ODYSSEY LINUX · REFERENCE DOCUMENTATION

Odyssey Control Center — Practical Guide

Every panel, section by section. What it does, what it changes, and what to click — written for someone who has never used it before.

Launch Edition

Don’t trust me. Verify me.

— nobody

Odyssey Linux — Odyssey Control Center: Practical Guide
Launch Edition. Covers the Control Center as shipped on the Launch Edition ISO — ten panels, from Boot to the NVIDIA module.

Copyright © 2026 the Odyssey Linux project.
Released under the Creative Commons Attribution-ShareAlike 4.0 International license (CC BY-SA 4.0).

This guide is a companion to the Security and Hardening Guide, which covers the Firewall and AppArmor panels in full depth — this document treats those two briefly and points there for everything else.

Every screen, button and behavior described here was read from the actual Control Center source, not from memory.

CONTENTS

1 What the Control Center is 1
1.1 Patterns you'll see in every panel 2
2 Boot 3
2.1 "How do I see other operating systems I have installed?" 3
2.2 The default boot entry 4
2.3 "How do I choose EFI?" — the EFI tab 4
2.4 Kernel parameters 5
3 Kernel 6
3.1 SYSCTL — the tunables 6
3.2 CPU governor 7
3.3 Hugepages 7
3.4 Modules 8
4 Memory & Swap 9
4.1 System memory 9
4.2 Swappiness 9
4.3 ZRAM — compressed RAM swap 9
5 Firewall 11
6 AppArmor 13
7 Privacy 15
7.1 SearXNG — private search 15
7.2 Tor — network anonymity 15
7.3 Routing SearXNG's own queries through Tor 16
8 Containers 17
8.1 Engine 17
8.2 Containers, and rootless vs. rootful 17
8.3 Available containers — the curated catalog 17
8.4 Run a container 17
8.5 Images and disk usage 18
9 Gaming Mode 19
9.1 The main toggle 19
9.2 Packages, and Steam specifically 19
9.3 GameMode and MangoHud 20
9.4 Proton 20
10 Telemetry 21
10.1 Connections 21
10.2 Traffic 21
10.3 DNS 22
10.4 History 22
11 About 23
11.1 Copy report 23
12 NVIDIA driver 24
12.1 Detected hardware and branch selection 24
12.2 Installing the driver 24
12.3 PRIME render offload — hybrid laptops 25

CHAPTER 1

What the Control Center is

The Odyssey Control Center is ten small, independent programs behind one launcher — one per area of the system: Boot, Kernel, Memory & Swap, Firewall, AppArmor, Privacy, Containers, Gaming Mode, Telemetry, About, and (on NVIDIA hardware) the NVIDIA driver module.

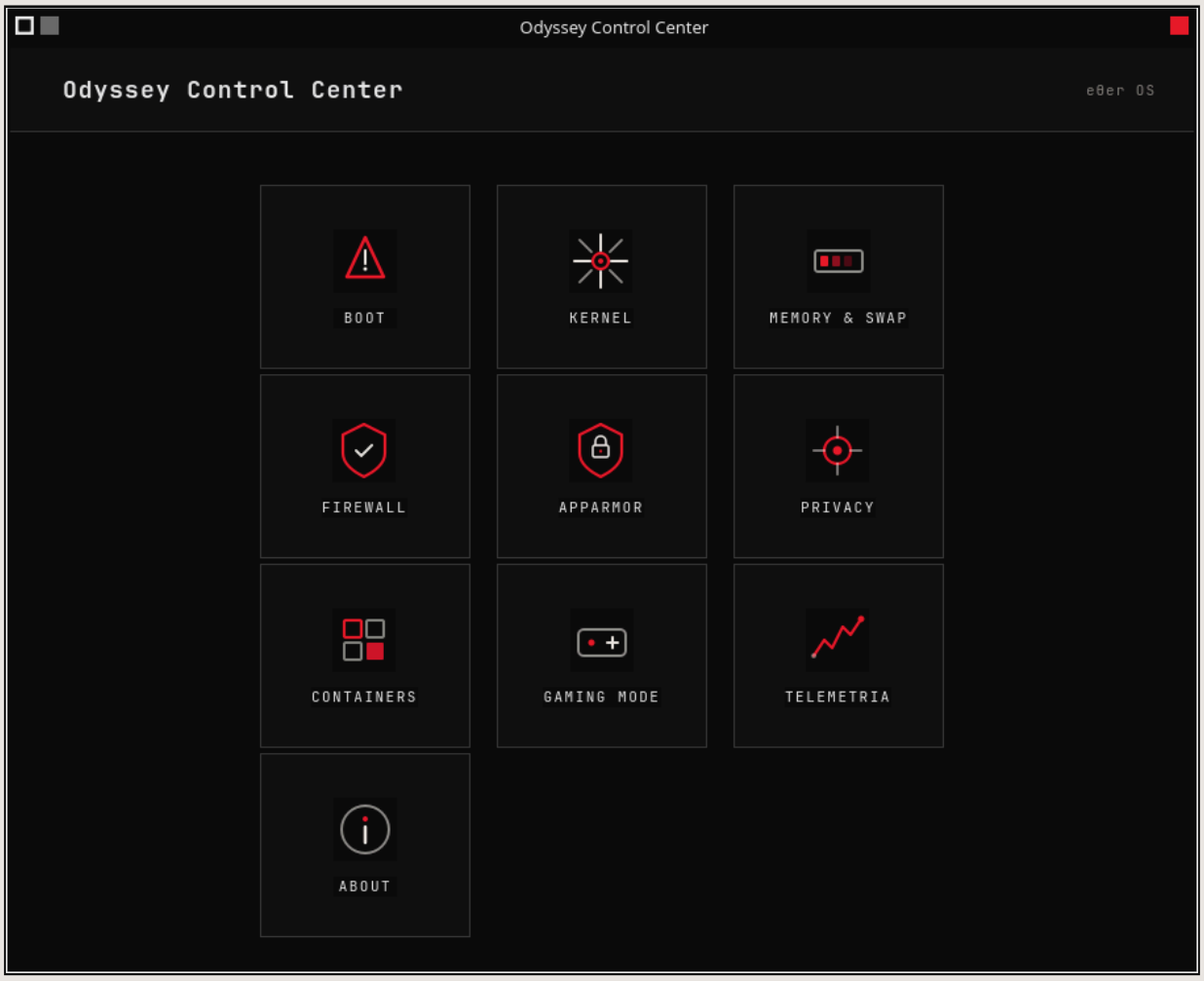

FIG · The Control Center home screen — one tile per panel.

Each panel is a thin graphical layer over the real Linux mechanism underneath: GRUB's own config, the kernel's own sysctl interface, `nftables`, `podman`. A panel never invents its own private state — it reads the actual files and commands, shows them to you, and writes back what you asked for. Nothing it does is hidden, and everything it does can be done by hand instead, from a terminal, with the exact commands this guide shows alongside each button.

This is a practical guide, not a reference of every option — it answers "what do I click, and what happens if I do." For the two panels with real security stakes — Firewall and AppArmor — the deep reference is the **Security and Hardening Guide**; here you get the short version and a pointer.

1.1 Patterns you'll see in every panel

A few conventions repeat across all ten panels — learn them once.

THE PASSWORD PROMPT, ONCE PER PANEL

Most panels need to read or write files that belong to root — GRUB's config, sysctl, the firewall ruleset. The first time a panel needs that, you get one authentication dialog. Panels are built to batch their reads: opening Boot, for instance, reads GRUB's config **and** the EFI firmware state in a single request, so you type your password once, not twice. Re-opening a panel in the same session does not ask again; actions that follow (applying a change, reloading) show their own progress in the panel's own log instead.

THE STEP TRACKER

Panels that guide you through installing something before it becomes usable — os-prober in Boot, SearXNG and Tor in Privacy, the container engine in Containers — show a small tracker across the top: a row of steps, with the current one highlighted, and one action button that always does **the next step**, whatever it is. You never have to guess what to click first.

THE LOG AT THE BOTTOM

Every panel ends in a small scrolling log. It is not decoration — it is the exact command that ran, its output, and a colored verdict (green for success, red for failure). If a panel does something you don't understand, the log is where the real answer is.

TRY VS. MAKE PERSISTENT

A recurring pair of buttons, in Kernel and Memory especially: **TRY NOW** (or **APPLY NOW**) changes the running system immediately, and reverts on the next reboot if you do nothing more. **MAKE PERSISTENT** writes the same value to a config file under `/etc/`, so it survives reboots too. The split exists on purpose — it costs nothing to try a value, and nothing is permanent until you say so twice.

CHAPTER 2

Boot

The Boot panel has two very different jobs, kept in two separate tabs on machines that boot in UEFI mode (the **GRUB** tab and the **EFI** tab — on older BIOS-only machines you only ever see GRUB, because the EFI layer below simply doesn't exist there).

> **TWO LAYERS, NOT ONE**
>
> **GRUB** is the boot menu you see on screen — it decides which operating system starts once GRUB itself is already running. **EFI/UEFI firmware** is one level below GRUB — it decides which bootloader the machine hands control to in the first place, before GRUB has even loaded. Most of the time you never touch the EFI tab at all; it exists for the cases the GRUB tab can't reach, like a second Windows install that manages its own boot entry independently.

2.1 "How do I see other operating systems I have installed?"

Straight answer: the GRUB tab, section "Other OS detection." Odyssey does not scan for other systems by default — most people only have Odyssey on their machine, and scanning costs a moment at every boot menu rebuild for nothing. When you do have something else installed (Windows, another Linux, anything), this section turns the scan on in three guided steps, shown as a small tracker at the top of the tab:

1. **Installed** — the tool that does the scanning, `os-prober`, is not on the system yet. The panel shows one button: **INSTALL OS-PROBER**.
2. **Enable scan** — the tool is installed, but GRUB is still told to ignore it (this is the Odyssey default). One button: **ENABLE SCAN**.
3. **Scan** — GRUB will use it. One button, and this is the one you'll use again any time you add or remove another OS later: **SCAN & REGENERATE**.

WHAT	Boot → GRUB tab → "Other OS detection" → click the single button shown (its label changes with the step: INSTALL OS-PROBER, then ENABLE SCAN, then SCAN & REGENERATE) — three clicks total, one password prompt.
WHY	Without this, GRUB only ever lists Odyssey — a second Windows or Linux install on the same disk is invisible at boot, even though it's still there.
RESULT	Every operating system os-prober can find is added to the GRUB menu. Reboot to see it. Re-run SCAN & REGENERATE whenever you install or remove something else.
RISK	Low. os-prober only reads partitions to identify what's on them — it does not modify anything outside GRUB's own menu. On a machine with only Odyssey installed, a scan simply finds nothing new. If detection ever misfires, the DISABLE button next to the tracker turns scanning back off without touching anything else.

Once entries exist, they show up automatically the same way Odyssey's own kernels do — as pickable items in "Default boot entry" below, each with an icon guessed from its type (Windows, Ubuntu, generic Linux, and so on).

2.2 The default boot entry

This is simply **which entry starts automatically when you don't touch the keyboard** at the GRUB menu. The panel shows the current default as a single row — icon, title, its GRUB id — and clicking it opens a list of everything else you can pick instead.

Per-kernel entries (the ones GRUB normally buries under "Advanced options") are grouped under a collapsible header per operating system, so the list you see day to day is short: one line per OS, expandable only if you specifically need to boot an older kernel.

WHAT	Boot → GRUB tab → click the current default entry → pick a different one from the list that opens → APPLY.
WHY	You dual-boot and want the **other** system to start by default, or a kernel update left an old kernel selected and you want the latest one back as default.
RESULT	The chosen entry boots automatically from the next restart on. The change is anchored to that entry's own stable id, so it survives future kernel updates — you don't need to redo this after every update.
RISK	None to your data. Worst case you pick the wrong entry and have to come back and change it again — nothing is destructive.

RELOAD re-reads GRUB's config if you suspect it changed outside the panel; SHOW FULL LIST displays every entry exactly as GRUB itself lists them, read-only, useful mainly to double check nothing unexpected is in there.

2.3 "How do I choose EFI?" — the EFI tab

This question usually means one of two things, and the EFI tab answers both.

MAKING A DIFFERENT BOOTLOADER THE FIRMWARE DEFAULT

If your machine's **firmware** — not GRUB, the UEFI setup screen your motherboard shows before any OS loads — is starting the wrong thing (say, Windows' own bootloader instead of Odyssey's GRUB, or vice versa), that's not a GRUB setting at all. It's the EFI tab.

The tab lists every boot entry the firmware itself knows about, each with a name and its loader path. The one currently starred is the firmware's default.

WHAT	Boot → EFI tab → find the entry you want (e.g. "Odyssey" or "Windows Boot Manager") → click DEFAULT on that row.
WHY	The firmware boot order decides which bootloader gets control first — before GRUB, before any OS. If it's pointed at the wrong one, that other system starts no matter what GRUB is configured to do.
RESULT	The entry moves to the front of the firmware's boot order and starts on every subsequent restart, until you change it again here.
RISK	Low, and always recoverable from this same tab or from your motherboard's own UEFI setup screen (usually Del or F2 at power-on) if the system won't boot the way you expect. Nothing here erases an entry — REGISTER and DEFAULT only reorder or add, never delete.

For a one-time exception — boot something else **just this once**, without changing the standing default — use BOOT NEXT on that row instead of DEFAULT. It affects only the very next restart; after that, the

normal order resumes automatically. A banner at the top of the tab always shows when a one-time boot is pending, with a CANCEL button.

A SYSTEM THAT'S INSTALLED BUT INVISIBLE TO THE FIRMWARE

Occasionally an installer writes its bootloader to disk without properly registering itself with the firmware — the loader exists, but no firmware entry points at it, so it never appears in your motherboard's boot menu at all. The EFI tab detects this automatically and lists such loaders in a separate section, **"ON DISK, NOT IN FIRMWARE,"** each with a REGISTER button.

WHAT	Boot → EFI tab → in "ON DISK, NOT IN FIRMWARE" (only shown if something is found there) → REGISTER on the entry.
WHY	A bootable system that the firmware doesn't know about can only ever be reached indirectly (chainloaded from another bootloader) — registering it gives it a real, direct entry in the boot menu.
RESULT	A new firmware entry pointing at that loader. Note: newly registered entries are placed **first** in the boot order by the firmware itself — if you want your usual system to keep booting by default, click DEFAULT on it afterward.
RISK	Low. This only adds an entry; it never touches the loader files themselves.

If `efibootmgr` — the tool this whole tab relies on — isn't installed yet, the tab shows a single INSTALL EFIBOOTMGR button before anything else; installing it triggers an immediate first read, so the tab is usable right away.

2.4 Kernel parameters

The last section of the GRUB tab is the plain kernel command line — `GRUB_CMDLINE_LINUX_DEFAULT` — and the menu timeout, edited as a normal text field and a number, no GRUB syntax required. SAVE writes them to `/etc/default/grub`; GRUB-MKCONFIG regenerates the actual boot menu from that file afterward, which is the step that makes a saved parameter take effect on the next boot. The panel always re-applies your current default entry choice when it regenerates, so this step never quietly resets which system boots by default.

> **WHEN TO CLICK GRUB-MKCONFIG**
>
> Saving a parameter alone does not change the boot menu on disk — it only updates the source file. If you changed the command line and want it live on the next reboot, click GRUB-MKCONFIG afterward. The SCAN & REGENERATE button in the os-prober section above does the same regeneration, so you never need to click both after an OS scan.

CHAPTER 3

Kernel

The Kernel panel is a graphical front-end to `sysctl` — the kernel's own live tuning interface — plus two related areas that don't go through sysctl: CPU frequency governors and transparent hugepages. Four tabs: **SYSCTL**, **CPU GOVERNOR**, **HUGEPAGES**, **MODULES**.

At the very top, above the tabs, two pieces of information and the preset row: your running kernel version, the active scheduler (`linux-ether` runs BORE — see the Launch Edition presentation document if you're curious why), and three one-click presets — **DESKTOP**, **GAMING**, **SERVER** — that set every sysctl parameter below to a matched, tested combination in one action.

WHAT	Kernel → click DESKTOP, GAMING, or SERVER at the top.
WHY	Tuning fifteen individual sysctl values by hand is a lot to ask of someone who just wants "good defaults for gaming." A preset is exactly that, pre-decided.
RESULT	Every sysctl parameter on the SYSCTL tab jumps to the preset's values, staged and ready — you still need TRY NOW or MAKE PERSISTENT (below) to actually apply them.
RISK	None — picking a preset only fills in the form; nothing is written until you explicitly apply it.

3.1 SYSCTL — the tunables

Parameters are grouped under four headers — **SCHEDULER**, **CPU**, **MEMORY**, **NETWORK**, plus a **SECURITY** group — each with a short label and, where it matters, a plain-language hint above the group.

GROUP	WHAT LIVES THERE
Scheduler	How processes are grouped and prioritized — autogroup, whether a forked child runs before its parent
CPU	Watchdog and timer behavior, and how strict performance-monitoring access is
Memory	Swappiness, how eagerly the page cache is dropped, dirty-page write-back thresholds
Network	TCP Fast Open, backlog queue size, socket buffer limits
Security	`kptr_restrict` and `dmesg_restrict` — whether kernel memory addresses and kernel log messages are hidden from unprivileged users

Two buttons apply your edits, and they are deliberately different:

WHAT	TRY NOW
WHY	You want to test a value without any risk of it surviving a bad reboot.
RESULT	The value is written live via `sysctl` — active immediately, for this session only.
RISK	None beyond the value's own effect — it reverts automatically on reboot regardless of what happens.

WHAT	MAKE PERSISTENT
WHY	You've tried a value (or a whole preset) and want it to survive reboots.
RESULT	The values are written to `/etc/sysctl.d/99-odyssey.conf` **and** applied live in the same action — one click covers both.
RISK	Low — it's still just a text file under `/etc/sysctl.d/`; if a value causes trouble, edit or delete the relevant line (or use the panel again) and re-apply.

RELOAD VALUES re-reads what's currently active on the system, discarding anything you'd typed but not applied — useful after using a preset you decided not to keep, or after another tool changed something underneath the panel.

3.2 CPU governor

The frequency governor decides how aggressively your CPU changes clock speed. A dropdown applies one governor to every core at once (**APPLY TO ALL**); the list below it shows the live governor per core individually, for spotting a core stuck on the wrong setting.

GOVERNOR	BEHAVIOR
performance	Maximum frequency, always — most responsive, most power draw
powersave	Minimum frequency, always — quietest and coolest, least responsive
schedutil	Frequency driven by the scheduler's own view of load — the recommended choice with the BORE scheduler `linux-ether` ships

Only the governors your specific CPU and kernel actually expose appear in the dropdown — the panel never offers a setting that would fail.

3.3 Hugepages

Transparent hugepages (THP) let the kernel use larger memory pages automatically, which can help memory-heavy applications. Two independent settings, each a row of three plain buttons — ALWAYS, MADVISE, NEVER — applied the instant you click:

SETTING	WHAT IT CONTROLS
Transparent hugepages	ALWAYS = on everywhere (best throughput); MADVISE = only for applications that explicitly ask for it (the recommended middle ground for a desktop or gaming machine); NEVER = fully off
Defrag	How aggressively the kernel compacts memory to **form** hugepages when they're requested — same three options, independent of the setting above

The current live value is always shown above the buttons, so you can tell at a glance whether anything actually needs changing.

3.4 Modules

A live, filterable list of every kernel module currently loaded — type in the filter box to narrow it, REFRESH to re-read. Read-only: this tab exists for inspection (checking a driver is loaded, confirming something unloaded cleanly), not for loading or removing modules from the panel.

CHAPTER 4

Memory & Swap

Three sections: a read-only summary of your system memory, swappiness, and ZRAM.

4.1 System memory

Just a readout — total RAM, in use, available — for context before you touch anything below. No controls here.

4.2 Swappiness

Swappiness is a single number from 0 to 100 that tells the kernel how eagerly to move memory out to swap versus keeping it in RAM under pressure. 0 means "only swap when there is truly no other choice"; 100 means "swap aggressively, well before RAM is actually full." The kernel ships with 60; the panel's own hint suggests 10–20 as a better desktop default, and that's exactly what the DESKTOP preset in the Kernel panel sets it to as well — the two panels agree with each other.

A slider sets the value, with the number shown live beside it, and the same TRY NOW / MAKE PERSISTENT split as the Kernel panel: APPLY NOW changes it live for this session, MAKE PERSISTENT writes it to a config file so it survives reboot too.

WHAT	Memory & Swap → drag the Swappiness slider → APPLY NOW to test, MAKE PERSISTENT once you're happy with it.
WHY	A lower value keeps more of what you're actively using in fast RAM instead of on slower swap, which is usually what you want on a desktop with enough memory.
RESULT	The kernel changes how aggressively it swaps immediately (APPLY NOW) or from every future boot on as well (MAKE PERSISTENT).
RISK	Very low, and always adjustable again the same way. An extremely low value on a machine genuinely short on RAM can make the system more prone to running out of memory under heavy load instead of swapping gracefully — if that happens, raise it back up.

4.3 ZRAM — compressed RAM swap

ZRAM is a swap device that lives inside RAM itself, compressed — meaningfully faster than swapping to a disk, because there's no disk involved at all. Size is set as a percentage of your total usable RAM, not a fixed number, so the same setting scales sensibly across different machines; a live readout next to the size field shows exactly how many GiB that percentage works out to on your system.

WHAT	Memory & Swap → set the size percentage and pick a compression algorithm (zstd, lz4, lzo, or deflate) → ENABLE ZRAM.
WHY	Compressed RAM swap absorbs memory pressure with far less of a performance hit than swapping to disk — useful on any machine, and especially valuable on one with limited RAM.
RESULT	A `zram0` device is created and activated at the chosen size and algorithm; the status line above turns to reflect it's active.
RISK	Low. zstd (the default) balances compression ratio and CPU cost well for most machines; lz4 trades some compression for speed if CPU time is the tighter resource. DISABLE turns it off cleanly at any time.

DISABLE removes the ZRAM device. The status line at the top of the section always reflects the live state, so you never have to guess whether it's currently on.

CHAPTER 5

Firewall

The Firewall panel is covered in full in the **Security and Hardening Guide** — the ruleset explained line by line, all 21 presets with their individual risks, the raw-config editor, and the command-line equivalents. This chapter is the short version.

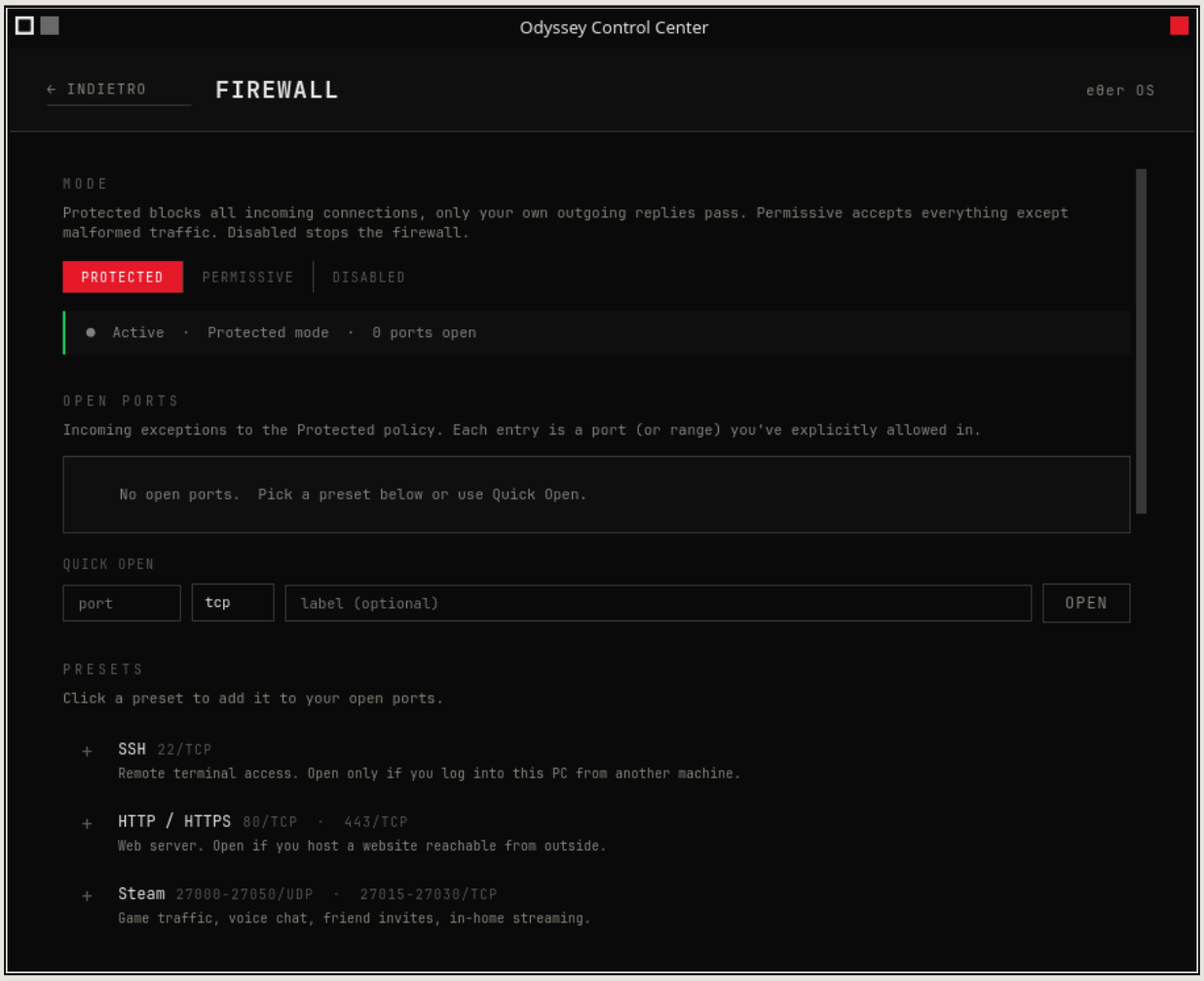

FIG · The Firewall panel — PROTECTED mode, zero ports open.

Three modes, chosen at the top: **PROTECTED** (the default — everything unsolicited is dropped), **PERMISSIVE** (accepts inbound traffic, kept strictly as a diagnostic setting — "is the firewall what's breaking this?"), and **DISABLED**.

The everyday task here is opening a port for something you're running — a game server, a media server, a dev server reachable from your phone. Click a tile under **PRESETS** (eight common ones, thirteen more under **MORE PRESETS**), or use **QUICK OPEN** for anything not listed by giving a port and protocol directly. Nothing takes effect until **SAVE & APPLY** — changes stage first, so you can review exactly what you're about to open before committing.

FOR THE FULL PICTURE

Which preset to trust on a public network versus your own LAN, what each one actually exposes, and how to read and hand-edit the raw `nftables.conf` — all in Chapter 2 of the Security and Hardening Guide, `/usr/share/doc/odyssey/` on your installed system.

CHAPTER 6

AppArmor

Same treatment as Firewall: the **Security and Hardening Guide** has the full chapter — what mandatory access control actually is, the confinement workflow end to end, and a worked example. Here's what you need to get started.

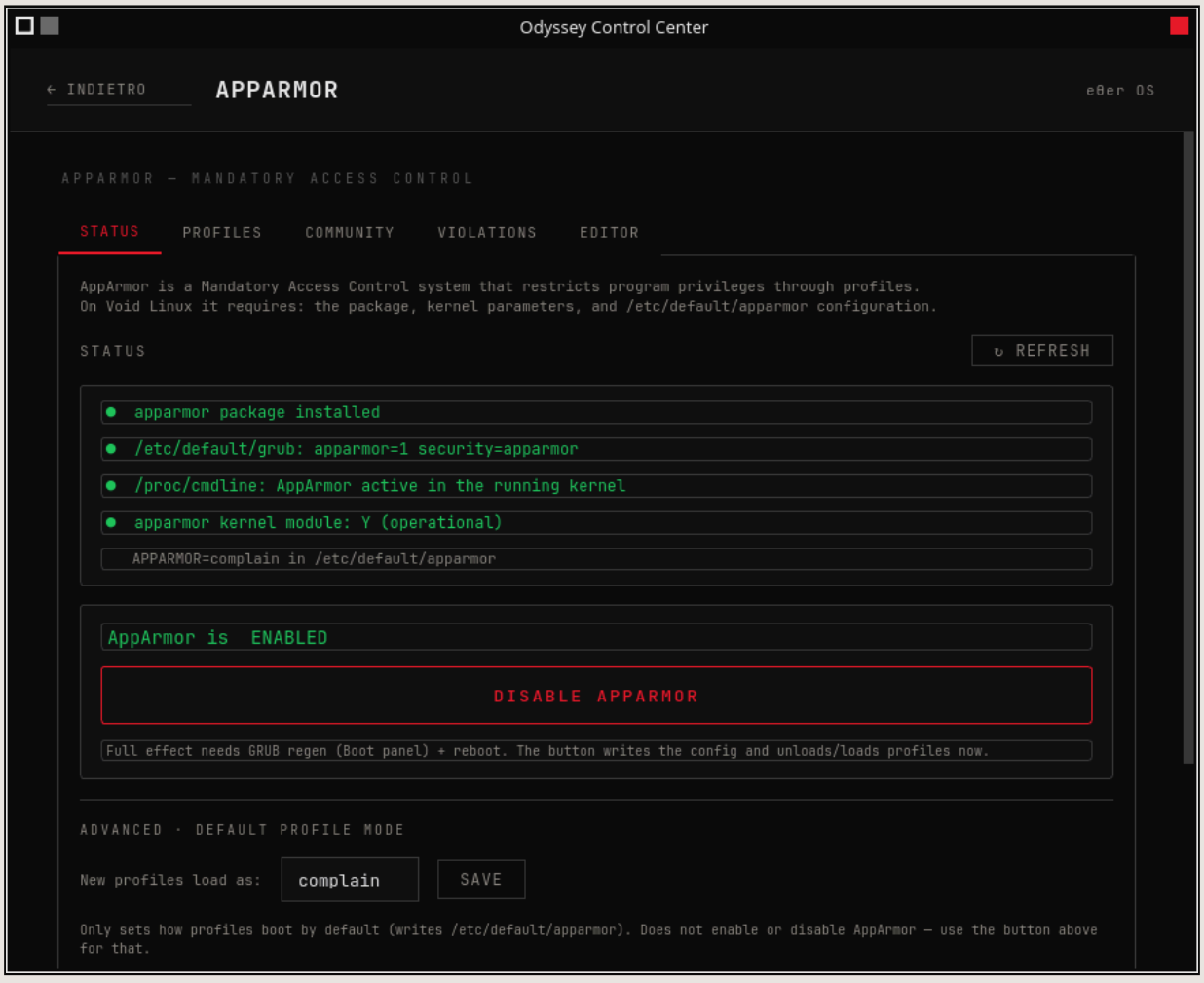

FIG · The AppArmor panel's STATUS tab — the framework verified end to end.

Five tabs. **STATUS** verifies the framework is actually active (package installed, kernel parameters set, kernel actually running with them, module loaded) and lets you set whether new profiles start in complain or enforce mode. **PROFILES** lists everything currently loaded, each with three switches — enforce, complain, disable. **COMMUNITY** downloads ready-made profiles for common desktop applications (browsers, mail, messengers) straight from the official upstream repository. **VIOLATIONS** shows what a profile has logged. **EDITOR** opens a profile's file directly, and — the feature worth knowing about — its **ADD SUGGESTED RULE** button turns a logged violation into the matching profile rule with one click, no hand-editing required.

WHAT	AppArmor → COMMUNITY → install a profile for an app you use daily → leave it running normally for a few days → VIOLATIONS → if anything legitimate got logged, double-click it and ADD SUGGESTED RULE → PROFILES → switch it to enforce.
WHY	This is the entire confinement workflow in one line: observe first, fix what's missing, only then start blocking.
RESULT	That application is now confined by the kernel to exactly what its profile allows — a compromise of the app no longer means a compromise of everything you can reach.
RISK	Low, by design — nothing blocks until you explicitly switch to enforce, and dropping back to complain is the same one click if anything misbehaves.

FOR THE FULL PICTURE

The complete lifecycle, reading raw denial logs, and a real worked example (confining `ping` from complain to enforce, start to finish) are in Chapter 3 of the Security and Hardening Guide.

CHAPTER 7

Privacy

Two independent tools, each with its own install → enable → running flow, and a bridge between them if you want both together: **SearXNG** (private web search) and **Tor** (network anonymity).

7.1 SearXNG — private search

SearXNG is a metasearch engine that runs locally on your own machine. When you search, it's SearXNG that queries Google, Bing and the rest on your behalf and hands you the combined results — the search engines see SearXNG making the request, never you directly.

A three-step tracker — **Install**, **Enable service**, **Running** — and a status line above it tell you exactly where things stand, with one primary button that always matches the next step.

WHAT	Privacy → SearXNG section → click the button shown (INSTALL SEARXNG, then ENABLE SERVICE) → once running, OPEN IN BROWSER.
WHY	Every other search engine builds a profile from your queries over time; a local metasearch instance means only you ever see the unaggregated query.
RESULT	A private search engine running on your own machine, reachable in your browser, that queries the big engines on your behalf without exposing you to them directly.
RISK	Low. It's a local web service like any other — RESTART if it misbehaves, STOP SERVICE to pause it without removing it, UNINSTALL to remove it entirely.

7.2 Tor — network anonymity

Tor routes your traffic through three relays before it reaches its destination, so the site you're visiting sees the exit relay's address, never yours. Same pattern: a status line, INSTALL TOR, ENABLE SERVICE.

A dedicated **TEST** button is worth using right after enabling it — it compares your real IP address against the IP Tor is presenting, side by side, so you can confirm it's actually working rather than taking it on faith.

WHAT	Privacy → Tor section → INSTALL TOR → ENABLE SERVICE → TEST (real IP vs Tor IP) to confirm.
WHY	Routing sensitive browsing through Tor hides your address from the sites you visit — and the TEST button gives you actual proof it engaged, not just a status light you have to trust.
RESULT	A running Tor service other applications can route through; the TEST comparison shows two different addresses if everything is working.
RISK	Traffic through Tor is meaningfully slower than a direct connection — that's the trade for the anonymity. STOP SERVICE turns it off without uninstalling; UNINSTALL removes it fully.

7.3 Routing SearXNG’s own queries through Tor

The two combine: an inline notice with a **ROUTE VIA TOR** button appears once both are installed, letting SearXNG’s own outgoing queries (the ones it makes to Google, Bing, and so on, on your behalf) go through Tor as well — a fully private search stack, not just a private front end.

WHAT	Privacy → the “Route SearXNG’s outgoing queries through Tor” notice → ROUTE VIA TOR → restart SearXNG (RESTART button in the SearXNG section) for it to take effect.
WHY	Without this, SearXNG itself still queries the big search engines directly — private from your side, but SearXNG’s own requests remain visible to whoever it’s asking. Routing those through Tor closes that gap.
RESULT	SearXNG’s outbound queries, not just your queries to SearXNG, go through Tor.
RISK	Search results may load noticeably slower, for the same reason any Tor-routed traffic is slower. REMOVE TOR ROUTING reverts it, again followed by a SearXNG restart.

CHAPTER 8

Containers

Container management end to end — engine setup, running containers, a curated catalog to start from, raw commands for anything not in it, and disk usage.

8.1 Engine

Odyssey recommends **Podman** — daemonless and rootless, which means containers run without a permanently running background service and, by default, without root privileges. **Docker** is supported too if it's already what you use. A two-step tracker (**Engine installed**, **Ready to use**) and one contextual button walk you through getting either one running.

8.2 Containers, and rootless vs. rootful

The main list shows every container you have, with name, image, ports, and per-row actions (start, stop, logs, remove). A **ROOTLESS / ROOTFUL** toggle at the top switches which context you're looking at and managing — the two are entirely separate namespaces of containers, not a filter on one list.

> WHICH ONE SHOULD I USE
>
> Rootless is the safer default and what the panel points you toward: a container escaping confinement in rootless mode still can't do anything your own user account couldn't already do. Rootful exists for the cases that specifically need it — certain low-numbered network ports, some hardware access — and should be the exception, not the habit.

8.3 Available containers — the curated catalog

A ready list of common, rootless-friendly containers, so the panel is never an empty box waiting for you to know Docker Hub image names by heart. **GET** downloads an image; **RUN** starts it. Once it's running, it moves up into the main Containers list above like anything else.

8.4 Run a container

For anything not in the catalog: a single field that takes a full `podman run` (or `docker run`) command and executes it exactly as typed — nothing added, nothing hidden, nothing substituted.

WHAT	Containers → “Run a container” → paste your full run command (e.g. `run -d --name myapp -p 8080:80 nginx:latest`) → RUN.
WHY	The curated catalog covers common cases; this is the escape hatch for anything specific — a particular image, particular flags, particular ports.
RESULT	The container starts exactly as the command specifies, and appears in the main Containers list once running.
RISK	Whatever the command itself does — the panel is a faithful pass-through, not a sandbox around your input. Treat it with the same care you’d give the same command in a terminal.

8.5 Images and disk usage

A separate list of downloaded images (independent of which containers are currently running from them), with REFRESH and **PRUNE UNUSED** to reclaim space from images nothing references anymore. Below it, three running totals — containers, images, total size — so you always know what containers are costing you on disk without opening a terminal.

CHAPTER 9

Gaming Mode

One master toggle at the top, a live status row beneath it, and four tabs for everything gaming-related: **PACKAGES**, **GAMEMODE**, **MANGOHUD**, **PROTON**.

9.1 The main toggle

ACTIVATE tunes the scheduler, CPU governor, swappiness and transparent hugepages for gaming in one action, and makes the change persistent in `/etc/sysctl.d/` — the same underlying mechanism as the Kernel and Memory panels, bundled into one button instead of four separate visits. The status row shows the four values live — governor, swappiness, THP, and whether the `gamemode` service is running — so you can confirm at a glance that activation actually took.

WHAT	Gaming Mode → ACTIVATE.
WHY	Manually visiting Kernel and Memory to set a GAMING-appropriate governor, swappiness and THP value is exactly the kind of multi-step chore a dedicated mode should remove.
RESULT	Scheduler, governor, swappiness and THP all switch to gaming-appropriate values, applied immediately and made persistent.
RISK	Low, and reversible — the same values you'd otherwise set by hand in Kernel/Memory, just bundled. Toggling off (or manually resetting those values in Kernel/Memory) restores general-purpose behavior.

9.2 Packages, and Steam specifically

A checklist of the packages generally needed for gaming, with CHECK to see what's missing and INSTALL MISSING to fill the gaps in one action.

Steam gets its own dedicated flow below it, because it needs more than a plain install — non-free and multilib repositories, plus 32-bit graphics libraries matching your GPU. **SET UP STEAM** handles the whole chain, with your confirmation, rather than leaving you to work out which repositories and libraries apply to your hardware.

WHAT	Gaming Mode → PACKAGES tab → SET UP STEAM.
WHY	Steam is proprietary and 32-bit at its core, and Void doesn't enable the repositories or libraries it needs by default — getting this wrong by hand is a common first-time stumbling block on any Void-based system.
RESULT	The required repositories, Steam itself, and matching 32-bit graphics libraries for your GPU, all installed and ready.
RISK	Low — this only adds repositories and packages; nothing existing is removed or reconfigured.

9.3 GameMode and MangoHud

GameMode is Feral Interactive's per-game performance daemon — games launched through it request a temporary performance boost (CPU governor, I/O priority) for just their own process, automatically handed back when the game closes. **MangoHud** is the on-screen overlay showing FPS, frame time, CPU/GPU load and temperatures during play. Both tabs offer install/check controls and, where relevant, quick configuration for the overlay's look.

9.4 Proton

Compatibility-layer management for running Windows games through Steam — checking what's installed and available, and installing additional Proton versions (including community builds like Proton-GE) when a specific game needs one Valve doesn't ship by default.

CHAPTER 10

Telemetry

Despite the name, this panel is not a setting you configure — it's a live monitor. Odyssey doesn't collect or send anything about you (there is nothing to switch off here); what this panel does instead is show you, in real time, every outbound network connection your **own** system is making, so "what is my machine talking to right now" is a question you can answer by looking rather than by trusting a claim. It calls itself **Reverse Telemetry** for exactly that reason — turning the watching around, onto the system itself, on your behalf.

A status bar across the top gives the running totals — total connections, how many are external, how many are flagged suspicious, current download and upload rates — plus **PAUSE** and **START** to freeze or resume the live view without losing what's already been captured. Four tabs beneath it: **CONNECTIONS**, **TRAFFIC**, **DNS**, **HISTORY**.

10.1 Connections

A live table, one row per active connection: process name, PID, protocol, remote address, port, and the service that port is known for (443 shown as HTTPS, 22 as SSH, and so on). Sortable by any column — click a header to sort by remote address and see everything talking to one host together, for instance.

Connections from processes that have no ordinary reason to be reaching the network on their own — a bare shell, a scripting interpreter running standalone — are flagged automatically, so an anomaly stands out from the normal traffic of your browser and system services instead of being buried in it.

WHAT	Telemetry → CONNECTIONS tab → watch what's listed, or sort by PROCESS or REMOTE IP to investigate something specific.
WHY	It's the direct, factual answer to "is anything on this machine phoning home that shouldn't be" — read from the kernel's own connection table, not from any vendor's claim about their own software.
RESULT	A live, accurate picture of every outbound connection, updated continuously while the panel is open and not paused.
RISK	None — entirely read-only.

10.2 Traffic

Per-interface throughput — download and upload rate right now, plus running totals since the panel started watching — one row per network interface on the machine.

10.3 DNS

A live feed of domain name lookups as they happen — which hostnames your system is actually resolving, in real time, which is often a clearer signal than the raw IP addresses in the Connections tab (a hostname tells you **what**, an IP address on its own often doesn't).

10.4 History

Everything the session has captured, kept as a running log you can scroll back through — useful for catching something that happened briefly and is no longer an active connection by the time you think to look.

CHAPTER 11

About

A single, exhaustive diagnostic sheet of the machine, organized into three sections — **System**, **Security**, **Hardware** — refreshed every time you open the tab, so it always reflects what changed in other panels without needing to reopen the Control Center.

SECTION	WHAT'S IN IT
System	Kernel version and architecture, init system, hostname, uptime, installed package count and xbps version, current desktop session
Security	Firewall status, AppArmor status, the privilege-escalation mechanism in use (polkit) — each with a colored status indicator, not just a label
Hardware	GPU driver, Mesa version, Vulkan support, motherboard model, audio

11.1 Copy report

A single button, **COPY REPORT**, puts the entire sheet on your clipboard as plain text — every section, every value — ready to paste directly into a forum post or a bug tracker entry.

WHAT	About → COPY REPORT → paste into your bug report or forum post.
WHY	Bug reports are only as useful as the system details attached to them, and typing out kernel version, driver, AppArmor state and a dozen other fields by hand is exactly the kind of friction that makes people skip it.
RESULT	A complete, correctly formatted system report on your clipboard, ready to paste — the same information the maintainer would otherwise have to ask you for individually.
RISK	None to your system. The report is plain diagnostic information (versions, states, hardware model) — it doesn't include personal files or credentials — but as with any diagnostic dump, read it over before posting somewhere public if you're unsure what it contains.

CHAPTER 12

NVIDIA driver

This panel only appears if NVIDIA hardware is actually detected on the machine — nothing to configure otherwise. It carries a visible **BETA** badge, and an honest note right at the top explains why: the driver itself is proprietary, packaged by Void from NVIDIA's own official installer with a pinned checksum, but its actual contents can't be audited the way the rest of Odyssey can be. Same principle as everywhere else in the project, stated plainly here too: don't trust NVIDIA on our word — we can't verify their driver either, only that what you're installing matches what NVIDIA published.

12.1 Detected hardware and branch selection

The panel identifies your specific GPU against a built-in database and picks the correct driver branch for it automatically — you don't need to know which branch your card needs.

BRANCH	COVERS
nvidia (current)	Turing architecture and newer — most GPUs from roughly 2018 onward
nvidia580	legacy branch — Maxwell / Pascal / Volta
nvidia470	legacy branch — Kepler
nvidia390	legacy branch — Fermi; the panel itself recommends nouveau instead on this old a card
nouveau	the open-source driver — used automatically when no proprietary branch applies, or when a detected card isn't in the database yet

12.2 Installing the driver

One checkbox — **install 32-bit libraries**, checked by default, needed for Steam and older 32-bit games — and one button.

WHAT	NVIDIA driver → confirm the 32-bit libraries checkbox matches what you need → INSTALL DRIVER.
WHY	nouveau (the open-source default) works but doesn't expose the GPU's full performance, particularly for gaming — the proprietary driver is the difference-maker there.
RESULT	A tracked pipeline runs through seven visible steps — Repos, Sync, Packages, Blacklist, DKMS build, Verify, Initramfs — shown as a progress tracker so you can see exactly where things stand rather than watching a single spinner. When it finishes, reboot for the driver to take over.
RISK	Moderate — this is a kernel-level graphics driver, the one category of change with real potential for a black screen on reboot if something about your specific hardware doesn't cooperate. The panel prints the recovery path right there, visible even before you install: if the screen stays black after reboot, switch to a text console with Ctrl+Alt+F2, log in, and run `odyssey-nvidia --revert`.

```
# the exact rescue command shown in the panel
$ odyssey-nvidia -revert
```

REVERT TO NOUVEAU does the same rollback from inside the panel itself, for when the desktop is working well enough to reach it — the terminal command above is strictly the fallback for when it isn't.

12.3 PRIME render offload — hybrid laptops

This section appears automatically only on laptops with both an integrated GPU and an NVIDIA GPU (an "Optimus" hybrid setup). On these machines the integrated GPU normally drives the whole desktop, and the NVIDIA GPU sits idle until something specifically asks to render on it — which is exactly what this section is for.

For a Steam game, set its launch options to `odyssey-prime %command%` and Steam will hand that one game to the NVIDIA GPU automatically. For anything else, a field right in the panel does the same thing directly:

WHAT	NVIDIA driver → PRIME render offload → type a program or game command into the field → RUN ON NVIDIA GPU (or, in Steam, set launch options to `odyssey-prime %command%`).
WHY	Running everything on the NVIDIA GPU all the time defeats the point of a hybrid laptop — better battery life from the integrated GPU the rest of the time. PRIME offload gives you the powerful GPU only for the one application that actually needs it.
RESULT	That specific program renders on the NVIDIA GPU while everything else on the desktop keeps running on the integrated one.
RISK	Low. Worst case a program launched this way behaves no differently than launching it normally.

One more control shows up only on hybrid machines: a checkbox for **external displays on NVIDIA ports**. Some laptops wire their HDMI or USB-C video output directly to the NVIDIA GPU, and the safe hybrid profile Odyssey uses by default can leave those ports dark. If an external monitor plugged into your laptop stays blank, this is the checkbox that fixes it.