

LordOS: a private, fast GNU/Linux desktop built on Debian and Wayland

Technical whitepaper: design, security model and current state

Version **0.8.0 Beta** Authors: **Pablo Miranda, Federico Pereira** · iPERFEX Date: **October 18, 2026**

Abstract

LordOS is a free GNU/Linux distribution based on Debian 13 that ships a desktop environment of its own, built from its fundamental components on top of Debian and Wayland. Instead of GNOME or KDE, it has its own display server and Wayland compositor written in C (`lord-server`), its own user-interface engine (React running on the QuickJS JavaScript engine, with layout, painting and text in native C code), its own native apps, and its own security services: an app sandbox launcher, a permission broker, a per-app permission store, a kernel-enforced per-app firewall, package signing and an encrypted password vault. It reuses what is mature and widely reviewed: the Linux kernel, Debian and its packages, Mesa, PipeWire, the Wayland protocol, and a handful of libraries (FFmpeg, Poppler, libarchive, OpenSSL, Monocypher) that are confined to small sandboxed helper processes.

This paper describes the design and the current state of version **0.8.0 Beta** (October 2026), the first public release, for a technical reader who wants to check the project rather than take it on trust. Every claim refers to a file in the public repository. Numbers are the ones measured in the project's test logs, and are labelled with where they were measured. Things that are designed but not built are marked as such.

1. Vision

1.1 The problem

A modern Linux desktop is assembled from many independent projects: a display server or compositor, a toolkit, a session manager, a settings daemon, a keyring, a network manager, a firewall front end, and the apps. Each is good on its own terms, but the seams between them are where things get slow, inconsistent or insecure. Three seams in particular motivated LordOS:

- **Weight.** A desktop that draws its shell with a web engine, or loads several toolkits at once, needs hundreds of megabytes before the user opens anything. LordOS wants to be comfortable on modest machines and on a Raspberry Pi 5.
- **Trust in apps.** On a traditional Linux desktop, every program the user runs can read the whole home folder, record the microphone, open any network connection and read the clipboard. Sandboxing exists, but it is usually an add-on that the desktop itself doesn't understand.
- **Visibility.** Users rarely see which app is talking to which server, and even more rarely can they say "this app may not use the internet" in a way the app can't get around.

1.2 Principles

The project follows a short list of rules, written down in `AGENTS.md` and `docs/ARCHITECTURE.md` :

1. **Own the parts the user touches; reuse the foundations.** The display server, the compositor, the UI engine, the apps and the security services are LordOS's code. The kernel, Debian's packages, Mesa, PipeWire and the Wayland protocol are reused.
2. **A strict dependency policy.** X11, wlroots, libdrm, libinput, WebKit, GTK and Qt are not used inside LordOS. Libraries that parse untrusted data are linked only by sandboxed helpers, never by the compositor.
3. **The interface can't run commands.** JavaScript in LordOS has no API to execute arbitrary programs. There are three narrow, argument-free exceptions (the Terminal's login shell, launching an installed `.desktop` app, and starting an installed `.lpk` app in its sandbox).
4. **Root only through closed doors.** Everything that needs root goes through a fixed table of validated actions or a small dedicated service with a closed protocol.
5. **The CPU path must always work.** GPU composition is an optimization, never a requirement.
6. **The compositor never waits.** Helpers, apps and services talk to `lord-server` over non-blocking sockets; a hung app hangs only itself.
7. **Verify on a real system.** Every milestone is checked on a virtual machine with real screenshots before it is marked done, and compatibility that wasn't tested isn't claimed.

Source files

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

2. Architecture

2.1 Layers

The layers of LordOS

YOU

Apps

Built-in apps and the theme · isolated .lpk apps · Linux apps (GTK, Qt, browsers)



OURS

LordOS UI engine

React on QuickJS, native layout, painting and text in C. Used by LordOS's own apps; Linux apps don't go through it



OURS

LordOS display server and Wayland compositor

lord-server, in C: KMS without libdrm, evdev without libinput, composition on CPU or GPU. Linux apps and isolated .lpk apps connect here as Wayland clients



REUSED

Debian 13 userland

dpkg and packages, Mesa (optional GPU), PipeWire, libwayland, libxkbcommon, systemd



KERNEL

Linux kernel

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

YOU

OURS

REUSED

KERNEL

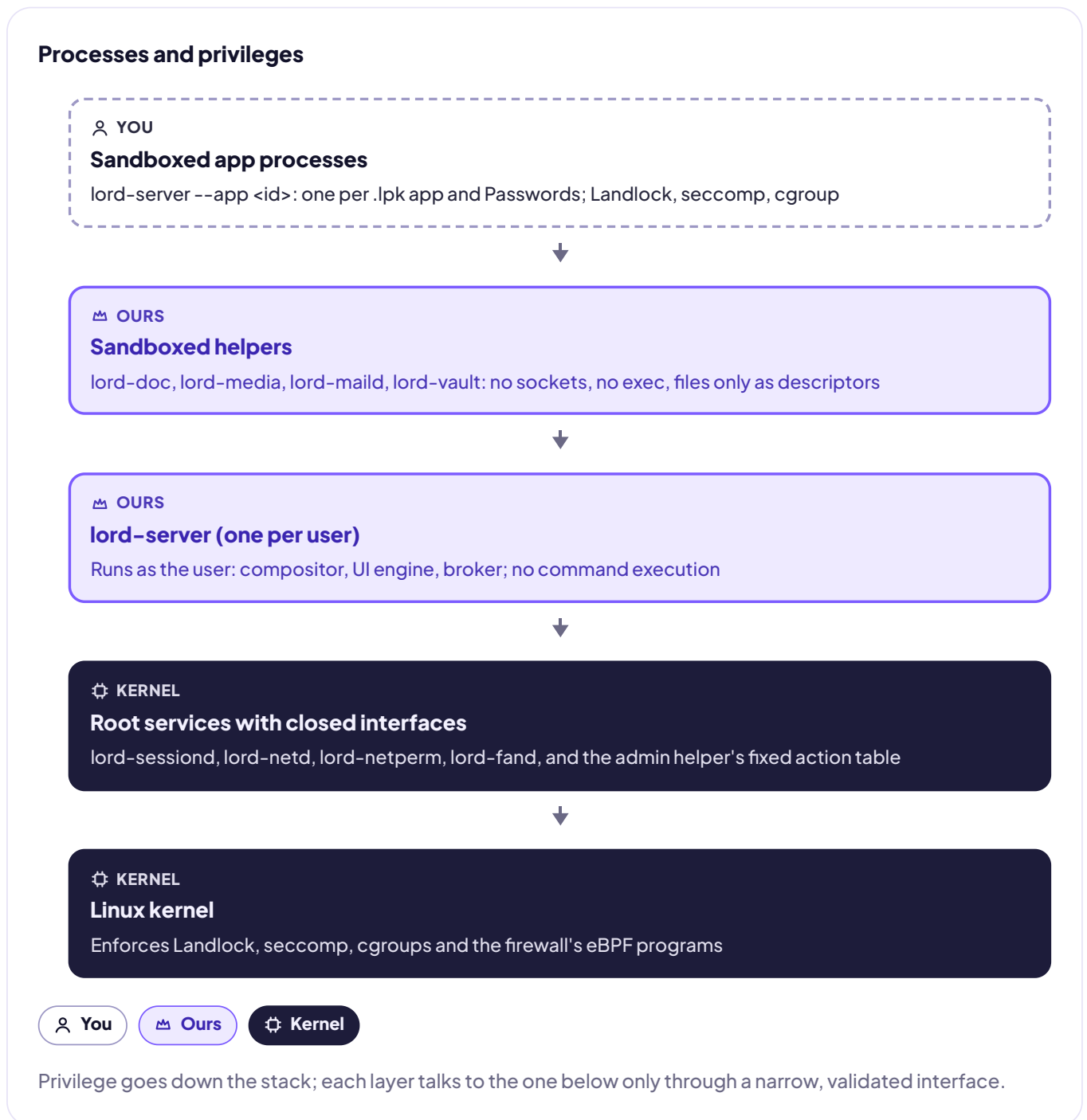
Violet layers are LordOS's own code; the lower layers are reused from Debian and Linux.

From the bottom up:

- **The Linux kernel** provides DRM/KMS for displays, evdev for input, and the security features LordOS builds on: Landlock, seccomp-bpf, cgroups v2 and eBPF. On PCs it is Debian's kernel; on the Raspberry Pi 5 it is LordOS's build of the Raspberry Pi kernel with Landlock enabled (§7.4).
- **Debian 13** provides the userland: `dpkg` and the package archive, systemd, Mesa, PipeWire, `libwayland`, `libxkbcommon`, CUPS, BlueZ and the rest.
- `lord-server` is the display server and Wayland compositor. It drives displays and input directly, composites, and serves the Wayland protocol to Linux apps.

- **The UI engine** lives inside `lord-server` : QuickJS runs the React interface, and native C code does layout, painting and text.
- **Apps** come in three kinds: built-in apps and the theme, which run in `lord-server` 's JavaScript context; installed `.lpk` apps, each in its own sandboxed process with its own engine instance; and Linux apps, which connect to the compositor as ordinary Wayland clients.

2.2 Processes and privileges



A running LordOS system has a small number of processes with clearly separated roles:

Process	Runs as	Role
<code>lord-sessiond</code>	root	login screen, sessions, lending the display and input to the visible session, remote desktop
<code>lord-netd</code>	root	the system network: DHCP, Wi-Fi, VPN; only the <code>net-*</code> actions
<code>lord-netperm</code>	root (limited capabilities)	per-app network rules, and the device and Bluetooth restrictions of Linux apps, enforced with cgroup and LSM eBPF programs
<code>lord-fand</code>	root (no capabilities)	fan control by temperature
<code>lord-server</code>	the user	one per session: compositor, UI engine, broker
<code>lord-server --app <id></code>	the user, sandboxed	one per running <code>.lpk</code> app (and Passwords)
<code>lord-doc</code> , <code>lord-media</code> , <code>lord-maild</code> , <code>lord-vault</code>	the user, sandboxed	helpers that process untrusted data or secrets
login screen	<code>lordos-greeter</code>	draws the login UI; never verifies passwords itself

Each root process has a narrow, documented interface. §6.3 describes them.

2.3 What is ours and what is reused

Ours (in this repository)	Reused
Display server and Wayland compositor (C)	Linux kernel
UI engine: native nodes, flexbox layout, rasterizer, text shaping, bidi, input methods	Debian 13 and its packages; <code>dpkg</code>
React renderer (host configuration for <code>react-reconciler</code>)	React 19.3.1, <code>react-reconciler</code> 0.34 (bundled in the theme)
Native apps and themes	QuickJS 2025-04-26 (vendored)
Sandbox, broker, permissions, signing, firewall service, vault	Mesa (optional), PipeWire, <code>libwayland</code> , <code>libxkbcommon</code>
Session manager, network service, fan service	systemd, systemd-resolved, BlueZ, CUPS, wpa_supplicant, OpenVPN, strongSwan's daemon
Debian package engine (OpenPGP, xz, HTTP, resolver)	FFmpeg, Poppler, libarchive, OpenSSL, Monocypher (in helpers only)
Touchpad driver, terminal emulator, WebAssembly interpreter	stb (fonts and images), fonts under the OFL

LordOS speaks to several Debian services over their own protocols with its own clients: D-Bus (BlueZ, power profiles, time and date), IPP (CUPS), the PulseAudio protocol (PipeWire), and the Secret Service. `libdbus`, `sd-bus`, `libcups`, `libpulse`, `libbpf` and `libapt` are not used.

2.4 System services

- **Network (`lord-netd`).** The network belongs to the system, not to a session. `lord-sessiond` starts `lord-netd` at boot with a fixed path and no arguments, so the machine has an address before anyone logs in. It runs only the closed `net-*` actions: its own DHCP client and netlink code, Wi-Fi through Debian's `wpa_supplicant` , WireGuard through LordOS's own netlink code, OpenVPN, IPsec/IKEv2 through strongSwan's `charon` driven over VICI with LordOS's own client, and optionally OpenConnect, PPTP and Fortinet-compatible tunnels. It talks only to the sessions' root helpers, over a socket that accepts only uid 0.
- **Fans (`lord-fand`).** A root service that reads the CPU temperature every 2 s and writes fan duty from a profile, from early boot. It drives only fans its own scan found: the Raspberry Pi 5 fan header and allow-listed motherboard controllers. Laptop, GPU and unknown fans are never written. On stop, or after a crash, it restores the original kernel settings. It runs no programs, has no network, no capabilities and no device nodes.
- **Hardware settings.** Displays (modes from KMS with EDID names, a 15-second "keep these settings?" revert, per-display persistence, rotation, Night Light), Bluetooth through BlueZ, battery and power profiles, idle and lid handling, date and time, notifications for Linux apps and default apps. Changes arrive as kernel uevents and D-Bus signals; nothing polls.

Source files

- [server/src/main.c](#)

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

3. The compositor

3.1 Displays without libdrm

`lord-server` sets display modes with the kernel's DRM ioctls directly (`kms.c`): it enumerates connectors, CRTC's and planes, allocates buffers, and page-flips on vertical blank. It picks the card that has connectors, not simply `card0`, which matters on boards where the display controller and the GPU are separate devices (on the Raspberry Pi 5, `vc4` and `v3d`).

3.2 Two ways to build a frame

- **CPU path.** The UI and app windows are copied into a canvas, and a "dumb" buffer is presented with a page flip. It needs nothing but the kernel. It is the default when there is no 3D-capable card, and `LORD_GPU=0` forces it (the boot menu's safe mode uses it).
- **GPU path.** Mesa's GBM, EGL and GLES2 (`server/src/gpu/gpu.c`). The UI becomes one texture, and only the changed region is uploaded each frame. App windows are not copied into the UI canvas: the UI leaves a transparent hole where each window goes, and window textures are drawn underneath, so an app that only changes its own content doesn't cause a UI repaint. Apps can hand over GPU buffers (`linux-dmabuf`) with no copies, and a fullscreen app with a suitable buffer is scanned out directly. The cursor uses the hardware cursor plane.

The path is chosen at start-up: a card without 3D (simplifiedrm, bochs, virtio-gpu without virgl, and others) uses the CPU without even loading Mesa; a software OpenGL renderer also falls back to the CPU, because composing with it costs more than LordOS's own CPU path. If a GPU frame fails, the GPU is turned off and the CPU path takes over.

Status: the GPU path was exercised with Mesa's software renderer in virtual machines. Testing on real AMD and Nvidia cards is pending.

3.3 Windows and input

- **Window management in C.** Moving and resizing run in C with one event per frame and no JavaScript, and a window being moved is drawn from its own layer. All four edges and corners resize, with LordOS's own cursors.
- **Input without libinput.** Keyboards, mice and tablets are read from evdev. Keyboard layouts come from `libxkbcommon` (every XKB layout, variants and compose).
- **Touchpad and touchscreen.** LordOS has its own multitouch driver (`server/src/tp`): pointer acceleration, tap and two-finger click, palm rejection, kinetic and natural scrolling, pinch, and three-finger swipes. Gestures reach both the LordOS interface and Linux apps (`zwp_pointer_gestures_v1`). It is tested with recorded gestures in `make test` .

3.4 Wayland compatibility

The compositor (`server/src/wl/wl.c`) uses only `libwayland-server` and code generated from the protocol XML. It offers: `wl_compositor` , `wl_subcompositor` , `wl_shm` , `wl_output` (one per display), `xdg_wm_base` with popups, `wl_seat` (pointer, keyboard, touch), `wl_data_device_manager` (a clipboard shared with LordOS), `zxdg_decoration_manager_v1` , the KDE server-decoration protocol (so GTK 3 apps such as Firefox get the theme's frame), `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` and `wp_cursor_shape_manager_v1` .

Linux windows enter the SDK's window manager like any other, with title, focus, maximize, minimize and a Dock icon. Apps that ask for server-side decoration get the theme's frame; apps that draw their own (GTK 4) keep it, and their shadow is clipped. GTK 4 and libadwaita apps follow light/dark mode, the accent and the button order through an `org.freedesktop.portal.Settings` implementation that `lord-server` serves itself on the session bus; GTK 3 follows through the GSettings keyfile.

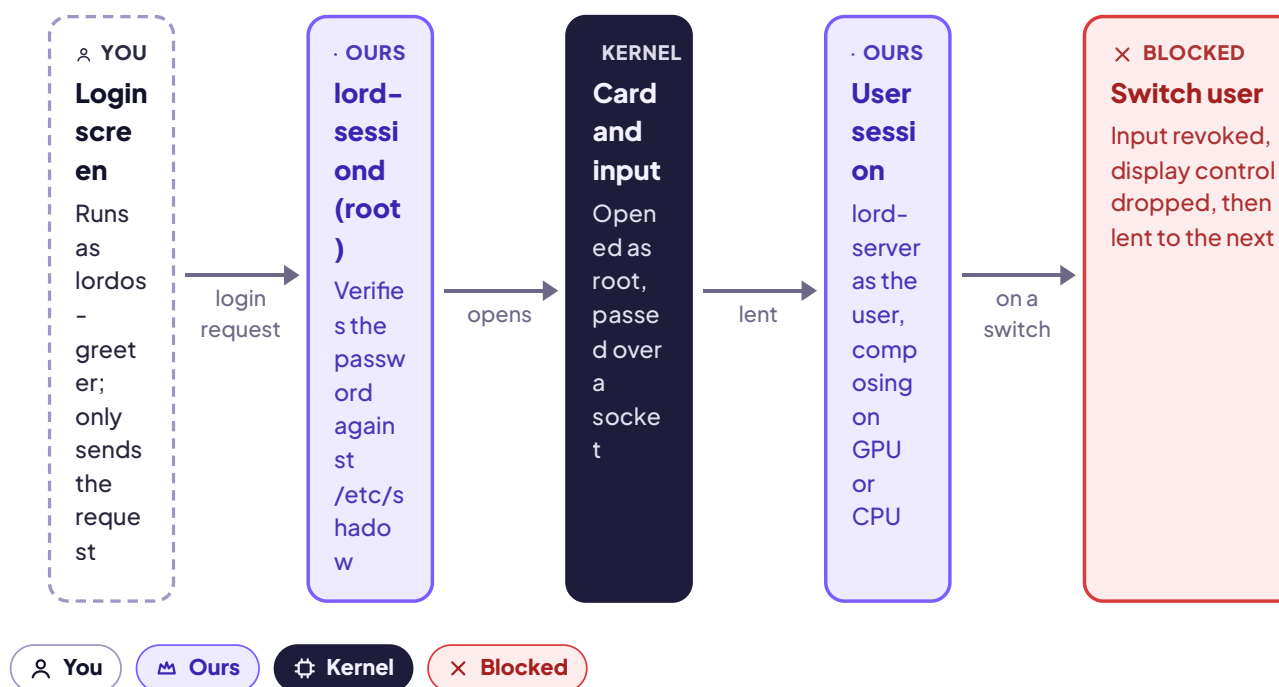
Xwayland is not supported yet: LordOS doesn't run X11-only apps (Xwayland is planned before 1.0, §10.3). Also pending: drag and drop, the primary selection, LordOS input methods inside Linux apps (`text-input`), activation (`xdg-activation`), explicit sync and SVG app icons. No screen-capture protocol is offered to apps.

3.5 Several displays and HiDPI

Every connected display is driven at once, extended or mirrored, in one logical space, with an arrangement remembered by EDID identity and hotplug handling. Each display is painted at its own scale (100, 125, 150, 175 or 200 %), so text and vectors are rasterized at its real resolution. Linux apps get one `wl_output` per display and fractional scaling. Tested with two virtual heads (1280×800 and 2560×1600) on both paths; real monitors, laptop panels and DisplayPort hubs are untested.

3.6 Sessions and login

Lending the display to a session



lord-sessiond is LordOS's own session manager:

- **The login screen** is a **lord-server** running as the unprivileged **lordos-greeter** user, with a special theme. It sends login requests to **lord-sessiond**; **passwords are always verified by the root process** against **/etc/shadow**, never by the greeter's JavaScript. Failed attempts wait 1, 2, 4... seconds, up to 5 minutes.
- **One session per user**, all able to live at once (fast user switching). Each is a **lord-server** that starts as root and drops privileges right away.
- **Display lending.** **lord-sessiond** opens the graphics card and the input devices as root and lends them to the session that is on the monitor, over a Unix socket. On a switch it revokes them: **EVIocrevoke** makes the old session's input file descriptors stop working, and **DROP_MASTER** takes display control away, before anything is given to the next session. The visible session keeps composing on the GPU without ever being root. A session off the monitor draws in memory, and only if someone is watching it remotely.
- **Locking really takes the session off the monitor**; it is not a layer drawn on top. If the user's session hangs, the login screen still works.
- **Boot.** A LordOS GRUB theme draws the boot menu, and a LordOS Plymouth theme draws the splash (the crown with a progress bar). The hand-off has no black frame and no console text: Plymouth stops drawing and gives up display control while its last frame stays on screen, **lord-sessiond** takes display control and sets the first real frame of the login screen, and only then does Plymouth quit, keeping the splash on screen until it is replaced.

Remote desktop. The same service offers the desktop to a browser on the local network at

https://<address>:5673 (5673 spells "LORD" on a phone keypad). It is **off by default** and is turned on in Settings → Sharing. It uses TLS 1.2 or later with a self-signed P-256 certificate whose fingerprint Settings shows

for comparison, a strict content security policy, same-origin checks, per-IP waits on failed logins, and a session cookie that is `Secure`, `HttpOnly` and `SameSite=Strict`. The login page never lists user accounts. The process that serves a connection drops root as soon as it knows whose session it is. Screen updates are sent as PNG when still and JPEG while moving. Audio is designed but not built. Internet access is not intended; a VPN is recommended.

Source files

- [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. The UI engine

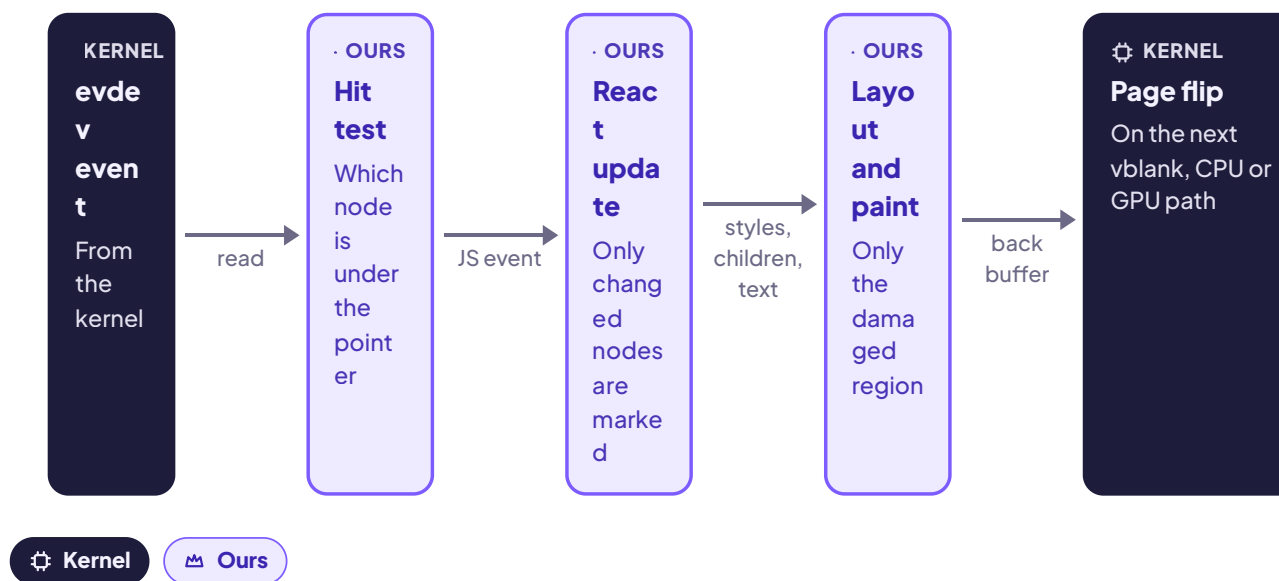
4.1 Not a browser

LordOS's interface is written in JavaScript with React, but there is no web engine underneath. WebKit or Chromium would cost 100 to 300 MB of RAM per process and would not be LordOS's code. Instead, the theme and the apps use a React renderer (`runtime/index.js`) that creates and updates **native nodes** in C: `View`, `Text`, `Image`, plus surfaces for Wayland windows and media. Styles are a subset of CSS (flexbox, colours, rounded and per-side borders, shadows, blur, transforms, transitions) applied by C code that the project controls. The model is close to how a mobile framework renders React to native views, except that the "native" side here is LordOS's own.

4.2 Pieces

- **JavaScript:** QuickJS 2025-04-26, a small ES2023 engine in C, vendored as source and embedded in `lord-server`.
- **React:** React 19.3.1 with `react-reconciler` 0.34, bundled with esbuild into each theme. The default theme with every app weighs 367 KB.
- **Layout:** LordOS's own flexbox implementation (`layout.c`), with wrapping, logical start/end edges and right-to-left mirroring.
- **Painting:** LordOS's own 2D rasterizer with antialiased rounded rectangles, cached shadows, reduced-resolution blur and damage tracking: only the changed region is repainted. On ARM, optional NEON loops speed up blits and masks, verified bit-identical to the C code.
- **Text:** glyphs are rasterized with `stb_truetype`. Shaping is LordOS's own OpenType code (GSUB substitutions and GPOS positioning, colour emoji), with full Unicode bidi, Arabic joining, Indic shaping and the Universal Shaping Engine (409 of 409 test sentences identical to HarfBuzz in `tests/hb_compare.py`), CJK line breaking, Thai word breaking, and per-script font fallback. HarfBuzz, FreeType, fontconfig, fribid and ICU are not linked by `lord-server`.
- **Input methods:** Korean (hangul), Japanese (kana and SKK) and Chinese (pinyin), in `server/src/ime.c`, with no `ibus` or `fcitx`.

From a mouse move to a new frame



The engine repaints only what changed, then flips the page on the next vertical blank.

4.3 How JavaScript reaches the system

The JavaScript side calls native services through `lord.*` APIs implemented in C (`server/src/ui/js_*.c`):

`lord.fs` (files), `lord.settings`, `lord.net`, `lord.pkg`, `lord.media`, `lord.mail`, `lord.vault`, `lord.fw`, `lord.monitor`, `lord.fan` and others. Every call validates its arguments in C. The rules are strict:

- `lord-server` runs without root, and **no API executes arbitrary commands**.
- `lord.fs` writes only in the home folder and the user's own mounts, has no general "delete" (only moving to the Trash), and refuses the folders the system reads to launch or install something or that store security decisions: `~/.local/share/applications`, `~/.config/autostart`, the installed app folder, `settings.json`, the permissions file, the firewall rules, the mail accounts file and the vault folder.
- Anything that needs root goes through `admin.c`: a closed table of actions with arguments validated on the root side, fixed program paths and configuration generated by LordOS, never user text passed to a shell. Passwords go through standard input, never on a command line.
- Long work never blocks a frame: helpers and services are polled from the main loop without waiting.

4.4 Languages

The interface ships in English (the default), Spanish and Portuguese, and more languages are added as language packs without code changes. Text goes through the SDK's `t('namespace.key')`, with plurals and locale formats. The whole interface mirrors for right-to-left languages, and the region (`LANG / LC_*`) is passed to Linux apps.

`node tools/i18n-check.mjs` checks that every key exists in every language and that no third-party brand names appear in the interface.

Source files

- [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. Applications

5.1 Native apps

Around thirty native apps ship with LordOS, all React apps on the same engine and SDK: Files, Settings, Terminal, Store, Text Editor, Notes, Calculator, Monitor, PDF Viewer, Fonts, Emoji & Symbols, Scanner, Photos, Player, Screenshots, Camera, Recorder, Passwords, Disks, Backups, Logs, Clock, Help, Mail, Calendar, Contacts, Reminders, About and the installer. Some highlights:

- **Terminal:** LordOS's own xterm-compatible emulator in C, with tabs, split panes, profiles, inline images and Sixel, and sessions that survive a restart of `lord-server`.
- **Monitor:** CPU, memory, disks, processes, cooling and per-app network connections on a world map, with country lookup from a local database. Its sampling thread exists only while the app is open.
- **Disks:** LordOS's own SMART reader, formatting, and bootable USB sticks verified with SHA-256.
- **Backups:** hard-link snapshots with optional encryption, running at low priority.

5.2 Installable `.lpk` apps

Third-party native apps are packaged as `.lpk`: a gzip-compressed tar with a manifest, the bundled `app.js`, a required icon, optional translations and assets. An app doesn't carry React or the SDK; it uses the system's, which keeps it small (the example Fractal app weighs 5 KB). They install without root into `~/.local/share/lordos/apps/<id>/`. The installer rejects links, devices, absolute paths, `..`, odd characters, more than 2,000 files or 64 MB, and replaces the previous version atomically. Each one runs isolated (§6.4).

5.3 The Store and Debian packages

The Store shows Debian's apps by category with search, `.lpk` apps and local `.deb` files, and always shows a plan before installing or removing. Debian packages are handled by **LordOS's own package engine** (`server/src/pkg`): it reads the apt sources, verifies `InRelease` signatures with its own OpenPGP code (RSA and Ed25519), refuses expired, replayed, wrong-suite or trailing-data indexes, checks every index and `.deb` against SHA-256 values from the signed index, and resolves dependencies (plans identical to apt's in 12 test cases). Only the last step is Debian's: `dpkg` unpacks and configures, and remains the only writer of its database, so `apt` keeps working. LordOS has no TLS client of its own, so `https://` sources are fetched over `http://`; integrity rests on the signatures and hashes, but the package names downloaded are visible on the network.

5.4 Linux apps

Linux apps with a `.desktop` file appear in the app launcher. Launching goes through `lord.apps.launch(id)`: the interface can only name an installed `.desktop` file; `lord-server` re-reads it, checks it, builds `argv` from its `Exec` line by the specification and runs it with `execve`, without a shell or caller arguments, inside its own root-owned cgroup scope. Chromium and Firefox are preinstalled in the ISO and were verified with the LordOS frame. **Linux apps are not sandboxed by default:** they run as the user, as on any Debian system. They get attribution, the

per-app firewall and the per-app restrictions the owner sets (§6.6), including an optional "no access to your files" view applied by the launcher.

5.5 WebAssembly

LordOS includes its own WebAssembly interpreter in C (`server/src/wasm`), exposed as the standard `WebAssembly` global, so a native app can carry modules compiled from C, C++, Rust or Zig. It covers WebAssembly 2.0 without SIMD (the MVP plus sign extension, saturating conversions, multi-value, bulk memory and reference types), passes 26,290 of 26,290 assertions of the official test suite on the VM, and turns every error into a trap rather than a crash. On the project's example it ran 3 to 5 times faster than the same code in JavaScript.

5.6 Installing LordOS

- **Live ISO** for 64-bit PCs, with a graphical installer (UEFI and BIOS).
- **Raspberry Pi 5 image** for SD card or NVMe (§7.4).
- **On an existing Debian 13:** `curl -fsSL https://lordos.org/install.sh | sh`. The script asks before changing anything, downloads `SHA256SUMS` and verifies its Ed25519 signature with the release key embedded in the script, refuses to continue without a valid signature, checks the `.deb`'s SHA-256, installs it with apt and sets the system up like the ISO. The release files are published on GitHub Releases.

Source files

- [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. Security

6.1 Threat model

LordOS is designed against these threats:

- **A malicious or compromised installed app** that wants the user's files, keys, microphone, camera, clipboard or network.
- **Malicious documents and messages:** PDFs, archives, fonts, images, video and mail crafted to exploit a parser.
- **A Linux app that wants to send data out**, or to escape the rules set for it.
- **An attacker on the local network** trying to use the remote desktop or to tamper with downloads.
- **Tampered packages and releases.**

Outside the model, for now: a compromised kernel or root account; physical attacks; side channels; and Linux apps that the owner hasn't restricted, which run as the user, as on any Debian system, and can read the user's files. Linux apps are confined only by the restrictions the owner sets for them (§6.6), and only when they are started from the app launcher.

6.2 Least privilege

- `lord-server` runs as the user. In a managed session it never holds the display or input devices itself: `lord-sessiond` lends them and revokes them.
- Libraries that parse untrusted data are linked only by helpers: FFmpeg and libva by `lord-media` ; Poppler and libarchive by `lord-doc` ; OpenSSL by `lord-maild` and `lord-sessiond` ; Monocypher by `lord-vault` .
- Helpers receive files and devices as file descriptors opened by `lord-server` ; they never open paths.
- The compositor never waits for a helper or an app, so a crash or a hang stays contained.

6.3 The root processes and their closed interfaces

Process	Accepts	Never does
<code>admin.c</code> helper	a fixed table of actions, arguments validated as root	generic actions, user text in a shell
<code>lord-sessiond</code>	login requests from the greeter; short session JSON ($\leq$ 4 KB)	let the greeter verify passwords; read pixels as root
its cgroup placement child	<code>enter <scope> <limits></code> for the calling process only, from a session's <code>server</code> leaf, running a binary from <code>lord-server</code> 's directory	place arbitrary processes; hand the tree back to the user
<code>lord-netd</code>	<code>net-*</code> actions from root peers only	non-network actions; run anything from a session
<code>lord-netperm</code>	<code>RELOAD</code> with no data	run programs; take arguments; open internet sockets
<code>lord-fand</code>	<code>STATUS</code> for anyone; <code>PROFILE</code> , <code>CURVE</code> , <code>FORCE</code> for administrators	name paths or devices; run programs; use the network

6.4 What is isolated, and the app sandbox

Not every program on LordOS is sandboxed. This is exactly what applies to each kind:

Kind	Process	Sandbox	cgroup scope	Broker and permissions	Firewall and restrictions (§6.6)
.lpk apps and Passwords	own (<code>lord-server --app <id></code>)	Landlock, seccomp, no capabilities	own, root-owned, with limits	every call checked in C	network rules and all eight non-network restrictions, in the broker
Other built-in apps	inside <code>lord-server</code>	no (they are part of the desktop)	<code>lord-server</code> 's	no broker; camera, microphone and screen are checked in C	as "LordOS": network rules; camera, microphone, screen and notifications
Linux apps from the app launcher	own	none by default; with "no access to your files", their own user, mount and PID namespaces plus Landlock, from the next launch	own, root-owned	no	network rules; camera, microphone, USB, Bluetooth sockets, files and notifications; not the clipboard
Terminal programs	children of the Terminal's shell	no	<code>lord-server</code> 's	no	as "LordOS": network rules

A program started some other way, outside the app launcher, is in no app scope and gets no per-app rules. Flatpak and bubblewrap are not used.

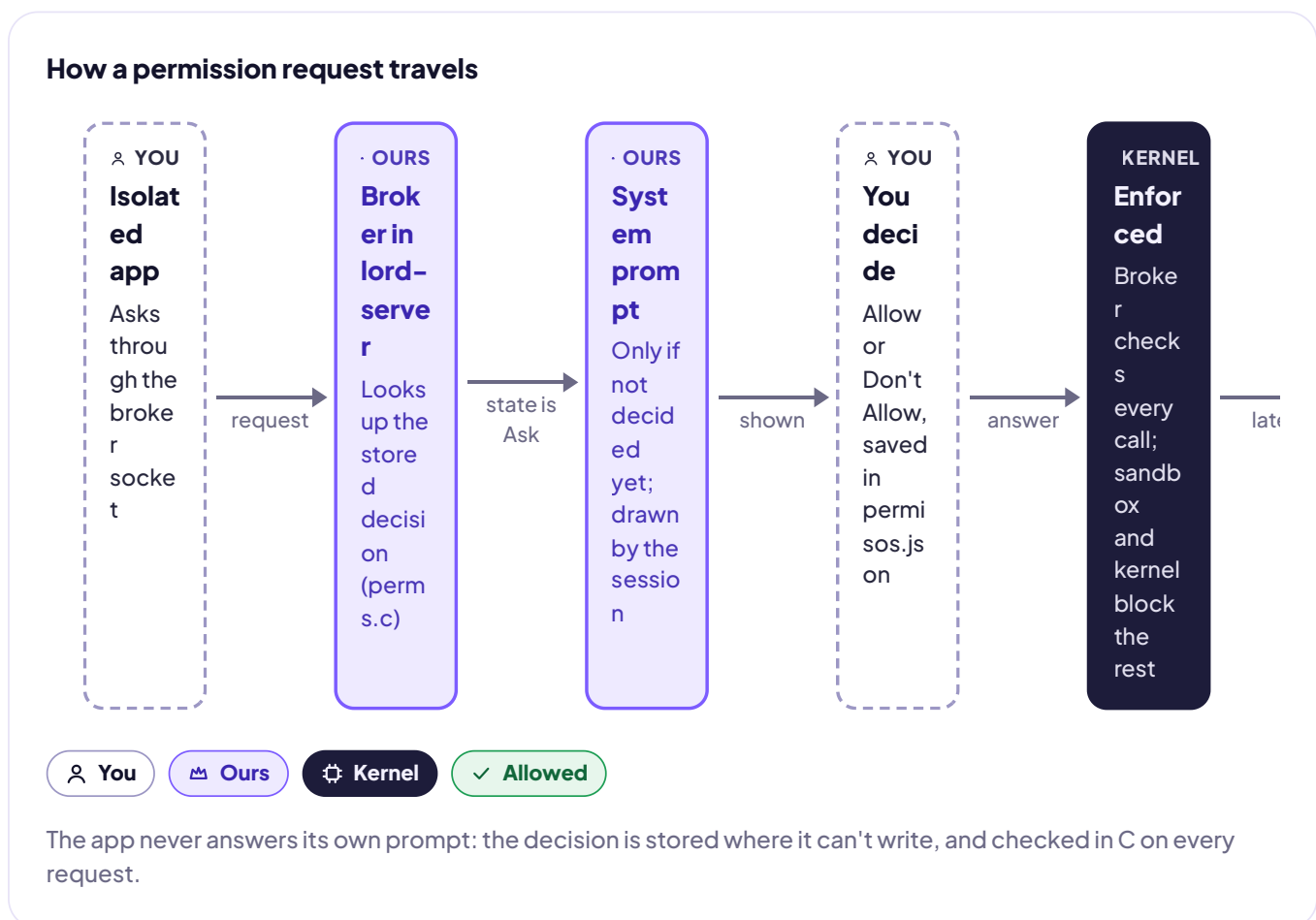
Each `.lpk` app, and the Passwords app, runs as `lord-server --app <id>` : the same binary in "app mode", launched by `lord-server` with a validated package id as its only argument. The process connects to the compositor as a Wayland client and opens the broker socket, then applies the sandbox, and only then loads the app's code (`server/src/sec/sandbox.c`):

- **Landlock** (ABI up to 6): read-only access to the package, the shared app runtime, fonts and libraries; read-write only to `~/local/share/lordos/appdata/<id>` ; scoping of abstract Unix sockets and signals.
- **seccomp-bpf**: an allow list of system calls on every thread (TSYNC), without `ptrace` , `mount` , `kexec` , `bpf` , `perf_event_open` , `keyctl` , module loading or `execve` . New Unix sockets and `socketpair` are refused, because Landlock up to ABI 6 doesn't filter `connect` to named Unix sockets; this keeps the app away from the session bus, the print service and the Wayland socket.
- **No privileges**: `no_new_privs` , an empty capability set and a dropped bounding set.
- **cgroup v2**: `app-lordos-lpk-<id>.scope` , created by root, with `memory.high` 192 MB, `memory.max` 512 MB and one CPU core.

- **Network:** without the `network` permission (declared **and** allowed), seccomp refuses internet sockets and Landlock refuses TCP connect and bind. Network namespaces are not used, because creating one needs privileges `lord-server` doesn't have.

If a kernel feature is missing, the app still starts and the log says which part is missing. The deliberately malicious test app (`tests/lpk-evil`) fails at everything it tries; `build/sandbox-test` passes 14 of 14 checks for apps and 14 of 14 with network. Measured on a VM: about 13.7 MB of RSS per isolated app, a median start of 106 ms, and 0.89 ms to apply the sandbox.

6.5 The broker, permissions and the file dialog



The isolated app talks to `lord-server` only over one `SOCK_SEQPACKET` socket. Every request is checked in C: file access only inside the app's own data folder; clipboard reads only with the `clipboard` permission; permission requests answered from the stored decision or, if none, by a **system-drawn prompt** (*Allow / Don't Allow*); camera and microphone only through `lord-server`.

- **Decisions** are stored per user in `~/.config/lordos/permisos.json`, outside the app's reach. The lookup is $O(1)$.
- **The file dialog is LordOS's own powerbox** (`sdk/powerbox.jsx`), not `xdg-desktop-portal`: `lord-server` draws it, opens the chosen file and passes only that descriptor. The only desktop portal interface LordOS implements is `org.freedesktop.portal.Settings`, for GTK theming.
- **Microphone:** `lord-server` owns the PipeWire stream and gives the app a read-only shared ring.

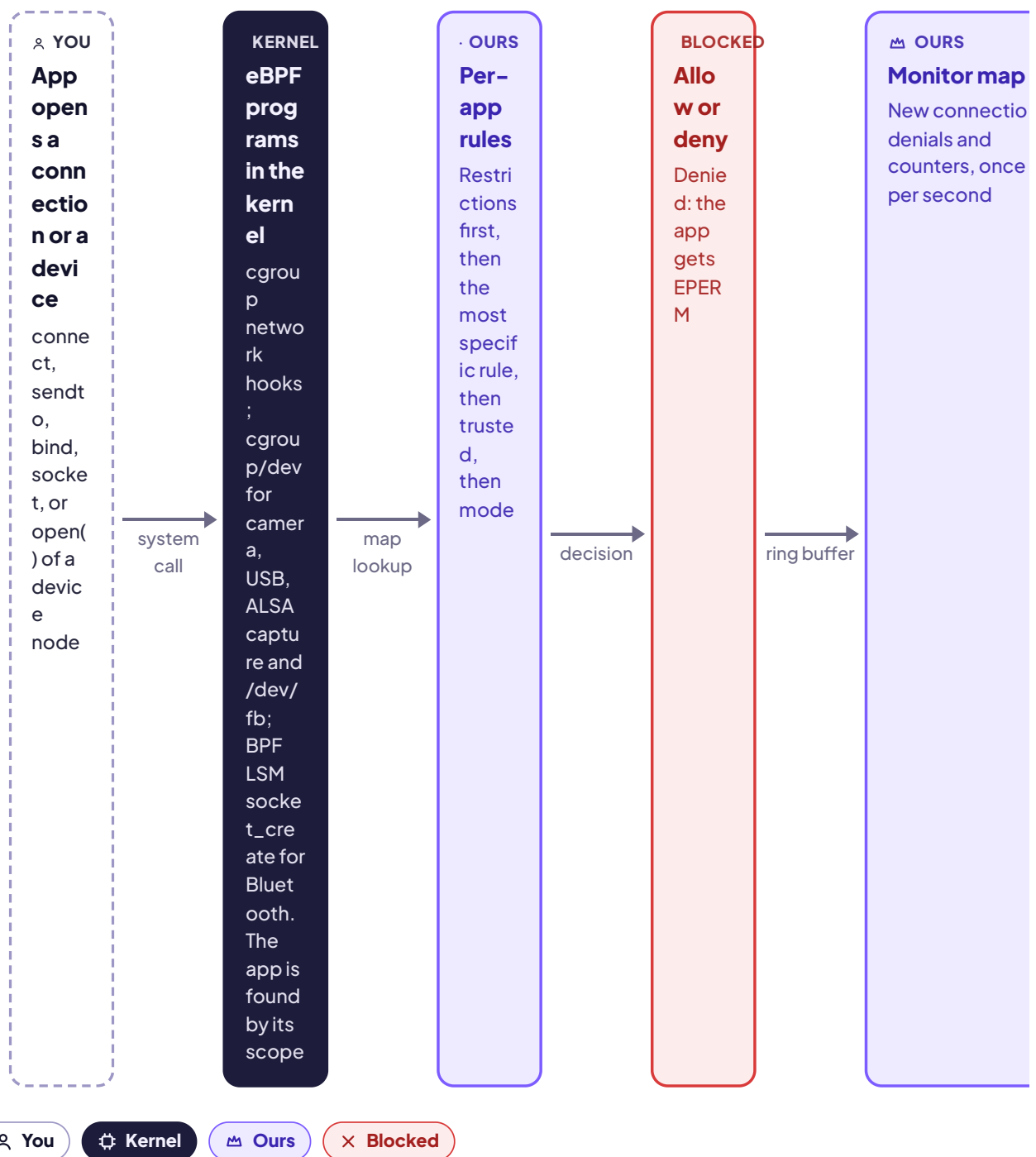
- **Camera:** `lord-server` opens the device and gives it only to `lord-media` ; the app gets a read-only view of the decoded frames and must acknowledge each one, and `lord-server` validates every acknowledgement.
- **Revocation:** denying the camera or the microphone in Settings stops it at once (the app's `lord-media` is killed; the PipeWire stream is closed). The `network` permission is decided at launch.
- **Keyring:** an isolated app's keyring requests carry an owner that `lord-server` sets from the verified app id.

Restrictions beat permissions. A firewall restriction (§6.6) wins over any stored decision: the request is refused without a prompt, even if the app was allowed before, and restricting a running app takes its camera and microphone away at once, like revoking.

Limits. There is no "allow once". Some declared permissions (Bluetooth, launching apps, notifications) have no broker call yet, so isolated apps can't use those capabilities at all; listening on a port is governed by the network permission. A Linux app running as the user could edit the permissions file, unless it is restricted with "no access to your files".

6.6 The per-app firewall, its restrictions and the Monitor

The path of a connection or a device



The decision happens in the kernel when a connection, a socket or a device opens; nothing runs per packet.

lord-netperm enforces per-app rules in the kernel. Linux apps opened from the app launcher run in **app-lordos-<id>.scope**, **.lpk** apps in **app-lordos-lpk-<id>.scope**, and the rest of the session counts as **lordos**. All scopes are root-owned. The service loads seven network programs (**connect4/6**, **sendmsg4/6**, **bind4/6**, **sock_create**), a device program (**cgroup/dev**) and a per-cgroup LSM program on **socket_create** into the cgroup of **lordos.service**, finds each process's app by its cgroup ancestor, and returns allow or deny (**EPERM**). The programs are compiled into the binary and loaded with LordOS's own loader over the **bpf()** system call,

which also reads the kernel's BTF to find the LSM hook; libbpf and bcc are not used. If the kernel refuses the device or the LSM program, the network programs still work and the service says so.

- **Rules** support exact domains, wildcards, IPv4/IPv6 networks, `lan`, `internet`, ports, protocols and direction. Restrictions (no internet, no LAN, no network, no incoming) come first; then the most specific rule wins, with deny beating allow at equal specificity; then trusted apps; then the mode (silent, the default, or off). `npcompile.c` precomputes this into hash and longest-prefix-match maps. A test runs the kernel's decision code against the reference decision maker over 480,000 random connections: 0 differences.
- **Domain names** come only from the system resolver: systemd-resolved's monitor interface or, without it, replies from port 53 of the configured servers read by a receive-only packet socket. Answers an app received on its own sockets are never trusted. So an app that resolves names itself with encrypted DNS (DNS over HTTPS inside a browser, for example) isn't matched by domain rules: an "allow only example.com" rule still blocks it, but a "block example.com" rule doesn't stop it. Address and network rules, the app's default and its restrictions still apply, and Settings says so under the rules.
- **Rules in the interface.** Settings → Privacy → Network rules by app shows each app's default ("allow all" or "block all except rules") and its rules in the order they win, with when each one last matched; the sheet that adds a rule checks what is typed as it is typed. In Monitor → Network, "Allow / Block this domain" on one of an app's destinations opens the same sheet, prefilled with the name the system resolver gave for that address.
- **Escapes.** The cgroup tree is root-owned, so an app can't move itself. A process that asks the user's systemd manager for a new unit (`systemd-run --user`) lands under `user@<uid>.service`; `lord-netperm` attaches the same programs there and holds such processes to the strictest union of that user's running restricted apps. Tested: all escape attempts denied, other users unaffected.
- **Cost:** about 0.5 μs per UDP connect, TCP within noise, no change in throughput, since nothing runs per packet.
- **The Monitor** shows each app's connections (from `sock_diag`) and the destinations the firewall reported recently, bytes, destinations by country from a local database, and a per-app Rule switch. No address ever leaves the machine for lookups: no reverse DNS, no online geolocation.

Restrictions beyond the network. Besides the four network restrictions (no internet, no LAN, no network, no incoming connections), the owner can restrict an app from the camera, the microphone, the screen, USB, files, the clipboard, Bluetooth and notifications. Each one is enforced where the resource is handed out, checked against the app's scope and never against a name the app gives itself, and it beats any permission granted in `permisos.json`.

Where each restriction is enforced

 OURS

Broker in lord-server

.lpk apps and Passwords: all eight restrictions, refused without a prompt



 OURS

App launcher

Linux apps, no access to your files: own user, mount and PID namespaces, a tmpfs home and Landlock, from the next launch



 OURS

Notification server (notifyd)

Linux apps' notifications: the sender's identity comes from the bus; failures drop it



 OURS

PipeWire guard (pwguard)

Microphone and camera through PipeWire: identity from the socket's credentials, applied live



 KERNEL

lord-netperm's eBPF programs in the kernel

Network hooks; cgroup/dev for camera, USB, ALSA capture and /dev/fb; BPF LSM socket_create for Bluetooth sockets

 Ours

 Kernel

Each restriction is checked where the resource is handed out, against the app's root-owned scope, never against a name the app gives itself.

Restriction	.lpk apps and Passwords	Built-in apps ("LordOS")	Linux apps from the app launcher
camera	broker (<code>brokercam.c</code>)	<code>js_media.c</code>	<code>cgroup/dev</code> on V4L2 nodes, and the PipeWire guard
microphone	broker	<code>js_media.c</code>	the PipeWire guard, and <code>cgroup/dev</code> on ALSA capture nodes
screen	broker (no capture API in app mode)	<code>js_media.c</code>	by design: no capture protocol, no X server, no screen-cast portal; <code>/dev/fb*</code> denied
USB	broker (the security key of Passwords)	not offered	<code>cgroup/dev</code> : raw USB, hidraw and USB serial nodes
files	broker (file picker, camera saves)	not offered	the launcher: own namespaces, a tmpfs home and Landlock, from the next launch
clipboard	broker	not offered	not available
Bluetooth	broker (no Bluetooth API in app mode)	not offered	BPF LSM program on <code>socket_create</code> : no <code>AF_BLUETOOTH</code> sockets
notifications	the <code>notifications</code> permission	<code>sdk/notify.js</code>	<code>notifyd</code> , by the caller's verified identity

For isolated apps the broker (`sec/restrict.c`) is the single enforcement point: a restricted request is refused without a prompt, and restricting a running app takes its camera and microphone away at once (`tests/broker_restrict_test.c` covers every restriction, with and without it). The "LordOS" row is the desktop itself (Files, copy and paste, Disks, Bluetooth settings), so it doesn't offer files, USB, clipboard or Bluetooth; individual apps are restricted instead. For Linux apps:

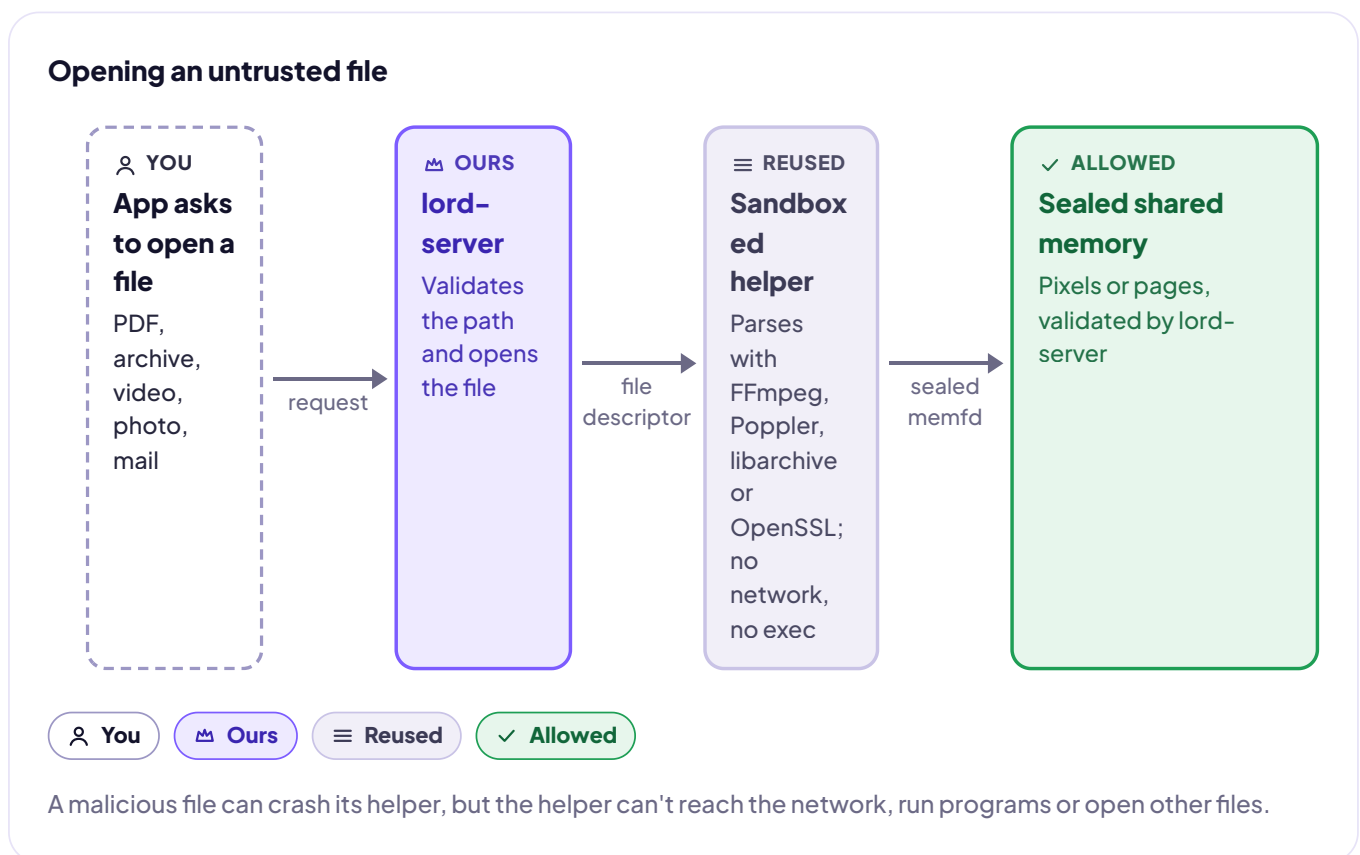
- **Devices.** In cgroup v2 the device controller is a BPF program. `lord-netperm` 's refuses `open()` and `mknod()` of the device classes the app is restricted from (V4L2, raw USB, hidraw, USB serial, ALSA capture nodes, `/dev/fb*`) and allows everything else. It applies to the next `open` ; a descriptor already open stays usable.
- **Bluetooth sockets.** The inet socket hook never sees `AF_BLUETOOTH` , so a per-cgroup LSM program on `socket_create` refuses them.
- **Files.** Right before `execve` , the launcher's child enters its own unprivileged user, mount and PID namespaces. The home folder is an empty tmpfs holding only the app's own folders (derived from its desktop id, never from fields a `.desktop` file controls) and the file it was opened with; the runtime folder holds only the Wayland and sound sockets, so the session bus and the user's systemd are out of reach; then Landlock limits the rest. It applies at the next launch. GTK apps, Firefox and Chromium run without their D-Bus integrations.
- **Notifications.** `notifyd` asks the bus for the caller's PID (only an answer the bus itself sent counts), finds its app scope, and drops the notification of a restricted app; an error or a late answer drops it too (fail closed).
- **Microphone and camera through PipeWire.** The PipeWire guard (`sec/pwguard.c`), `lord-server` 's own PipeWire client, identifies each client by the PID PipeWire took from the socket's credentials and, for a restricted app, hides capture nodes and destroys its capture streams within a second, a recording already

running included. While a restricted app runs, a client whose identity can't be established is treated as restricted.

Limits. "Ask" mode is designed but behaves as silent today. ICMP is blocked only by "no network"; connections to an already-open port aren't checked. The restrictions for Linux apps don't cover:

- BlueZ over D-Bus: a Bluetooth-restricted app can still ask `bluetoothd` on the system bus to scan, pair and connect;
- the system bus, which stays reachable even with "no access to your files";
- the clipboard;
- apps started outside the app launcher, which are in no app scope;
- side effects of the `user@` union: while a restricted app runs, the user's own services are held to the same union (a headset's call audio, which PipeWire opens per call, stops for that time);
- a window of a few milliseconds in which a new PipeWire client of a restricted app has full permissions, or in which a capture stream can receive audio before it is destroyed.

6.7 Helpers that process untrusted data



`lord-doc` (PDF, archives, fonts, scanner images), `lord-media` (video, audio, photos, camera), `lord-maild` (mail, calendars, contacts) and `lord-vault` (secrets) share one hardened sandbox, `sandbox_apply_helper()` :

- no `socket` , `socketpair` , `connect` , `bind` , `listen` , `accept` ; no `execve` ; `clone` only for threads; `seccomp` on every thread, and if `seccomp` can't be applied the helper exits;

- files only through descriptors passed by `lord-server` ; ioctls only by class (DRM, V4L2, DMA-BUF) where needed;
- results returned in sealed shared memory, validated by `lord-server` .

`lord-media` doesn't even receive the PipeWire socket; audio flows through a shared ring whose stream belongs to `lord-server` . `lord-maild` can't create sockets: it asks `lord-server` , which connects only to servers in the accounts file (and, after the user asks to load images, only to public addresses on ports 80/443 for ten minutes) and hands over the connected socket. `build/sandbox-test helper` passes 17 of 17 checks.

6.8 Package signing and trust levels

`.lpk` packages carry a manifest of every file's SHA-256 and an Ed25519 signature over it. SHA-256 and Ed25519 are LordOS's own code (Ed25519 following TweetNaCl's arithmetic), tested against NIST and RFC 8032 vectors (55 of 55). Verification happens in C at install and at load. The Store shows the trust level: **LordOS** (official key, compiled into the binary so no file can add one), **Verified author** (listed in the signed catalog), **Signed by <name>**, or **Unsigned** with a warning. Updates must be signed with the same key. A signed catalog carries a revocation list. If you edit an installed app, LordOS marks it "Modified by you", can restore the original, or can share your version signed with your own key, with credit to the original.

Status: in 0.8.0 Beta, official packages are still signed with a development key, until the final key replaces it. A signed LordOS app repository for the Store is not built yet.

6.9 Release signing

Every release file (the `.deb` , the ISO and the Raspberry Pi image) is listed with its SHA-256 in `SHA256SUMS` , and `SHA256SUMS` is signed with the LordOS release key (Ed25519). The public key is `tools/release/lordos-release.pub.pem` , also embedded in the Debian installer. Its fingerprint (SHA-256 of the DER public key) is:

```
c7155b8e6bbbbb180f907381994d06af76d333016cd6f3facf67cadd053ea108e
```

The private key lives only on the maintainer's machine, never in the repository or on a build machine. Each release also carries an SPDX 2.3 software bill of materials (SBOM) per architecture, listed in `SHA256SUMS` and so covered by the signature, and its release notes come from `CHANGELOG.md` , kept in the Keep a Changelog format. §11 shows how to check a download.

6.10 The vault

`lord-vault` keeps passwords and the session keyring in `.lvault` files: one random 32-byte vault key, wrapped once per unlock method (Argon2id password with a cost calibrated per machine, FIDO2 `hmac-secret` with LordOS's own CTAP2 client, key file, password plus key file, recovery key); content sealed with XChaCha20-Poly1305 with the whole header as associated data; attachments sealed separately; atomic saves with a backup. Every flipped byte and every truncation is detected (1,165 of 1,165 in the test). Secrets live in an `mlock` ed arena excluded from core dumps and wiped on free, in a process that is not dumpable. The Passwords window runs in its own sandboxed process, so revealed values never enter the desktop's JavaScript. The keyring opens at login with the password `lord-sessionid` verified, and serves Linux apps through LordOS's own Secret Service implementation. Real hardware security keys are not yet tested.

6.11 Protected windows

A window marked protected (Passwords is) is painted black in every path that copies pixels out: screenshots, recordings, thumbnails, the remote desktop, and Wayland menus and fullscreen. The real monitor shows it normally. Linux apps have no screen-capture protocol to begin with.

6.12 Disclosure and standards

Vulnerabilities are reported as described in [SECURITY.md](#) , with a machine-readable contact at <https://lordos.org/.well-known/security.txt> (RFC 9116). LordOS's processes are **aligned with, not certified against**, ISO/IEC 29147 and ISO/IEC 30111 (vulnerability disclosure and handling), ISO/IEC 5230 (OpenChain license compliance: the inventory in [THIRD_PARTY.md](#) , the license notices and the SBOM) and ISO/IEC 18974 (OpenChain security assurance: the SBOM plus Debian's security updates for the packages LordOS depends on). lordos.org publishes the legal and trust pages (security policy, code signing, licenses, privacy, support lifecycle, standards) and says what is still missing: no formal self-certification, no automated matching of the SBOM against vulnerability databases, no SBOM for the whole images, and no third-party audit.

Source files

- [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. Performance

All figures below were measured by the project and are documented in the files cited. Most come from virtual machines; real-hardware measurements are noted as such.

7.1 Memory and idle

Measurement	Value	Where
Whole system, live ISO, idle after a minute (4 GB, 4 cores)	465 MB used, CPU 99 % idle	VM 117, docs/ISO.md
Whole system, installed from the ISO, idle	487 MB used, CPU 99 % idle	VM 120, docs/ISO.md
Boot, live ISO with UEFI	16.3 s (4.9 s kernel + 11.4 s userspace)	VM 117, docs/ISO.md
<code>lord-server</code> without a 3D card (Mesa not loaded)	about 32 MB RSS	docs/GPU.md
A user session (<code>lord-server</code>), remote desktop tests	about 50 MB; login screen 41 MB	VM 118, docs/REMOTE.md
One isolated <code>.lpk</code> app	13.7 MB RSS	VM 122, docs/PRIVACY.md

7.2 Drawing

- Dragging a window on one 1280×800 display on the CPU path: about 6.7–6.9 ms of work per frame, around 120 frames per second (VM 118, [docs/DISPLAYS.md](#)).
- The same window on a 2560×1600 display at 150 %: about 5.1 ms per frame.
- Video: painting straight from the decoded frame when only an opaque video changed brought `lord-server` from 42 % to 13 % of a core for a windowed 1080p video, and from 92 % to 15 % fullscreen ([docs/ROADMAP.md](#) , media).
- The Monitor closed costs nothing: its sampling thread doesn't exist until opened ([docs/MONITOR.md](#)).

7.3 GPU

On the VMs only a software renderer was available. With it, the CPU path is cheaper than "GPU" composition, which is why LordOS uses the CPU path when the only renderer is a software one. The GPU path's real benefit (composition and present close to free, zero-copy fullscreen) is expected on real cards, but **real AMD and Nvidia hardware has not been measured yet**.

7.4 The Raspberry Pi 5

The Raspberry Pi 5 is the only supported board. Results on a real board (16 GB, SD card, without a monitor), from [docs/RASPBERRY.md](#) :

Measurement	Value
Building every LordOS binary on the board	1 min 02 s, 0 warnings
<code>make -C server test</code>	23 passed, 0 failed, 0 skipped
NEON blit and mask vs. plain C	9.90 → 4.33 ms and 11.09 → 5.49 ms per screen (2.0–2.3×), bit-identical
Idle at the login screen, headless	<code>lord-server</code> 36 MB RSS, 0.6 % CPU; system 450 MB of 16 GB; 38–43 °C, no throttling
Building the SD image on the board	8 min 18 s; 503 MB compressed
Building the LordOS kernel on the board	about 50 min per flavour

The LordOS kernel. The Raspberry Pi's own kernel doesn't build Landlock. Without it, the document-helper hostile test fails 5 of its 25 filesystem checks and the sandbox test can't run. LordOS builds the Raspberry Pi kernel from its own source and default configuration, changing only three options (Landlock on, the LSM list, the version suffix), and pins it so a newer archive kernel can't replace it. The image also enables the memory cgroup, which Raspberry Pi turns off at boot.

Fans. On the Pi 5 in a test case, `lord-fand` kept the fan off at 50–53 °C during a 4-thread load (the aluminium case acts as a heatsink), drove it to 55 % (4,500 RPM) and full speed (7,400 RPM) with test curves, and restored the kernel's own settings after `SIGTERM` and `kill -9` (`docs/FANS.md`).

Not verified yet on the board: anything with a monitor attached (modesetting, GPU composition on screen, frame rates), the desktop session, and booting the built image.

7.5 Modern PCs

There are no measurements on real PCs in the repository yet. The ISO is tested on virtual machines (UEFI and BIOS); testing on a real AMD machine from the live USB is planned.

Source files

- [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. Visual experience

- **A design guide.** Apps follow a shared guide (`docs/DESIGN.md`): an 8 px spacing grid, a small type scale, consistent control heights and one accent-coloured action per view. The tokens are exported by the SDK

(`spacing` , `typeScale` , `controlSize`), and every app went through documented before-and-after polish passes in light, dark, right-to-left and small windows.

- **Themes.** A theme is a complete desktop written in React: menu bar, Dock, app launcher, Control Center and window frame, plus its tokens. LordOS ships a default theme with a menu bar and a Dock, and a Classic theme with a taskbar and a start menu, which today is a template and becomes a second full appearance before 1.0 (§10.3). Themes switch live; settings persist.
- **Light and dark, accents.** Both modes and the accent colour apply live to native apps, isolated apps and, through the portal and GSettings, GTK apps.
- **Right to left.** The whole interface mirrors for Arabic, Hebrew and similar scripts, including isolated apps, which follow the session's language live.
- **Lumo.** LordOS's mascot is Lumo, a small celestial fox, used in welcome screens, empty states and brand material, with a fixed set of poses defined in the brand guide.
- **Accessibility today:** a "reduce motion" setting that turns off decorative animations, keyboard shortcuts throughout, and scalable displays up to 200 %. A screen reader, zoom and high contrast are on the roadmap and **not built yet**.

Source files

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

9. Extensibility

- **The SDK** (`@lordos/sdk`) is the only thing apps import: windows, settings, theme tokens, controls, files, media, notifications, keyring, printing and more. Apps don't import from themes, so any app works in any theme.
- **Writing an app** means a folder with `manifest.json` and `index.jsx` . `node tools/lpk.mjs <folder>` bundles it without React or the SDK, adds the icon and translations, and signs it if a key is given; `keygen` , `sign` , `verify` and `catalog` cover the rest. Apps can include WebAssembly modules and draw raw pixels with `<Image pixels>` .
- **Themes** follow a fixed contract (`Desktop` , `MenuBar` , `Dock` , `Launcher` , `ControlCenter` , `WindowFrame` , tokens), documented in `themes/README.md` , with the Classic theme as a template.
- **Linux interoperability** is through standards: Wayland, the Settings portal, the Secret Service, the notifications service, `mimeapps.list` default apps and `.desktop` files.
- **Automation:** there is no scripting or automation API beyond the SDK today. Integrations with AI assistants (for example through the Model Context Protocol) **do not exist in LordOS**; they are only a possible future direction and would have to respect the same rule that JavaScript can't execute commands.

Source files

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

10. Roadmap and limitations

10.1 Where things stand

0.8.0 Beta is the first public release; 0.7.0 was an internal build and was never published. Milestones M1 to M9 are complete: the display server, the UI engine, the SDK and themes, the terminal, languages and complex scripts, Linux app compatibility, packages and system services, sessions, isolation and the firewall, and the M9 app suites (documents and printing, media, system apps, passwords, hardware settings, mail and calendar). Before the release, the firewall's non-network restrictions became enforced for every kind of app, domain rules got an interface in Settings and the Monitor, the boot got a LordOS splash and GRUB theme, and releases gained a signed SBOM and a changelog. The final ISO was built and tested on virtual machines on 18 October 2026.

10.2 Known limitations

- Real hardware: AMD and Nvidia GPUs, real monitors, laptops (battery, backlight, lid), Bluetooth controllers, printers and scanners, FIDO2 keys and real mail providers are untested.
- The Raspberry Pi 5 is the only supported board, with LordOS's build of its kernel with Landlock; it hasn't been tested with a monitor, and its built image hasn't been booted yet.
- Linux apps are not sandboxed by default; no Xwayland yet.
- Firewall: no ask mode; domain rules don't see apps that resolve names with their own encrypted DNS.
- Restrictions for Linux apps don't cover BlueZ over D-Bus, the system bus, the clipboard or apps started outside the app launcher; the `user@` union also holds the user's own services while a restricted app runs; a new PipeWire client has a window of a few milliseconds before the guard restricts it (§6.6).
- The `.lpk` signing key is still a development key, and there is no signed LordOS app repository yet.
- Permissions: no "allow once"; camera and microphone use by Linux apps isn't shown.
- Secure Boot must be turned off; signing with a shim is pending.
- Package downloads use HTTP (signed and hashed).
- Remote desktop has no audio.
- Accessibility (screen reader, zoom, high contrast) is not built.

10.3 Next

Next, a Raspberry Pi 5 image booted and tested on the real board. Before 1.0, as planned in [docs/ROADMAP.md](#) :

- **Xwayland**, started on demand, with the compositor as the X window manager and X11 apps getting the same per-app identity, firewall and restrictions as Wayland apps;
- **the Classic theme as a second full appearance**: taskbar, start menu with search, tray, quick settings, notification panel and window snapping, with the same apps in a different skin, built only from LordOS's own design and assets.

After 1.0: a desktop virtualization app built on KVM and QEMU; an Android container with LineageOS and apps from LordOS's Store; the firewall's ask mode; accessibility; and a full demo video.

Source files

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

11. How to verify the project yourself

11.1 Build it

On a development machine with Node.js, bundle the themes, SDK and apps:

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

On Debian 13, build the binaries and run the tests:

```
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` fails if any suite fails; a suite whose tool or data isn't installed reports "skipped" rather than passing silently. The sandbox tests need a kernel with Landlock. The ISO and the Raspberry Pi image are built as described in [docs/ISO.md](#) and [docs/RASPBERRY.md](#).

11.2 What the tests cover

- **Terminal and text:** the VT emulator, OpenType shaping against HarfBuzz, Unicode bidi against the official test file, input methods, Thai word breaking against libthai.
- **Security:** the sandbox (app, network, helper), signing (NIST and RFC 8032 vectors), package state, archive path rules against malicious archives, `lord.fs` protections, the firewall rules (with 20,000 JSON mutations and 30,000 random operations against a linear reference), the DNS reader (200,000 random packets), the kernel-map compiler (480,000 connections), and the kernel integration test (18 of 18 on a VM).
- **Parsers:** mail, MIME, HTML sanitizer, iCalendar, vCard and XML (30,953 checks with mutation fuzzing), the vault format, importers, QR, CBOR, D-Bus marshalling, the country database reader, journal, TZif and IPP.
- **Engine and hardware:** layout and painting, NEON vs. C, the touchpad driver with recorded gestures, fans on fake sysfs trees, display arrangement, sound, and the WebAssembly specification suite.

On the final ISO build: `make -C server test` reported 23 passed, 0 failed, 1 skipped (bidi data not installed).

11.3 Check a release

Download `SHA256SUMS`, `SHA256SUMS.sig`, the release files and the public key (`tools/release/lordos-release.pub.pem`). Check that the key's fingerprint is

`c7155b8e6bbbbb180f907381994d06af76d333016cd6f3facf67cadd053ea108e`, then, with OpenSSL 3:

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

The SBOM of each architecture (`lordos_<version>_<arch>.spdx.json`) is listed in `SHA256SUMS` too, so the same check covers it.

11.4 Check a running system

- `systemctl status lord-netperm lord-fand lordos` shows the services; the ISO test confirmed the seven network programs attached. If the kernel refuses the device or the LSM program, `lord-netperm` reports `"devices":false` or `"bluetooth":false`.
- `cat /proc/<pid>/status | grep -E 'Seccomp|NoNewPrivs'` on a `lord-server --app` process shows the sandbox.
- `systemd-cgls` shows the root-owned `lordos-session-<uid>` tree with one scope per app.
- Remote desktop: `ss -ltn` shows nothing on port 5673 until you enable it in Settings → Sharing.

12. License and credits

LordOS's code is released under the **MIT license**. Third-party components keep their own licenses, listed in `THIRD_PARTY.md`: QuickJS and stb (vendored), Monocypher (CC0/BSD-2), the EFF word list (CC BY 3.0 US), the common-password list (MIT), DB-IP Lite country data (CC BY 4.0), Natural Earth (public domain), and the Inter, JetBrains Mono and Plus Jakarta Sans fonts (OFL). Each release carries an SPDX SBOM of what the package bundles and depends on (`docs/COMPLIANCE.md`).

LordOS is created by the **iPERFEX** group: **Pablo Miranda** and **Federico Pereira**.

13. Glossary

- **Broker**: the single socket through which an isolated app asks `lord-server` for anything that touches the user's things; every request is checked in C.
- **cgroup (v2)**: a kernel mechanism that groups processes to account for and limit resources. LordOS gives each app its own group, a "scope", and uses it to identify the app in the firewall.
- **Compositor**: the program that combines the windows of all apps into the final image on screen.
- **DRM/KMS**: the kernel interfaces for graphics devices and for setting display modes.
- **eBPF**: small verified programs that run inside the kernel; LordOS attaches them to cgroups to decide network connections, device opens and Bluetooth sockets.
- **evdev**: the kernel interface for input devices.
- **LSM (BPF LSM)**: the kernel's security hooks; with the `bpf` LSM, a small program attached per cgroup can refuse an operation, such as creating a Bluetooth socket.
- **Namespace**: a kernel feature that gives a process its own view of something (users, mounts, process ids); LordOS uses unprivileged ones for Linux apps restricted from the user's files.
- **Helper**: a small LordOS process that handles untrusted data or secrets in a stricter sandbox (`lord-doc` , `lord-media` , `lord-maild` , `lord-vault`).
- **Landlock**: a kernel feature that lets an unprivileged process restrict which files and network ports it, and its children, may use.
- `.lpk` : LordOS's package format for installable native apps.
- `.lvault` : LordOS's encrypted vault file format.

- **PipeWire guard:** `lord-server` 's own PipeWire client that hides capture devices from restricted Linux apps and stops their capture streams.
- **Powerbox:** a file dialog drawn by the system that hands the app only the file the user chose.
- **QuickJS:** a small JavaScript engine written in C.
- **SBOM:** a software bill of materials, the machine-readable list of the components a release contains, in the SPDX format.
- **Scope:** a cgroup for one app, such as `app-lordos-lpk-<id>.scope` .
- **seccomp-bpf:** a kernel filter that restricts which system calls a process may make.
- **Secret Service:** the freedesktop D-Bus standard Linux apps use to store secrets.
- **Wayland:** the protocol between graphical apps and the compositor.

14. Where to read more

Chapter	Documents in <code>docs/</code>
1. Vision	ARCHITECTURE.md, LORDOS-BRAND.md
2. Architecture	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. UI engine	ARCHITECTURE.md, LANGUAGES.md, TERMINAL.md
5. Applications	NATIVE-APPS.md, PACKAGES.md, WASM.md, FILES.md, MEDIA.md, DOCUMENTS.md, MAIL.md, ISO.md, INSTALL-DEBIAN.md
6. Security	PRIVACY.md, FIREWALL.md, MONITOR.md, SIGNING.md, PASSWORDS.md, MEDIA.md, REMOTE.md, COMPLIANCE.md, and <code>SECURITY.md</code>
7. Performance	ISO.md, RASPBERRY.md, FANS.md, DISPLAYS.md, GPU.md
8. Visual experience	DESIGN.md, LORDOS-BRAND.md, LANGUAGES.md, DOCK.md
9. Extensibility	NATIVE-APPS.md, WASM.md, and <code>apps/README.md</code> , <code>themes/README.md</code>
10. Roadmap	ROADMAP.md, RELEASING.md and <code>CHANGELOG.md</code>