

LordOS incluye su propio intérprete de WebAssembly en C (`server/src/wasm`), expuesto como el objeto global estándar `WebAssembly`, así una app nativa puede traer módulos compilados desde C, C++, Rust o Zig. Cubre WebAssembly 2.0 sin SIMD (el MVP más extensión de signo, conversiones saturadas, multi-valor, memoria masiva y tipos de referencia), pasa 26.290 de 26.290 aserciones de la suite oficial de pruebas en la VM, y convierte cada error en un trap en lugar de un fallo. En el ejemplo del proyecto corrió de 3 a 5 veces más rápido que el mismo código en JavaScript.

5.6 Instalar LordOS

- **ISO live** para PC de 64 bits, con un instalador gráfico (UEFI y BIOS).
- **Imagen para Raspberry Pi 5** para tarjeta SD o NVMe (§7.4).
- **Sobre un Debian 13 existente**: `curl -fsSL https://lordos.org/install.sh | sh`. El script pregunta antes de cambiar nada, descarga `SHA256SUMS` y verifica su firma Ed25519 con la clave de releases embebida en el script, se niega a seguir sin una firma válida, verifica el SHA-256 del `.deb`, lo instala con apt y configura el sistema como la ISO. Los archivos del release se publican en GitHub Releases.

Archivos fuente

- [apps](#), [server/src/lpk.c](#), [tools/lpk.mjs](#)
- [server/src/pkg](#), [server/src/apps.c](#)
- [server/src/wasm/wasm.c](#), [server/src/ui/js_wasm.c](#)
- [tools/install/install.sh](#)

6. Seguridad

6.1 Modelo de amenazas

LordOS está diseñado contra estas amenazas:

- **Una app instalada maliciosa o comprometida** que quiere los archivos, las claves, el micrófono, la cámara, el portapapeles o la red del usuario.
- **Documentos y mensajes maliciosos**: PDF, archivos comprimidos, fuentes, imágenes, video y correo armados para explotar un lector.
- **Una app de Linux que quiere sacar datos**, o escaparse de las reglas que se le pusieron.
- **Un atacante en la red local** que intenta usar el escritorio remoto o alterar descargas.
- **Paquetes y releases alterados**.

Fuera del modelo, por ahora: un kernel o una cuenta root comprometidos; ataques físicos; canales laterales; y apps de Linux que el dueño no restringió, que corren como el usuario, como en cualquier Debian, y pueden leer sus archivos. A las apps de Linux se las limita solo con las restricciones que el dueño les fija (§6.6), y solo cuando se abren desde el lanzador de apps.

6.2 Mínimo privilegio

- `lord-server` corre como el usuario. En una sesión administrada nunca tiene los dispositivos de pantalla ni de entrada por sí mismo: `lord-sessiond` se los presta y se los revoca.
- Las bibliotecas que leen datos no confiables las enlazan solo ayudantes: FFmpeg y libva, `lord-media` ; Poppler y libarchive, `lord-doc` ; OpenSSL, `lord-maild` y `lord-sessiond` ; Monocypher, `lord-vault` .
- Los ayudantes reciben archivos y dispositivos como descriptores abiertos por `lord-server` ; nunca abren rutas.
- El compositor nunca espera a un ayudante ni a una app, así un fallo o un cuelgue queda contenido.

6.3 Los procesos root y sus interfaces cerradas

Proceso	Acepta	Nunca hace
ayudante <code>admin.c</code>	una tabla fija de acciones, con argumentos validados como root	acciones genéricas, texto del usuario en una shell
<code>lord-sessiond</code>	pedidos de inicio de sesión desde la pantalla de inicio; JSON de sesión corto (≤ 4 KB)	dejar que la pantalla de inicio verifique contraseñas; leer píxeles como root
su hijo de ubicación en cgroups	<code>enter <scope> <limits></code> solo para el proceso que llama, desde la hoja <code>server</code> de una sesión, corriendo un binario del directorio de <code>lord-server</code>	ubicar procesos arbitrarios; devolverle el árbol al usuario
<code>lord-netd</code>	acciones <code>net-*</code> solo de pares root	acciones que no son de red; ejecutar nada que venga de una sesión
<code>lord-netperm</code>	<code>RELOAD</code> sin datos	ejecutar programas; recibir argumentos; abrir sockets de internet
<code>lord-fand</code>	<code>STATUS</code> para cualquiera; <code>PROFILE</code> , <code>CURVE</code> , <code>FORCE</code> para administradores	nombrar rutas o dispositivos; ejecutar programas; usar la red

6.4 Qué está aislado, y la caja de arena de las apps

No todos los programas de LordOS están aislados. Esto es exactamente lo que se aplica a cada tipo:

Tipo	Proceso	Caja de arena	Scope de cgroup	Broker y permisos	Firewall y restricciones (§6.6)
Apps <code>.lpk</code> y Contraseñas	propio (<code>lord-server --app <id></code>)	Landlock, seccomp, sin capacidades	propio, de root, con límites	cada llamada se verifica en C	reglas de red y las ocho restricciones que no son de red, en el broker
Las demás apps incluidas	dentro de <code>lord-server</code>	no (son parte del escritorio)	el de <code>lord-server</code>	sin broker; cámara, micrófono y pantalla se verifican en C	como "LordOS": reglas de red; cámara, micrófono, pantalla y notificaciones
Apps de Linux desde el lanzador de apps	propio	ninguna de forma predeterminada; con "sin acceso a tus archivos", espacios de nombres de usuario, montaje y PID propios más Landlock, desde el próximo inicio	propio, de root	no	reglas de red; cámara, micrófono, USB, sockets Bluetooth, archivos y notificaciones; no el portapapeles
Programas de la Terminal	hijos de la shell de la Terminal	no	el de <code>lord-server</code>	no	como "LordOS": reglas de red

Un programa iniciado de otra forma, fuera del lanzador de apps, no está en ningún scope de app y no recibe reglas por app. No se usan Flatpak ni bubblewrap.

Cada app `.lpk` , y la app Contraseñas, corre como `lord-server --app <id>` : el mismo binario en "modo app", lanzado por `lord-server` con un id de paquete validado como único argumento. El proceso se conecta al compositor como cliente Wayland y abre el socket del broker, después aplica la caja de arena, y recién entonces carga el código de la app (`server/src/sec/sandbox.c`):

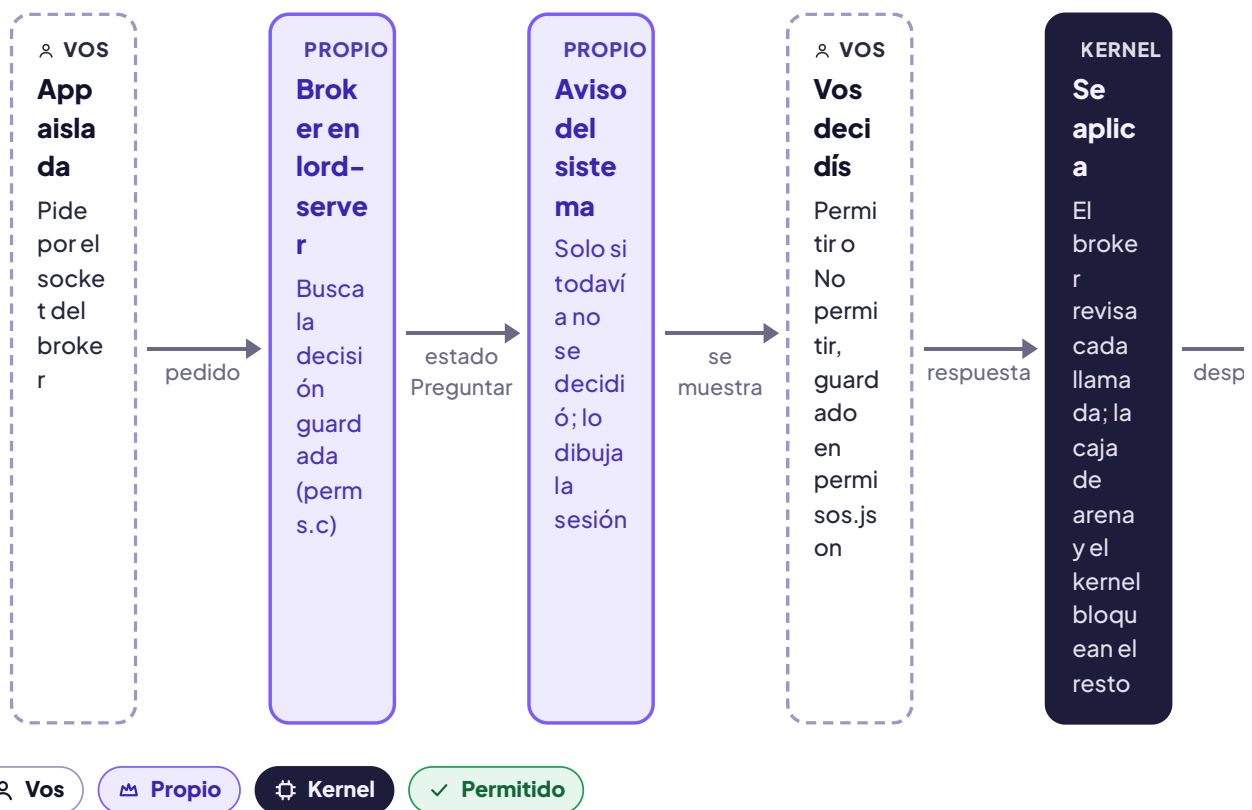
- **Landlock** (ABI hasta 6): acceso de solo lectura al paquete, al runtime compartido de las apps, a las fuentes y a las bibliotecas; lectura y escritura solo en `~/.local/share/lordos/appdata/<id>` ; acotamiento de sockets Unix abstractos y de señales.
- **seccomp-bpf**: una lista de llamadas al sistema permitidas en todos los hilos (TSYNC), sin `ptrace` , `mount` , `kexec` , `bpf` , `perf_event_open` , `keyctl` , carga de módulos ni `execve` . Se rechazan los sockets Unix nuevos y `socketpair` , porque Landlock hasta la ABI 6 no filtra `connect` a sockets Unix con nombre; así la app queda lejos del bus de sesión, del servicio de impresión y del socket de Wayland.
- **Sin privilegios**: `no_new_privs` , un conjunto de capacidades vacío y el conjunto límite descartado.

- **cgroup v2:** `app-lordos-lpk-<id>.scope` , creado por root, con `memory.high` 192 MB, `memory.max` 512 MB y un núcleo de CPU.
- **Red:** sin el permiso `network` (declarado y permitido), seccomp rechaza los sockets de internet y Landlock rechaza connect y bind de TCP. No se usan espacios de nombres de red, porque crear uno necesita privilegios que `lord-server` no tiene.

Si falta una función del kernel, la app arranca igual y el registro dice qué parte falta. La app de prueba maliciosa a propósito (`tests/lpk-evil`) fracasa en todo lo que intenta; `build/sandbox-test` pasa 14 de 14 verificaciones para apps y 14 de 14 con red. Medido en una VM: unos 13,7 MB de RSS por app aislada, un arranque mediano de 106 ms y 0,89 ms para aplicar la caja de arena.

6.5 El broker, los permisos y el diálogo de archivos

Cómo viaja un pedido de permiso



La app nunca responde su propio aviso: la decisión se guarda donde no puede escribir y se revisa en C en cada pedido.

La app aislada habla con `lord-server` solo por un socket `SOCK_SEQPACKET` . Cada pedido se verifica en C: acceso a archivos solo dentro de la carpeta de datos propia de la app; lectura del portapapeles solo con el permiso `clipboard` ; pedidos de permiso respondidos según la decisión guardada o, si no hay, con un **aviso dibujado por el sistema** (*Permitir/No permitir*); cámara y micrófono solo a través de `lord-server` .

- **Las decisiones** se guardan por usuario en `~/.config/lordos/permisos.json` , fuera del alcance de la app. La consulta es $O(1)$.

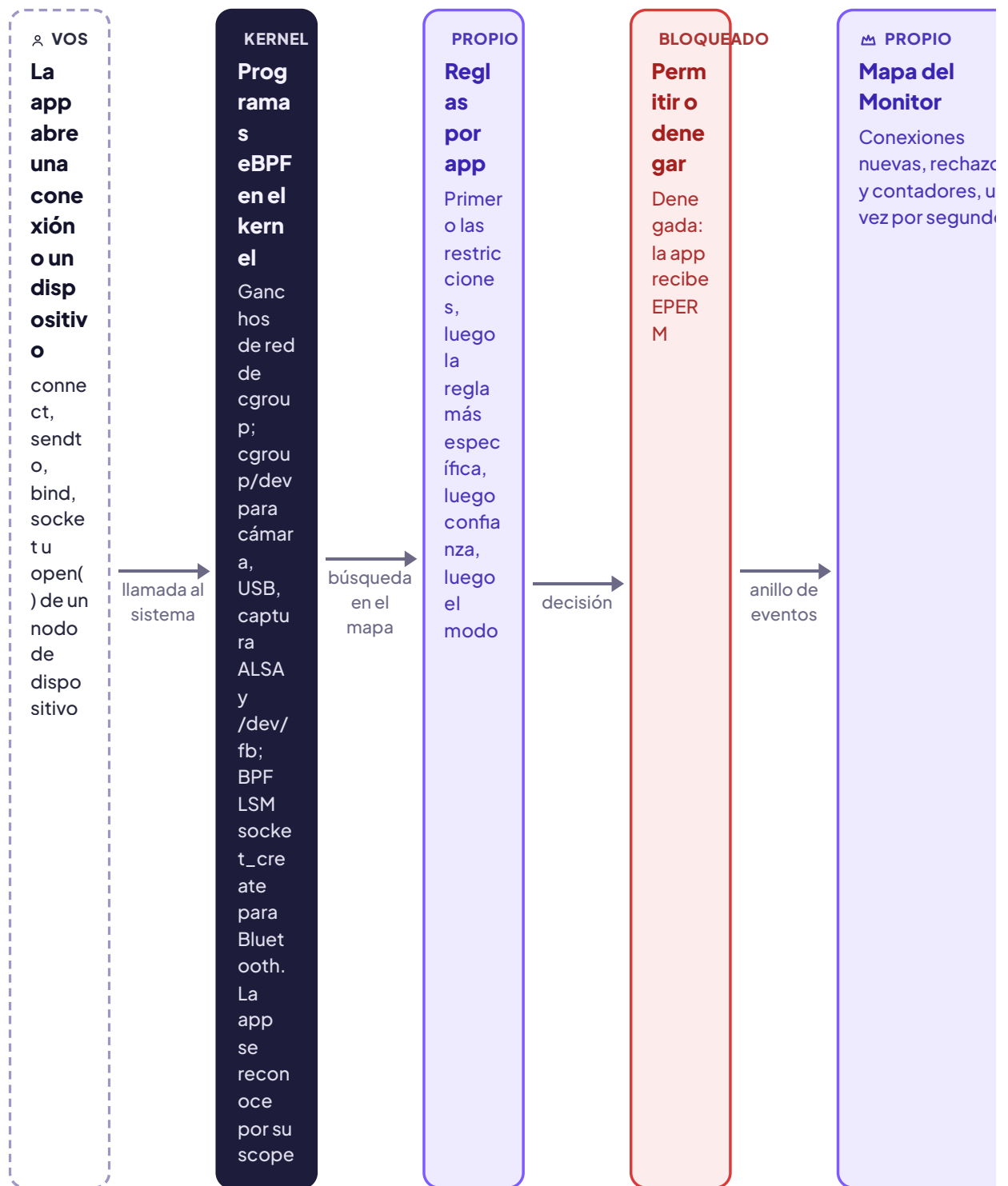
- **El diálogo de archivos es el powerbox propio de LordOS** (`sdk/powerbox.jsx`), no `xdg-desktop-portal` : lo dibuja `lord-server` , que abre el archivo elegido y pasa solo ese descriptor. La única interfaz de portal de escritorio que implementa LordOS es `org.freedesktop.portal.Settings` , para la apariencia de GTK.
- **Micrófono:** `lord-server` es dueño del stream de PipeWire y le da a la app un anillo compartido de solo lectura.
- **Cámara:** `lord-server` abre el dispositivo y se lo da solo a `lord-media` ; la app recibe una vista de solo lectura de los cuadros decodificados y tiene que confirmar cada uno, y `lord-server` valida cada confirmación.
- **Revocación:** denegar la cámara o el micrófono en Ajustes los corta al instante (se mata el `lord-media` de la app; se cierra el stream de PipeWire). El permiso `network` se decide al lanzar la app.
- **Llavero:** los pedidos al llavero de una app aislada llevan un dueño que `lord-server` fija a partir del id verificado de la app.

Las restricciones les ganan a los permisos. Una restricción del firewall (§6.6) gana sobre cualquier decisión guardada: el pedido se rechaza sin aviso, aunque la app haya tenido permiso antes, y restringir una app en marcha le quita la cámara y el micrófono al instante, como revocar.

Límites. No hay "permitir una vez". Algunos permisos declarables (Bluetooth, lanzar apps, notificaciones) todavía no tienen llamada en el broker, así que las apps aisladas directamente no pueden usar esas capacidades; escuchar en un puerto depende del permiso de red. Una app de Linux que corre como el usuario podría editar el archivo de permisos, salvo que esté restringida con "sin acceso a tus archivos".

6.6 El firewall por app, sus restricciones y el Monitor

El camino de una conexión o un dispositivo



👤 Vos

⚙️ Kernel

🏠 Propio

❌ Bloqueado

La decisión ocurre en el kernel cuando se abre una conexión, un socket o un dispositivo; nada corre por cada paquete.

lord-netperm aplica las reglas por app en el kernel. Las apps de Linux abiertas desde el lanzador de apps corren en `app-lordos-<id>.scope`, las apps `.lpk` en `app-lordos-lpk-<id>.scope`, y el resto de la sesión cuenta como

lordos . Todos los scopes pertenecen a root. El servicio carga siete programas de red (`connect4/6` , `sendmsg4/6` , `bind4/6` , `sock_create`), un programa de dispositivos (`cgroup/dev`) y un programa LSM por cgroup sobre `socket_create` en el cgroup de `lordos.service` , identifica la app de cada proceso por su ancestro de cgroup y devuelve permitir o denegar (`EPERM`). Los programas están compilados dentro del binario y se cargan con un cargador propio de LordOS sobre la llamada al sistema `bpf()` , que además lee el BTF del kernel para encontrar el gancho LSM; no se usan libbpf ni bcc. Si el kernel rechaza el programa de dispositivos o el LSM, los programas de red siguen funcionando y el servicio lo avisa.

- **Las reglas** admiten dominios exactos, comodines, redes IPv4/IPv6, `lan` , `internet` , puertos, protocolos y dirección. Primero van las restricciones (sin internet, sin LAN, sin red, sin entrantes); después gana la regla más específica, y denegar le gana a permitir a igual especificidad; después las apps de confianza; después el modo (silencioso, el predeterminado, o apagado). `npcompile.c` precalcula todo esto en mapas hash y de coincidencia por prefijo más largo. Una prueba corre el código de decisión del kernel contra el decisor de referencia sobre 480.000 conexiones aleatorias: 0 diferencias.
- **Los nombres de dominio** vienen solo del resolutor del sistema: la interfaz de monitoreo de `systemd-resolved` o, sin él, las respuestas del puerto 53 de los servidores configurados, leídas por un socket de paquetes que solo recibe. Nunca se confía en respuestas que una app recibió por sus propios sockets. Por eso, una app que resuelve nombres por su cuenta con DNS cifrado (DNS sobre HTTPS dentro de un navegador, por ejemplo) no queda alcanzada por las reglas de dominio: una regla "permitir solo example.com" la sigue bloqueando, pero una regla "bloquear example.com" no la frena. Las reglas por dirección y por red, el valor predeterminado de la app y sus restricciones siguen valiendo, y Ajustes lo dice debajo de las reglas.
- **Las reglas en la interfaz.** Ajustes → Privacidad → Reglas de red por app muestra el valor predeterminado de cada app ("Permitir todo" o "Bloquear todo salvo las reglas") y sus reglas en el orden en que ganan, con cuándo coincidió cada una por última vez; la hoja para agregar una regla revisa lo que se escribe mientras se escribe. En Monitor → Red, "Permitir / Bloquear este dominio" sobre un destino de una app abre la misma hoja, completada con el nombre que dio el resolutor del sistema para esa dirección.
- **Escapes.** El árbol de cgroups pertenece a root, así que una app no puede moverse sola. Un proceso que le pide al gestor de `systemd` del usuario una unidad nueva (`systemd-run --user`) termina bajo `user@<uid>.service` ; `lord-netperm` engancha los mismos programas ahí y somete esos procesos a la unión más estricta de las apps restringidas en ejecución de ese usuario. Probado: todos los intentos de escape denegados, los otros usuarios sin afectar.
- **Costo:** alrededor de 0,5 µs por connect de UDP, TCP dentro del ruido, sin cambio en el rendimiento, porque nada corre por paquete.
- **El Monitor** muestra las conexiones de cada app (desde `sock_diag`) y los destinos que el firewall informó hace poco, los bytes, los destinos por país desde una base de datos local y un interruptor de Regla por app. Ninguna dirección sale de la máquina para consultas: sin DNS inverso, sin geolocalización en línea.

Restricciones más allá de la red. Además de las cuatro restricciones de red (sin internet, sin LAN, sin red, sin conexiones entrantes), el dueño puede restringirle a una app la cámara, el micrófono, la pantalla, el USB, los archivos, el portapapeles, el Bluetooth y las notificaciones. Cada una se aplica donde se entrega el recurso, se verifica contra el scope de la app y nunca contra un nombre que la app se da a sí misma, y gana sobre cualquier permiso otorgado en `permisos.json` .

Dónde se aplica cada restricción

 PROPIO

Broker en lord-server

Apps .lpk y Contraseñas: las ocho restricciones, rechazadas sin aviso



 PROPIO

Lanzador de apps

Apps de Linux, sin acceso a tus archivos: espacios de nombres de usuario, montaje y PID propios, un home en tmpfs y Landlock, desde el próximo inicio



 PROPIO

Servidor de notificaciones (notifyd)

Notificaciones de apps de Linux: la identidad del remitente la da el bus; si algo falla, se descarta



 PROPIO

Guardia de PipeWire (pwguard)

Micrófono y cámara por PipeWire: identidad tomada de las credenciales del socket, en vivo



 KERNEL

Programas eBPF de lord-netperm en el kernel

Ganchos de red; cgroup/dev para cámara, USB, captura ALSA y /dev/fb; BPF LSM socket_create para sockets Bluetooth

 Propio

 Kernel

Cada restricción se revisa donde se entrega el recurso, contra el scope de la app, que es de root, nunca contra un nombre que la app se da a sí misma.

Restricción	Apps .lpk y Contraseñas	Apps incluidas ("LordOS")	Apps de Linux desde el lanzador de apps
cámara	broker (<code>brokercam.c</code>)	<code>js_media.c</code>	<code>cgroup/dev</code> sobre los nodos V4L2, y la guardia de PipeWire
micrófono	broker	<code>js_media.c</code>	la guardia de PipeWire, y <code>cgroup/dev</code> sobre los nodos de captura ALSA
pantalla	broker (sin API de captura en modo app)	<code>js_media.c</code>	por diseño: sin protocolo de captura, sin servidor X, sin portal de captura de pantalla; <code>/dev/fb*</code> denegado
USB	broker (la llave de seguridad de Contraseñas)	no se ofrece	<code>cgroup/dev</code> : nodos USB directos, hidraw y serie USB
archivos	broker (selector de archivos, guardados de la cámara)	no se ofrece	el lanzador: espacios de nombres propios, un home en tmpfs y Landlock, desde el próximo inicio
portapapeles	broker	no se ofrece	no disponible
Bluetooth	broker (sin API de Bluetooth en modo app)	no se ofrece	programa BPF LSM sobre <code>socket_create</code> : sin sockets <code>AF_BLUETOOTH</code>
notificaciones	el permiso <code>notifications</code>	<code>sdk/notify.js</code>	<code>notifyd</code> , por la identidad verificada de quien llama

Para las apps aisladas, el broker (`sec/restrict.c`) es el único punto de aplicación: un pedido restringido se rechaza sin aviso, y restringir una app en marcha le quita la cámara y el micrófono al instante (`tests/broker_restrict_test.c` cubre cada restricción, con y sin ella). La fila "LordOS" es el escritorio mismo (Archivos, copiar y pegar, Discos, los ajustes de Bluetooth), así que no ofrece archivos, USB, portapapeles ni Bluetooth; en su lugar se restringen apps individuales. Para las apps de Linux:

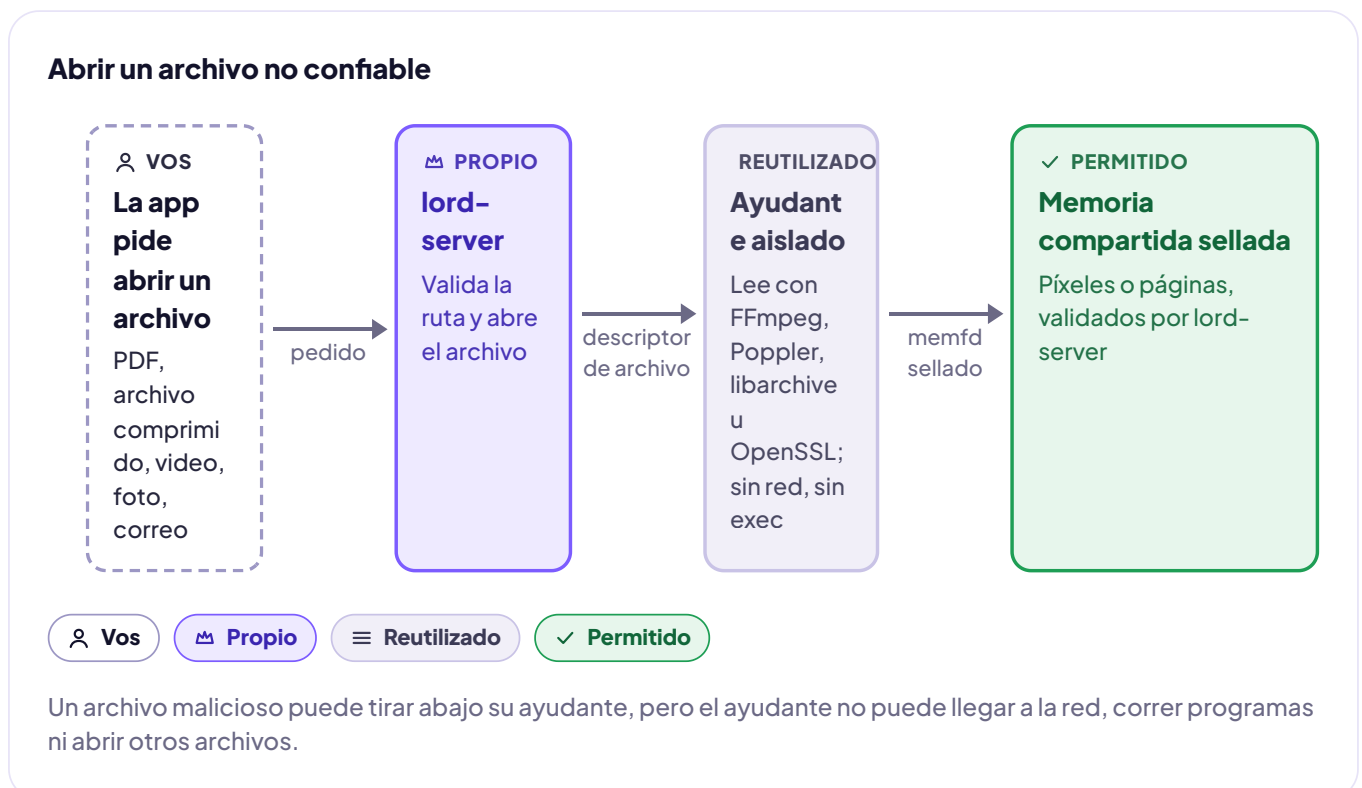
- **Dispositivos.** En cgroup v2 el controlador de dispositivos es un programa BPF. El de `lord-netperm` rechaza `open()` y `mknod()` de las clases de dispositivo que la app tiene restringidas (V4L2, USB directo, hidraw, serie USB, nodos de captura ALSA, `/dev/fb*`) y permite todo lo demás. Se aplica en el próximo `open` ; un descriptor ya abierto sigue sirviendo.
- **Sockets Bluetooth.** El gancho de sockets inet nunca ve `AF_BLUETOOTH` , así que un programa LSM por cgroup sobre `socket_create` los rechaza.
- **Archivos.** Justo antes de `execve` , el hijo del lanzador entra en sus propios espacios de nombres de usuario, montaje y PID, sin privilegios. La carpeta personal es un tmpfs vacío que tiene solo las carpetas propias de la app (derivadas de su id de escritorio, nunca de campos que controla un archivo `.desktop`) y el archivo con el que se abrió; la carpeta de runtime tiene solo los sockets de Wayland y de sonido, así que el bus de sesión y el systemd del usuario quedan fuera de alcance; después Landlock limita el resto. Se aplica desde el próximo inicio. Las apps GTK, Firefox y Chromium corren sin sus integraciones de D-Bus.

- **Notificaciones.** `notifyd` le pide al bus el PID de quien llama (solo cuenta una respuesta que mandó el propio bus), encuentra su scope de app y descarta la notificación de una app restringida; un error o una respuesta tardía también la descartan (falla cerrado).
- **Micrófono y cámara por PipeWire.** La guardia de PipeWire (`sec/pwguard.c`), el cliente de PipeWire propio de `lord-server` , identifica a cada cliente por el PID que PipeWire tomó de las credenciales del socket y, para una app restringida, oculta los nodos de captura y destruye sus streams de captura en menos de un segundo, incluida una grabación ya en curso. Mientras corre una app restringida, un cliente cuya identidad no se puede establecer se trata como restringido.

Límites. El modo "preguntar" está diseñado pero hoy se comporta como silencioso. ICMP se bloquea solo con "sin red"; las conexiones a un puerto ya abierto no se verifican. Las restricciones para apps de Linux no cubren:

- BlueZ por D-Bus: una app con el Bluetooth restringido todavía puede pedirle a `bluetoothd` , por el bus del sistema, que busque, empareje y conecte;
- el bus del sistema, que sigue al alcance incluso con "sin acceso a tus archivos";
- el portapapeles;
- las apps iniciadas fuera del lanzador de apps, que no están en ningún scope de app;
- los efectos secundarios de la unión de `user@` : mientras corre una app restringida, los servicios propios del usuario quedan sujetos a la misma unión (el audio de llamadas de unos auriculares, que PipeWire abre en cada llamada, se corta durante ese tiempo);
- una ventana de pocos milisegundos en la que un cliente nuevo de PipeWire de una app restringida tiene todos los permisos, o en la que un stream de captura puede recibir audio antes de ser destruido.

6.7 Ayudantes que procesan datos no confiables



`lord-doc` (PDF, archivos comprimidos, fuentes, imágenes del escáner), `lord-media` (video, audio, fotos, cámara), `lord-maild` (correo, calendarios, contactos) y `lord-vault` (secretos) comparten una caja de arena endurecida, `sandbox_apply_helper()` :

- sin `socket` , `socketpair` , `connect` , `bind` , `listen` , `accept` ; sin `execve` ; `clone` solo para hilos; `seccomp` en todos los hilos, y si `seccomp` no se puede aplicar el ayudante termina;
- archivos solo a través de descriptores que pasa `lord-server` ; `ioctl`s solo por clase (DRM, V4L2, DMA-BUF) donde hace falta;
- resultados devueltos en memoria compartida sellada, validada por `lord-server` .

`lord-media` ni siquiera recibe el socket de PipeWire; el audio pasa por un anillo compartido cuyo stream es de `lord-server` . `lord-maild` no puede crear sockets: se lo pide a `lord-server` , que conecta solo con los servidores del archivo de cuentas (y, después de que el usuario pide cargar imágenes, solo con direcciones públicas en los puertos 80/443 durante diez minutos) y le entrega el socket conectado. `build/sandbox-test helper` pasa 17 de 17 verificaciones.

6.8 Firma de paquetes y niveles de confianza

Los paquetes `.lpk` llevan un manifiesto con el SHA-256 de cada archivo y una firma Ed25519 sobre él. SHA-256 y Ed25519 son código propio de LordOS (Ed25519 siguiendo la aritmética de TweetNaCl), probado contra los vectores del NIST y del RFC 8032 (55 de 55). La verificación ocurre en C al instalar y al cargar. La Tienda muestra el nivel de confianza: **LordOS** (clave oficial, compilada dentro del binario para que ningún archivo pueda agregar otra), **Autor verificado** (listado en el catálogo firmado), **Firmado por <nombre>**, o **Sin firmar** con una advertencia. Las actualizaciones tienen que estar firmadas con la misma clave. Un catálogo firmado lleva una lista de revocación. Si editás una app instalada, LordOS la marca "Modificada por vos", puede restaurar la original o compartir tu versión firmada con tu propia clave, con crédito al original.

Estado: en la 0.8.0 Beta los paquetes oficiales todavía se firman con una clave de desarrollo, hasta que la reemplace la clave final. Todavía no está construido un repositorio firmado de apps de LordOS para la Tienda.

6.9 Firma de releases

Cada archivo de release (el `.deb` , la ISO y la imagen para Raspberry Pi) aparece con su SHA-256 en `SHA256SUMS` , y `SHA256SUMS` está firmado con la clave de releases de LordOS (Ed25519). La clave pública es `tools/release/lordos-release.pub.pem` , también embebida en el instalador para Debian. Su huella (SHA-256 de la clave pública en DER) es:

```
c7155b8e6bbbb180f907381994d06af76d333016cd6f3facf67cadd053ea108e
```

La clave privada vive solo en la máquina del mantenedor, nunca en el repositorio ni en una máquina de compilación. Cada release trae además una lista de materiales de software (SBOM) SPDX 2.3 por arquitectura, listada en `SHA256SUMS` y por lo tanto cubierta por la firma, y sus notas salen de `CHANGELOG.md` , escrito en el formato Keep a Changelog. La §11 muestra cómo verificar una descarga.

6.10 La bóveda

`lord-vault` guarda las contraseñas y el llavero de la sesión en archivos `.lvault` : una clave de bóveda aleatoria de 32 bytes, envuelta una vez por cada método de desbloqueo (contraseña con Argon2id con un costo calibrado

por máquina, FIDO2 `hmac-secret` con el cliente CTAP2 propio de LordOS, archivo llave, contraseña más archivo llave, clave de recuperación); contenido sellado con XChaCha20-Poly1305 con todo el encabezado como datos asociados; adjuntos sellados por separado; guardado atómico con copia de respaldo. Se detecta cada byte invertido y cada truncamiento (1.165 de 1.165 en la prueba). Los secretos viven en un área de memoria con `mlock`, excluida de los volcados de memoria y borrada al liberarse, en un proceso que no se puede volcar. La ventana de Contraseñas corre en su propio proceso aislado, así los valores revelados nunca entran al JavaScript del escritorio. El llavero se abre al iniciar sesión con la contraseña que verificó `lord-sessiond`, y atiende a las apps de Linux mediante la implementación propia de LordOS del Secret Service. Las llaves de seguridad de hardware reales todavía no se probaron.

6.11 Ventanas protegidas

Una ventana marcada como protegida (Contraseñas lo está) se pinta en negro en todos los caminos que copian píxeles hacia afuera: capturas, grabaciones, miniaturas, el escritorio remoto, y los menús y la pantalla completa de Wayland. El monitor real la muestra normalmente. Las apps de Linux, para empezar, no tienen ningún protocolo de captura de pantalla.

6.12 Divulgación y estándares

Las vulnerabilidades se informan como describe `SECURITY.md`, con un contacto legible por máquinas en `https://lordos.org/.well-known/security.txt` (RFC 9116). Los procesos de LordOS están **alineados con, no certificados en**, ISO/IEC 29147 e ISO/IEC 30111 (divulgación y manejo de vulnerabilidades), ISO/IEC 5230 (cumplimiento de licencias de OpenChain: el inventario en `THIRD_PARTY.md`, los avisos de licencia y el SBOM) e ISO/IEC 18974 (garantía de seguridad de OpenChain: el SBOM más las actualizaciones de seguridad de Debian para los paquetes de los que depende LordOS). lordos.org publica las páginas legales y de confianza (política de seguridad, firma de código, licencias, privacidad, ciclo de soporte, estándares) y dice lo que todavía falta: sin autocertificación formal, sin cruce automático del SBOM con bases de vulnerabilidades, sin SBOM de las imágenes completas y sin auditoría de terceros.

Archivos fuente

- [server/src/sec/sandbox.c](#), [server/src/sec/brokersrv.c](#), [server/src/sec/perms.c](#), [server/src/sec/cgroup.c](#)
- [server/src/netperm/np.bpf.c](#), [server/src/netperm/netperm.c](#), [server/src/fw/rules.c](#)
- [server/src/sec/restrict.c](#), [server/src/sec/pwguard.c](#), [server/src/hw/notifyd.c](#), [server/src/apps.c](#), [server/tests/broker_restrict_test.c](#)
- [server/src/sec/sign.c](#), [server/src/pkg/crypto.c](#), [tools/release/sign-release.sh](#), [tools/release/make-sbom.sh](#)
- [server/src/vault/vault.c](#), [server/src/vault/vfile.c](#)
- [server/src/media/capture.c](#)
- [SECURITY.md](#), [docs/COMPLIANCE.md](#), [CHANGELOG.md](#)

7. Rendimiento

Todas las cifras de abajo las midió el proyecto y están documentadas en los archivos citados. La mayoría vienen de máquinas virtuales; las mediciones en hardware real se indican como tales.

7.1 Memoria y reposo

Medición	Valor	Dónde
Sistema completo, ISO live, en reposo después de un minuto (4 GB, 4 núcleos)	465 MB usados, CPU 99 % ociosa	VM117, docs/ISO.md
Sistema completo, instalado desde la ISO, en reposo	487 MB usados, CPU 99 % ociosa	VM120, docs/ISO.md
Arranque, ISO live con UEFI	16,3 s (4,9 s kernel + 11,4 s espacio de usuario)	VM117, docs/ISO.md
<code>lord-server</code> sin placa 3D (Mesa sin cargar)	unos 32 MB de RSS	docs/GPU.md
Una sesión de usuario (<code>lord-server</code>), pruebas de escritorio remoto	unos 50 MB; pantalla de inicio 41 MB	VM118, docs/REMOTE.md
Una app <code>.lpk</code> aislada	13,7 MB de RSS	VM122, docs/PRIVACY.md

7.2 Dibujo

- Arrastrar una ventana en una pantalla de 1280×800 por el camino de CPU: unos 6,7–6,9 ms de trabajo por cuadro, alrededor de 120 cuadros por segundo (VM118, [docs/DISPLAYS.md](#)).
- La misma ventana en una pantalla de 2560×1600 al 150 %: unos 5,1 ms por cuadro.
- Video: pintar directamente desde el cuadro decodificado cuando solo cambió un video opaco bajó `lord-server` del 42 % al 13 % de un núcleo para un video 1080p en ventana, y del 92 % al 15 % a pantalla completa ([docs/ROADMAP.md](#) , multimedia).
- El Monitor cerrado no cuesta nada: su hilo de muestreo no existe hasta que se abre ([docs/MONITOR.md](#)).

7.3 GPU

En las VM solo había un renderizador por software. Con él, el camino de CPU es más barato que la composición "por GPU", y por eso LordOS usa el camino de CPU cuando el único renderizador es por software. El beneficio real del camino por GPU (composición y presentación casi gratis, pantalla completa sin copias) se espera en placas reales, pero **todavía no se midió en hardware AMD ni Nvidia real**.

7.4 La Raspberry Pi 5

La Raspberry Pi 5 es la única placa soportada. Resultados en una placa real (16 GB, tarjeta SD, sin monitor), de [docs/RASPBERRY.md](#) :

Medición	Valor
Compilar todos los binarios de LordOS en la placa	1 min 02 s, 0 advertencias
<code>make -C server test</code>	23 pasaron, 0 fallaron, 0 omitidas
Copias y máscaras NEON vs. C puro	9,90 → 4,33 ms y 11,09 → 5,49 ms por pantalla (2,0–2,3×), idénticas bit a bit
En reposo en la pantalla de inicio, sin monitor	<code>lord-server</code> 36 MB de RSS, 0,6 % de CPU; sistema 450 MB de 16 GB; 38–43 °C, sin limitación térmica
Armar la imagen SD en la placa	8 min 18 s; 503 MB comprimida
Compilar el kernel de LordOS en la placa	unos 50 min por variante

El kernel de LordOS. El kernel propio de Raspberry Pi no compila Landlock. Sin él, la prueba hostil del ayudante de documentos falla 5 de sus 25 verificaciones de sistema de archivos y la prueba de la caja de arena no puede correr. LordOS compila el kernel de Raspberry Pi desde su propio código fuente y configuración predeterminada, cambiando solo tres opciones (Landlock activado, la lista de LSM, el sufijo de versión), y lo fija para que un kernel más nuevo del archivo no pueda reemplazarlo. La imagen también activa el cgroup de memoria, que Raspberry Pi apaga al arrancar.

Ventiladores. En la Pi 5 dentro de un gabinete de prueba, `lord-fand` mantuvo el ventilador apagado a 50–53 °C durante una carga de 4 hilos (el gabinete de aluminio funciona como disipador), lo llevó al 55 % (4.500 RPM) y a velocidad máxima (7.400 RPM) con curvas de prueba, y restauró la configuración propia del kernel después de `SIGTERM` y de `kill -9 ( docs/FANS.md )`.

Todavía no verificado en la placa: todo lo que requiere un monitor conectado (configuración de modos, composición por GPU en pantalla, cuadros por segundo), la sesión de escritorio y arrancar la imagen generada.

7.5 PC modernas

Todavía no hay mediciones en PC reales en el repositorio. La ISO se prueba en máquinas virtuales (UEFI y BIOS); está planeado probarla en una máquina AMD real desde el USB live.

Archivos fuente

- [docs/ISO.md](#), [docs/RASPBERRY.md](#), [docs/FANS.md](#), [docs/DISPLAYS.md](#)
- [tools/rpi/build-kernel.sh](#), [tools/rpi/build-img.sh](#)
- [server/src/canvas.c](#)

8. Experiencia visual

- **Una guía de diseño.** Las apps siguen una guía común (`docs/DESIGN.md`): una grilla de espaciado de 8 px, una escala tipográfica chica, alturas de controles consistentes y una sola acción con el color de acento por vista.

El SDK exporta los tokens (`spacing` , `typeScale` , `controlSize`), y cada app pasó por rondas de pulido documentadas con capturas de antes y después en claro, oscuro, de derecha a izquierda y en ventanas chicas.

- **Temas.** Un tema es un escritorio completo escrito en React: barra de menú, Dock, lanzador de apps, Centro de control y marco de ventana, más sus tokens. LordOS trae un tema predeterminado con barra de menú y Dock, y un tema Clásico con barra de tareas y menú de inicio, que hoy es una plantilla y pasa a ser una segunda apariencia completa antes de la 1.0 (\$10.3). Los temas cambian en vivo; la configuración se conserva.
- **Claro y oscuro, acentos.** Los dos modos y el color de acento se aplican en vivo a las apps nativas, a las apps aisladas y, a través del portal y de GSettings, a las apps GTK.
- **De derecha a izquierda.** Toda la interfaz se espeja para el árabe, el hebreo y escrituras similares, incluidas las apps aisladas, que siguen en vivo el idioma de la sesión.
- **Lumo.** La mascota de LordOS es Lumo, un pequeño zorro celestial, que aparece en pantallas de bienvenida, estados vacíos y material de marca, con un conjunto fijo de poses definido en la guía de marca.
- **Accesibilidad hoy:** una opción "Reducir movimiento" que apaga las animaciones decorativas, atajos de teclado en todo el sistema y pantallas escalables hasta el 200 %. Un lector de pantalla, zoom y alto contraste están en la hoja de ruta y **todavía no están contruidos**.

Archivos fuente

- [docs/DESIGN.md](#), [docs/LORDOS-BRAND.md](#)
- [themes](#), [sdk/controls.jsx](#), [sdk/i18n.js](#)

9. Extensibilidad

- **El SDK** (`@lordos/sdk`) es lo único que importan las apps: ventanas, configuración, tokens del tema, controles, archivos, multimedia, notificaciones, llavero, impresión y más. Las apps no importan nada de los temas, así cualquier app funciona en cualquier tema.
- **Escribir una app** es una carpeta con `manifest.json` e `index.jsx` . `node tools/lpk.mjs <folder>` la empaqueta sin React ni el SDK, agrega el ícono y las traducciones, y la firma si se le da una clave; `keygen` , `sign` , `verify` y `catalog` cubren el resto. Las apps pueden incluir módulos de WebAssembly y dibujar píxeles crudos con `<Image pixels>` .
- **Los temas** siguen un contrato fijo (`Desktop` , `MenuBar` , `Dock` , `Launcher` , `ControlCenter` , `WindowFrame` , tokens), documentado en `themes/README.md` , con el tema Clásico como plantilla.
- **La interoperabilidad con Linux** pasa por estándares: Wayland, el portal de configuración, el Secret Service, el servicio de notificaciones, las apps predeterminadas de `mimeapps.list` y los archivos `.desktop` .
- **Automatización:** hoy no hay ninguna API de scripting ni de automatización más allá del SDK. Las integraciones con asistentes de IA (por ejemplo a través del Model Context Protocol) **no existen en LordOS**; son solo una posible dirección futura y tendrían que respetar la misma regla de que el JavaScript no puede ejecutar comandos.

Archivos fuente

- [sdk](#), [apps/README.md](#), [themes/README.md](#)
- [docs/NATIVE-APPS.md](#), [docs/WASM.md](#), [examples/apps/fractal](#)

10. Hoja de ruta y limitaciones

10.1 Dónde estamos

La 0.8.0 Beta es el primer lanzamiento público; la 0.7.0 fue una compilación interna y nunca se publicó. Los hitos M1 a M9 están completos: el servidor gráfico, el motor de interfaz, el SDK y los temas, la terminal, los idiomas y las escrituras complejas, la compatibilidad con apps de Linux, los paquetes y los servicios del sistema, las sesiones, el aislamiento y el firewall, y las suites de apps de M9 (documentos e impresión, multimedia, apps del sistema, contraseñas, configuración de hardware, correo y calendario). Antes del lanzamiento, las restricciones del firewall que no son de red pasaron a aplicarse para todos los tipos de app, las reglas por dominio tuvieron interfaz en Ajustes y en el Monitor, el arranque sumó una pantalla de carga y un tema de GRUB de LordOS, y los releases sumaron un SBOM firmado y un registro de cambios. La ISO final se armó y se probó en máquinas virtuales el 18 de octubre de 2026.

10.2 Limitaciones conocidas

- Hardware real: no se probaron GPU AMD y Nvidia, monitores reales, notebooks (batería, brillo, tapa), controladores Bluetooth, impresoras y escáneres, llaves FIDO2 ni proveedores de correo reales.
- La Raspberry Pi 5 es la única placa soportada, con la compilación de LordOS de su kernel con Landlock; no se probó con un monitor, y su imagen generada todavía no se arrancó.
- Las apps de Linux no están aisladas de forma predeterminada; todavía no hay Xwayland.
- Firewall: sin modo "preguntar"; las reglas por dominio no ven a las apps que resuelven nombres con su propio DNS cifrado.
- Las restricciones para apps de Linux no cubren BlueZ por D-Bus, el bus del sistema, el portapapeles ni las apps iniciadas fuera del lanzador de apps; la unión de `user@` también sujeta a los servicios propios del usuario mientras corre una app restringida; un cliente nuevo de PipeWire tiene una ventana de pocos milisegundos antes de que la guardia lo restrinja (§6.6).
- La clave de firma de `.1pk` todavía es una clave de desarrollo, y todavía no hay un repositorio firmado de apps de LordOS.
- Permisos: no hay "permitir una vez"; no se muestra el uso de cámara y micrófono por parte de apps de Linux.
- Hay que desactivar Secure Boot; la firma con un shim está pendiente.
- Las descargas de paquetes usan HTTP (firmadas y con hash).
- El escritorio remoto no tiene audio.
- La accesibilidad (lector de pantalla, zoom, alto contraste) no está construida.

10.3 Lo que viene

Lo próximo, una imagen para Raspberry Pi 5 arrancada y probada en la placa real. Antes de la 1.0, según lo planeado en `docs/ROADMAP.md` :

- **Xwayland**, iniciado a pedido, con el compositor como gestor de ventanas X y las apps X11 con la misma identidad por app, el mismo firewall y las mismas restricciones que las apps Wayland;
- **el tema Clásico como segunda apariencia completa**: barra de tareas, menú de inicio con búsqueda, bandeja, ajustes rápidos, panel de notificaciones y acomodo de ventanas, con las mismas apps con otra piel, hecho solo con diseño y recursos propios de LordOS.

Después de la 1.0: una app de virtualización de escritorio construida sobre KVM y QEMU; un contenedor Android con LineageOS y apps desde la Tienda de LordOS; el modo "preguntar" del firewall; accesibilidad; y un video de demostración completo.

Archivos fuente

- [docs/ROADMAP.md](#), [CHANGELOG.md](#)

11. Cómo verificar el proyecto por tu cuenta

11.1 Compilalo

En una máquina de desarrollo con Node.js, empaquetá los temas, el SDK y las apps:

```
npm install && npm run build
```

En Debian 13, compilá los binarios y corré las pruebas:

```
sudo apt-get install -y --no-install-recommends build-essential
make -C server -j4
make -C server test          # every self-contained suite
make -C server test-full    # adds the helpers, the sandbox and the HarfBuzz comparison
```

make test falla si falla cualquier suite; una suite cuya herramienta o datos no están instalados informa "skipped" en lugar de pasar en silencio. Las pruebas de la caja de arena necesitan un kernel con Landlock. La ISO y la imagen para Raspberry Pi se arman como se describe en [docs/ISO.md](#) y [docs/RASPBERRY.md](#).

11.2 Qué cubren las pruebas

- **Terminal y texto:** el emulador VT, el shaping OpenType contra HarfBuzz, el bidi Unicode contra el archivo de prueba oficial, los métodos de entrada y la separación de palabras en tailandés contra libthai.
- **Seguridad:** la caja de arena (app, red, ayudante), la firma (vectores del NIST y del RFC 8032), el estado de los paquetes, las reglas de rutas de archivos comprimidos contra archivos maliciosos, las protecciones de **lord.fs**, las reglas del firewall (con 20.000 mutaciones de JSON y 30.000 operaciones aleatorias contra una referencia lineal), el lector DNS (200.000 paquetes aleatorios), el compilador de mapas del kernel (480.000 conexiones) y la prueba de integración con el kernel (18 de 18 en una VM).
- **Lectores:** correo, MIME, sanitizador de HTML, iCalendar, vCard y XML (30.953 verificaciones con fuzzing por mutación), el formato de la bóveda, los importadores, QR, CBOR, la serialización de D-Bus, el lector de la base de países, el journal, TZif e IPP.
- **Motor y hardware:** layout y pintado, NEON vs. C, el driver de touchpad con gestos grabados, ventiladores sobre árboles sysfs falsos, la disposición de pantallas, el sonido y la suite de la especificación de WebAssembly.

En la compilación de la ISO final: `make -C server test` informó 23 pasaron, 0 fallaron, 1 omitida (faltaban los datos de bidi).

11.3 Verificá un release

Descargá `SHA256SUMS` , `SHA256SUMS.sig` , los archivos del release y la clave pública (`tools/release/lordos-release.pub.pem`). Comprobá que la huella de la clave sea

`c7155b8e6bbbb180f907381994d06af76d333016cd6f3facf67cadd053ea108e` y después, con OpenSSL 3:

```
base64 -d SHA256SUMS.sig > SHA256SUMS.sig.bin
openssl pkeyutl -verify -pubin -inkey lordos-release.pub.pem -rawin -in SHA256SUMS -sigfile SHA256SUMS.sig.bin
sha256sum -c --ignore-missing SHA256SUMS
```

El SBOM de cada arquitectura (`lordos_<version>_<arch>.spdx.json`) también está en `SHA256SUMS` , así que la misma verificación lo cubre.

11.4 Verificá un sistema en marcha

- `systemctl status lord-netperm lord-fand lordos` muestra los servicios; la prueba de la ISO confirmó los siete programas de red enganchados. Si el kernel rechaza el programa de dispositivos o el LSM, `lord-netperm` informa `"devices":false` o `"bluetooth":false` .
- `cat /proc/<pid>/status | grep -E 'Seccomp|NoNewPrivs'` sobre un proceso `lord-server --app` muestra la caja de arena.
- `systemd-cgls` muestra el árbol `lordos-session-<uid>` , de root, con un scope por app.
- Escritorio remoto: `ss -ltn` no muestra nada en el puerto 5673 hasta que lo activás en Ajustes → Compartir.

12. Licencia y créditos

El código de LordOS se publica bajo la **licencia MIT**. Los componentes de terceros conservan sus propias licencias, listadas en `THIRD_PARTY.md` : QuickJS y stb (incluidos como código fuente), Monocypher (CC0/BSD-2), la lista de palabras de la EFF (CC BY 3.0 US), la lista de contraseñas comunes (MIT), los datos de países de DB-IP Lite (CC BY 4.0), Natural Earth (dominio público) y las fuentes Inter, JetBrains Mono y Plus Jakarta Sans (OFL). Cada release trae un SBOM SPDX de lo que el paquete incluye y de lo que depende (`docs/COMPLIANCE.md`).

LordOS lo crea el grupo **iPERFEX**: **Pablo Miranda** y **Federico Pereira**.

13. Glosario

- **Broker**: el único socket por el que una app aislada le pide a `lord-server` cualquier cosa que toque las cosas del usuario; cada pedido se verifica en C.
- **cgroup (v2)**: un mecanismo del kernel que agrupa procesos para contabilizar y limitar recursos. LordOS le da a cada app su propio grupo, un "scope", y lo usa para identificar la app en el firewall.
- **Compositor**: el programa que combina las ventanas de todas las apps en la imagen final de la pantalla.
- **DRM/KMS**: las interfaces del kernel para los dispositivos gráficos y para configurar los modos de pantalla.

- **eBPF**: programas pequeños y verificados que corren dentro del kernel; LordOS los engancha a cgroups para decidir sobre las conexiones de red, la apertura de dispositivos y los sockets Bluetooth.
- **evdev**: la interfaz del kernel para los dispositivos de entrada.
- **LSM (BPF LSM)**: los ganchos de seguridad del kernel; con el LSM `bpf`, un programa chico enganchado por cgroup puede rechazar una operación, como crear un socket Bluetooth.
- **Espacio de nombres (namespace)**: una función del kernel que le da a un proceso su propia vista de algo (usuarios, montajes, ids de proceso); LordOS usa espacios sin privilegios para las apps de Linux a las que se les restringen los archivos del usuario.
- **Ayudante**: un proceso chico de LordOS que maneja datos no confiables o secretos en una caja de arena más estricta (`lord-doc`, `lord-media`, `lord-maild`, `lord-vault`).
- **Landlock**: una función del kernel que permite que un proceso sin privilegios restrinja qué archivos y puertos de red puede usar él, y sus hijos.
- `.lpk`: el formato de paquete de LordOS para apps nativas instalables.
- `.lvault`: el formato de archivo de bóveda cifrada de LordOS.
- **Guardia de PipeWire**: el cliente de PipeWire propio de `lord-server` que les oculta los dispositivos de captura a las apps de Linux restringidas y frena sus streams de captura.
- **Powerbox**: un diálogo de archivos dibujado por el sistema que le da a la app solo el archivo que eligió el usuario.
- **QuickJS**: un motor de JavaScript chico escrito en C.
- **SBOM**: una lista de materiales de software, el listado legible por máquina de los componentes que contiene un release, en formato SPDX.
- **Scope**: un cgroup para una app, como `app-lordos-lpk-<id>.scope`.
- **seccomp-bpf**: un filtro del kernel que restringe qué llamadas al sistema puede hacer un proceso.
- **Secret Service**: el estándar de D-Bus de freedesktop que usan las apps de Linux para guardar secretos.
- **Wayland**: el protocolo entre las apps gráficas y el compositor.

14. Dónde leer más

Capítulo	Documentos en docs/
1. Visión	ARCHITECTURE.md, LORDOS-BRAND.md
2. Arquitectura	ARCHITECTURE.md, NETWORK.md, FANS.md, SETTINGS.md, SYSTEM.md
3. Compositor	GPU.md, DISPLAYS.md, WINDOWS.md, TRACKPAD.md, WAYLAND.md, SESSIONS.md, REMOTE.md
4. Motor de interfaz	ARCHITECTURE.md, LANGUAGES.md, TERMINAL.md
5. Aplicaciones	NATIVE-APPS.md, PACKAGES.md, WASM.md, FILES.md, MEDIA.md, DOCUMENTS.md, MAIL.md, ISO.md, INSTALL-DEBIAN.md
6. Seguridad	PRIVACY.md, FIREWALL.md, MONITOR.md, SIGNING.md, PASSWORDS.md, MEDIA.md, REMOTE.md, COMPLIANCE.md, y SECURITY.md
7. Rendimiento	ISO.md, RASPBERRY.md, FANS.md, DISPLAYS.md, GPU.md
8. Experiencia visual	DESIGN.md, LORDOS-BRAND.md, LANGUAGES.md, DOCK.md
9. Extensibilidad	NATIVE-APPS.md, WASM.md, y apps/README.md , themes/README.md
10. Hoja de ruta	ROADMAP.md, RELEASING.md y CHANGELOG.md