ODYSSEY LINUX · 参考文档

Odyssey Control Center — 实用指南

每个面板，逐节讲解。它做什么、改变了什么，
以及该点哪里——为从未用过它的人而写。

Launch Edition

Don’t trust me. Verify me.

— nobody

Odyssey Linux — Odyssey Control Center：实用指南
Launch Edition。涵盖 Launch Edition ISO 所附带的 Control Center——从 Boot 到 About 共十个完整面板，外加一章关于 NVIDIA 驱动模块的占位章节（仅在检测到 NVIDIA 硬件时可见，等待拥有该硬件 的用户提供截图）。

Copyright © 2026 the Odyssey Linux project.
以 Creative Commons Attribution-ShareAlike 4.0 International (CC BY-SA 4.0) 许可发布。

本指南是 Security and Hardening Guide 的实用补充，后者以完整的 技术深度介绍了 Firewall 和 AppArmor 面板——本文提供的是逐步的 图解指南，其余内容则给出交叉参考。

本文描述的每一个界面、按钮和行为，均取自 Control Center 的真实 源代码和真实截图，而非凭记忆写就。

按钮标签、面板/标签页名称、文件路径和命令均保留英文原文，因为 它们就是程序的真实界面：在 Control Center 中寻找翻译后的名称是 找不到的。

目录

1 Control Center 是什么 1
1.1 你会在每个面板中看到的模式 2
2 Boot 3
2.1 图解指南——找到其他操作系统并设置默认项 3
2.2 图解指南——选择默认启动项 5
2.3 图解指南——EFI 标签页 7
2.4 内核参数 9
2.5 Advanced 9
3 Kernel 11
3.1 图解指南——预设 11
3.2 图解指南——SYSCTL，逐组讲解 12
3.3 图解指南——CPU governor 14
3.4 图解指南——hugepages 15
3.5 图解指南——modules 16
3.6 Advanced 17
4 Memory & Swap 19
4.1 图解指南 19
4.2 Advanced 20
5 Firewall 22
5.1 图解指南 22
5.2 Advanced 23
6 AppArmor 24
6.1 图解指南——STATUS 24
6.2 图解指南——PROFILES 25
6.3 图解指南——COMMUNITY 26
6.4 图解指南——VIOLATIONS 27
6.5 图解指南——EDITOR 28
6.6 Advanced 29
7 Privacy 31
7.1 图解指南——SearXNG，逐步讲解 31
7.2 图解指南——Tor，逐步讲解 33
7.3 图解指南——将两者结合起来 36
7.4 Advanced 37
8 Containers 39
8.1 图解指南——引擎 39
8.2 图解指南——rootless vs. rootful，以及空状态 41
8.3 图解指南——精选目录，从头到尾 41
8.4 图解指南——容器列表及其操作 45
8.5 图解指南——运行一个容器，并打开它 46
8.6 磁盘占用 47
8.7 Advanced 47
9 Gaming Mode 48
9.1 图解指南——主开关 48
9.2 图解指南——Packages 与 Steam 49
9.3 GameMode 与 MangoHud 51
9.4 Proton 52
9.5 Advanced 52
10 Telemetry 54
10.1 图解指南 54
10.2 Advanced 56
11 About 57

11.1 图解指南 57
11.2 Copy report 58
11.3 Advanced 58
12 NVIDIA driver 60

CHAPTER 1

Control Center 是什么

Odyssey Control Center 是十个独立的小程序，共用同一个启动器——每个 系统区域对应一个：Boot、Kernel、Memory & Swap、Firewall、AppArmor、 Privacy、Containers、Gaming Mode、Telemetry、About，以及（在 NVIDIA 硬件上）NVIDIA 驱动模块。

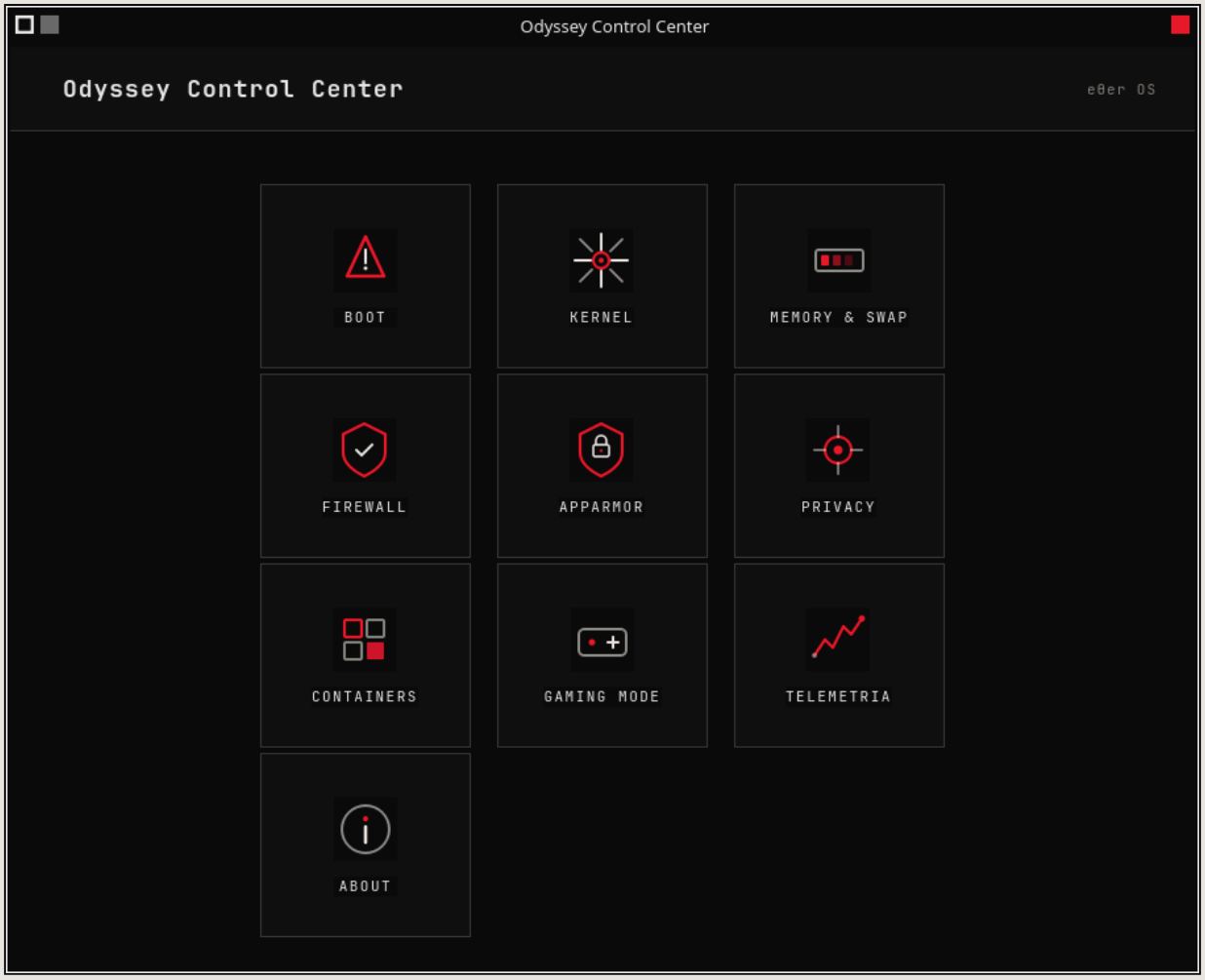

FIG · Control Center 主屏幕——每个面板对应一个图块。

每个面板都是覆盖在真实 Linux 机制之上的一层薄薄的图形界面：GRUB 的 配置、内核自身的 sysctl 接口、nftables、podman。面板从不凭空 制造自己的私有状态——它读取真实的文件和命令，展示给你看，并写入你 所要求的内容，不多不少。它所做的一切都不是隐藏的，而且它所做的一切 也都可以手动完成，从终端出发，用本指南在每个按钮旁展示的那些精确 命令。

这是一份实用指南，而不是每个选项的详尽参考——它回答的是「我该点哪 里，点了会发生什么」。每一章都遵循同样的两段式结构：先是一份**图解 指南**，一张截图接一张截图，一个按钮接一个按钮，写给从未打开过 Control Center 的人——然后是一个以黑色横幅标记的 **ADVANCED** 小节， 深入细节：确切的文件路径、每个按钮在底层执行的真实命令，以及如果你 想完全跳过面板、该从终端做什么。你可以只读图解指南，也可以如果已经熟悉 Linux 就直接跳到 Advanced——两部分各自独立完整。

对于两个真正涉及安全问题的面板——Firewall 和 AppArmor——深入的参考 资料是 **Security and Hardening Guide**；这里你会找到图解指南、要点， 以及一个交叉参考。

1.1 你会在每个面板中看到的模式

有几条惯例在全部十个面板中反复出现——学会一次，处处适用。

密码提示，每个面板只出现一次

大多数面板需要读取或写入属于 root 的文件——GRUB 配置、sysctl、防火墙 规则集。面板第一次需要这样做时，你会收到一个认证对话框。面板的设计 会把读取操作打包在一起：例如打开 Boot 会在一次请求中同时读取 GRUB 配置**和** EFI 固件状态，所以你只需输入一次密码，而不是两次。在同一 会话中重新打开一个面板不会再次询问；随后的操作（应用更改、重新加载） 会在面板自己的日志中显示各自的进度。

步骤追踪器

那些在某样东西变得可用之前引导你完成安装的面板——Boot 中的 os-prober、Privacy 中的 SearXNG 和 Tor、Containers 中的容器引擎——会 在顶部显示一个小型编号追踪器：一排步骤，当前步骤高亮显示，只有一个 操作按钮，它始终执行**下一步**，无论那是什么。你永远不需要猜测该先点 哪里。

底部的日志

每个面板都以一个可滚动的小日志结束。这不是装饰——它是实际执行过的 确切命令、其输出，以及一个彩色的结果判定（绿色表示成功，红色表示 失败）。如果某个面板做了你不理解的事情，日志就是真正答案所在之处。

TRY VS. MAKE PERSISTENT

一对反复出现的按钮，尤其在 Kernel 和 Memory 中：**TRY NOW**（或 **APPLY NOW**）会立即改变正在运行的系统，如果你不再做任何操作，下次 重启时会恢复原状。**MAKE PERSISTENT** 则把同样的值写入 `/etc/` 下的配置文件，使其也能在重启后保留。这种区分是刻意设计的——尝试一个值 不需要任何代价，而没有什么是永久的，除非你说了两次。

CHAPTER 2

Boot

在任何以 UEFI 模式启动的机器上，Boot 面板都有两个标签页：**GRUB** （启动菜单本身）和 **EFI**（其下方的固件层）。在只有 BIOS 的老旧机器 上，只存在 GRUB，因为下方的 EFI 层根本不存在。

> **两个层级，而非一个**
>
> **GRUB** 是你在屏幕上看到的启动菜单——它在 GRUB 本身已经运行起来 之后，决定哪个操作系统启动。**EFI/UEFI 固件**位于 GRUB 之下的一层 ——它首先决定机器把控制权交给哪个引导加载程序，甚至早于 GRUB 被 加载之前。大多数人从不触碰 EFI 标签页；它是为 GRUB 无法触及的 情况而存在的，比如一个独立管理自己启动项的第二个 Windows 安装。

2.1 图解指南—找到其他操作系统并设置默认项

GRUB 标签页会打开在 **Other OS detection** 小节。Odyssey 默认不扫描 其他系统——大多数安装的磁盘上只有 Odyssey，而每次重新生成启动菜单 时都进行扫描只会白白浪费时间。启用它是一个引导式的三步流程，以一个 小型编号追踪器的形式呈现。

1. **`Installed`**

   扫描工具 os-prober 还没有安装在系统上。只有 一个按钮：**INSTALL OS-PROBER**。

2. **`Enable scan`**

   工具已安装，但 GRUB 仍被设置为忽略它——这 是 Odyssey 的默认状态。只有一个按钮：**ENABLE SCAN**。

3. **`Scan`**

   GRUB 现在会使用它了。只有一个按钮，也是你未来每次 添加或移除其他系统时会再次用到的那个：**SCAN & REGENERATE**。

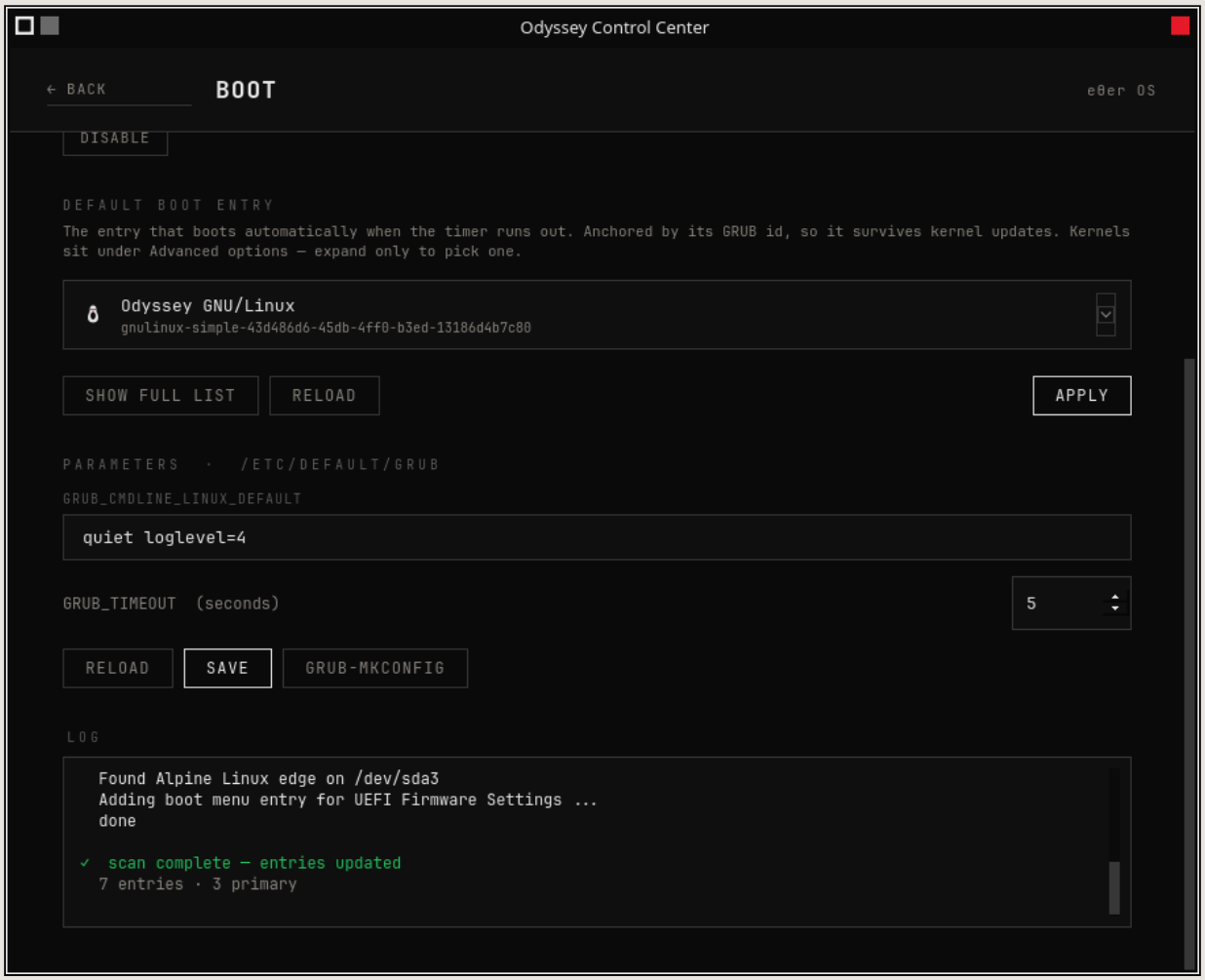

FIG · 三个步骤全部变绿——os-prober 已安装，扫描已启用，一切就绪。唯一 的操作按钮始终对应当前步骤。

依次点击这些步骤（每一步只有在需要时才会重新要求认证），然后运行 扫描。在你离开这个小节之前，面板会准确显示它找到了什么：

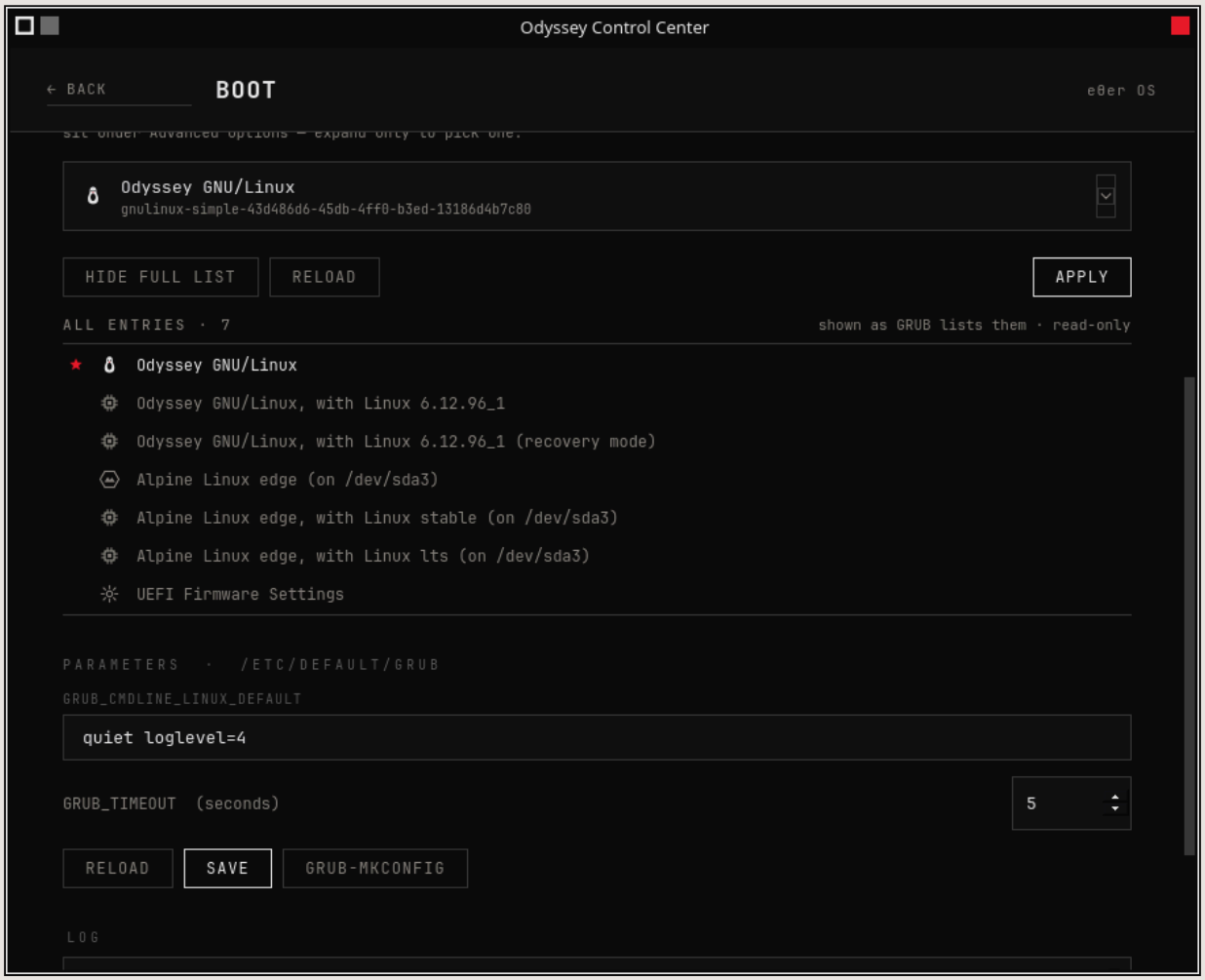

FIG · 扫描完成：在 /dev/sda3 上找到 Alpine Linux，与 Odyssey 自身的 条目以及 UEFI Firmware Settings 条目并列—共 7 个条目，3 个 主要条目。

日志用简单的话确认了这一点——**「scan complete — entries updated · 7 entries · 3 primary」**。现在 os-prober 找到的任何内容都会自动出现 在下方小节中，准备好被选中。

2.2 图解指南—选择默认启动项

滚动到 **Default boot entry**。它以一行的形式显示当前的默认项；点击 它会打开一个下拉菜单，列出你可以选择的所有其他选项。

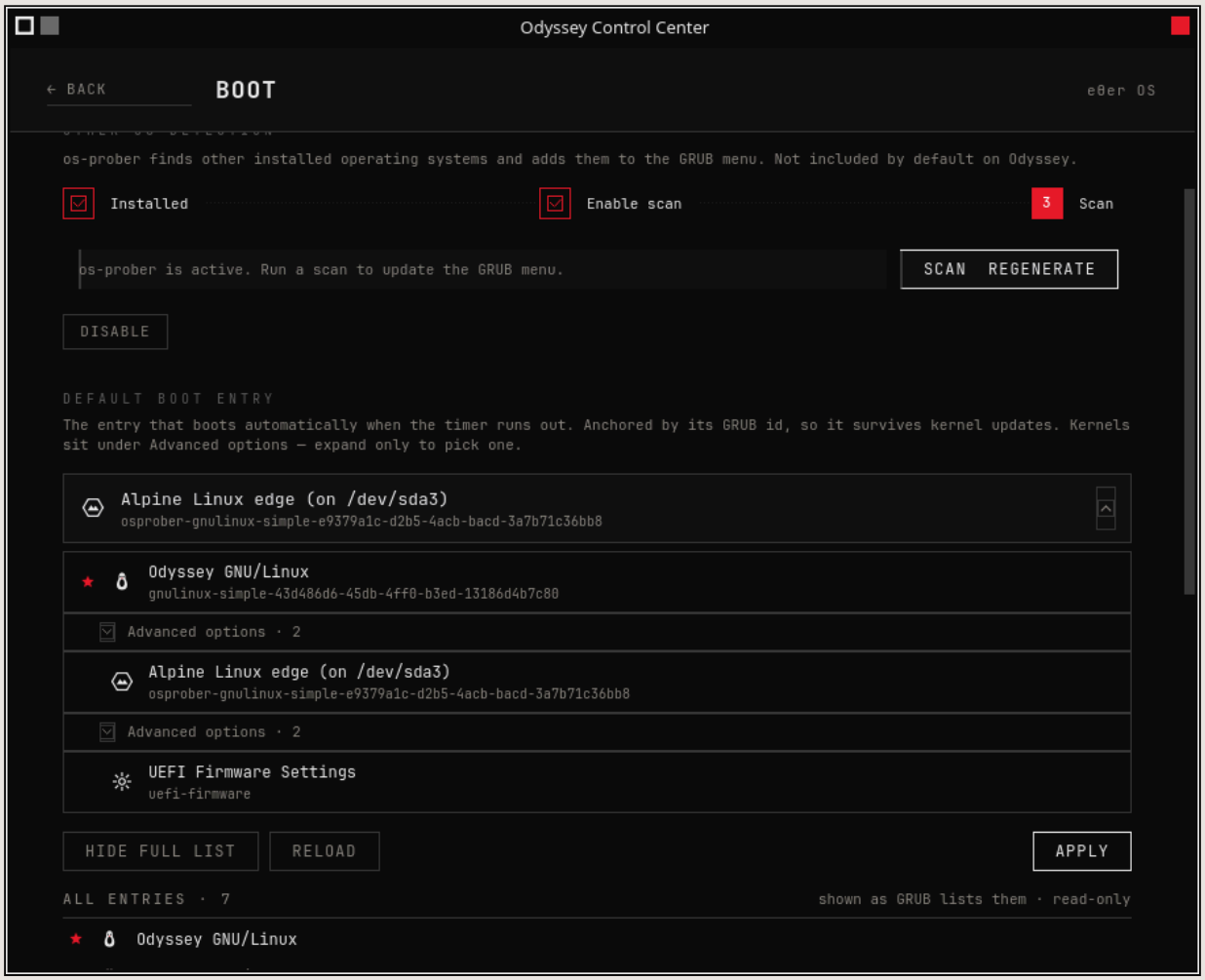

FIG · 打开的下拉菜单，Alpine Linux 被选中为待定选项—尚未应用。

从列表中选择一个条目会**将选择置于待定状态**——它尚未生效。你可以看出 它只是待定的，因为方框仍然显示新的选择，但还没有任何东西被写入。 确认它的按钮是右侧的 **APPLY**：

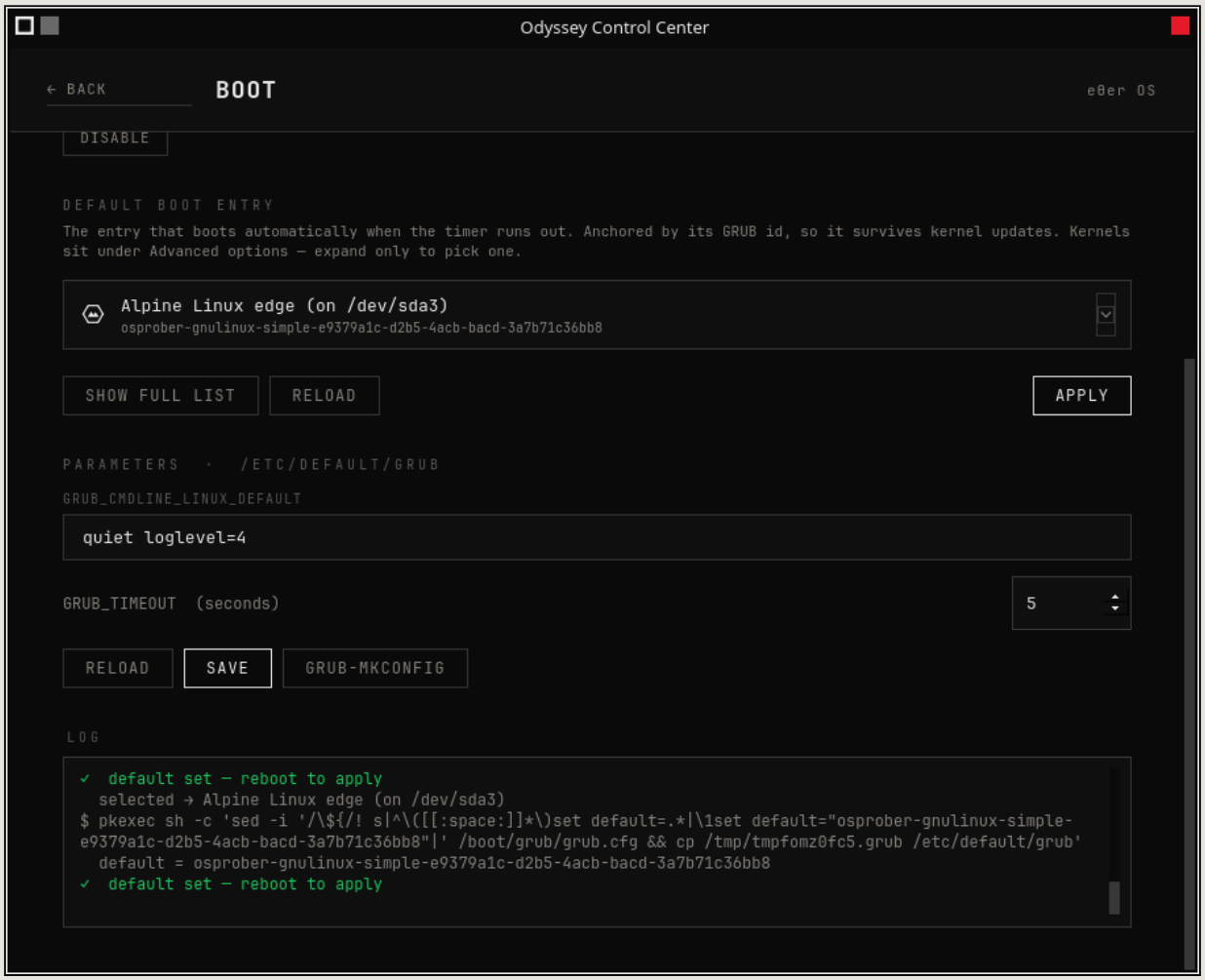

FIG · 点击 APPLY 之后：日志确认「default set — reboot to apply」， 选择器现在显示 Alpine Linux 为当前生效的默认项。

WHAT	Boot → GRUB 标签页 → 点击当前的默认启动项 → 从下拉菜单中 选择一个不同的项 → 点击 **APPLY**。
WHY	你在使用双系统，希望另一个系统自动启动，或者一次内核更新 让某个旧内核成了当前选中项，而你想切回最新的那个。
RESULT	被选中的条目从下次重启起自动启动。这个更改被锚定在该 条目的稳定 id 上，因此可以在未来的内核更新中保持不变。
RISK	对你的数据没有任何风险——最坏的情况是选错了条目，回来 再改一次就好。

SHOW FULL LIST 会按照 GRUB 自身列出条目的方式精确显示每一项—— 只读，并按分组显示：单个内核的变体会归入一个可展开的「Advanced options」行下，而不是塞满主列表。如果你怀疑配置在面板之外发生了变化，**RELOAD** 会重新读取 GRUB 的配置。

2.3 图解指南—EFI 标签页

点击顶部的 **EFI** 标签页。这是固件层——即在任何操作系统开始运行 之前，你的主板自身的启动菜单所显示的内容。

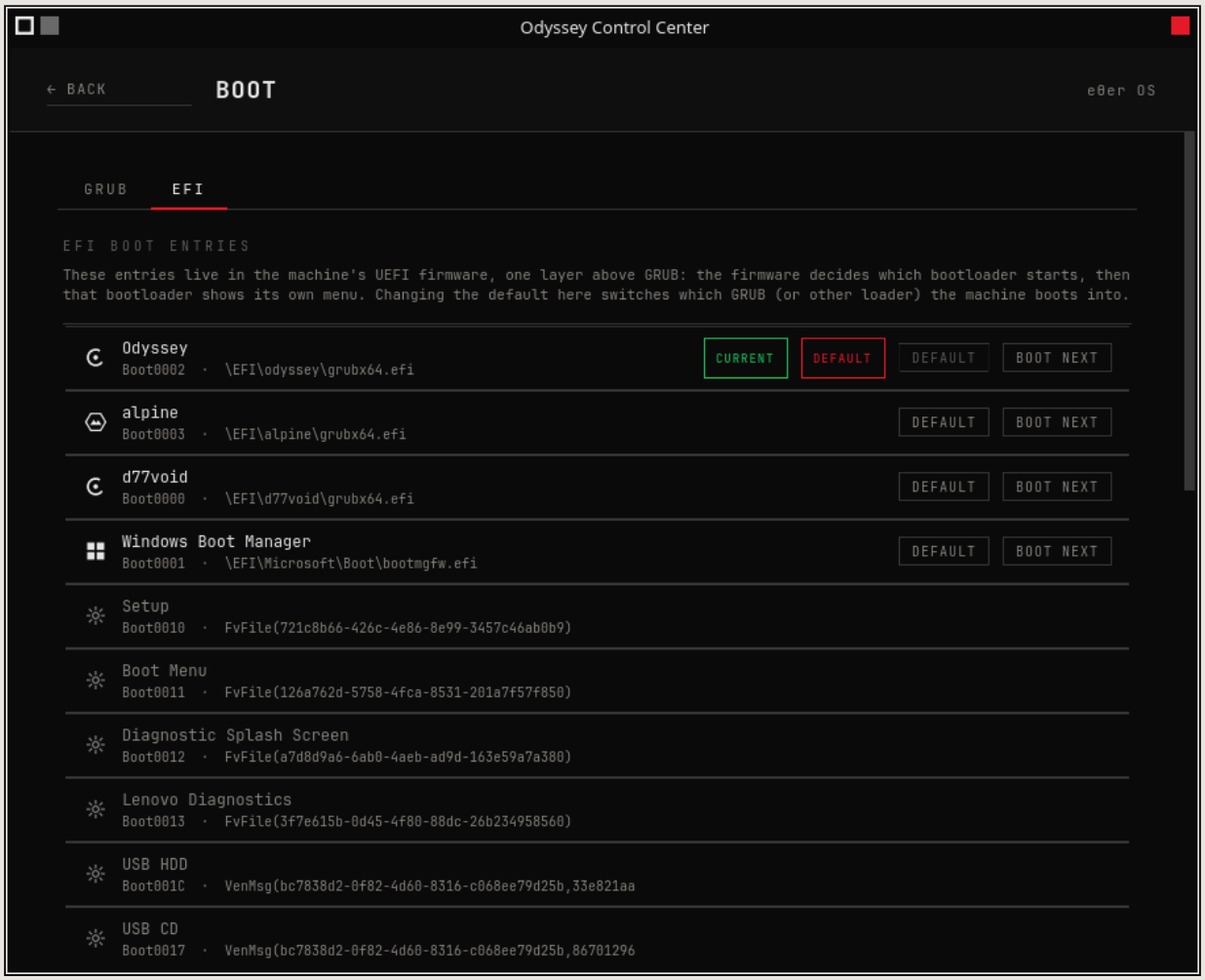

FIG · EFI 标签页：Odyssey 以绿色/红色标记为 CURRENT 和 DEFAULT， alpine、d77void 和 Windows Boot Manager 各自拥有自己的 DEFAULT / BOOT NEXT 按钮，下方还有固件实用工具条目。

每一行都是一个固件条目——它的名称、它的 GRUB id（`Boot0002` 等等），以及它指向的确切 `.efi` 文件。每行有两个按钮：

按钮	作用
DEFAULT	使这个条目成为固件的永久默认项——从现在起每次重启都会启动它。
BOOT NEXT	一次性的例外——这个条目只在**下一次**重启时启动，之后正常顺序会自动恢复。当有例外待处理时，顶部会出现一个带 CANCEL 按钮的横幅。

WHAT	Boot → EFI 标签页 → 找到你想要的条目（例如「Odyssey」或「Windows Boot Manager」）→ 点击 DEFAULT，或点击 BOOT NEXT 实现 一次性例外。
WHY	如果你主板自身的启动菜单直接启动了完全错误的引导加载 程序——这不是 GRUB 的设置问题，而是固件的设置问题——这里是唯一 能解决它的地方。
RESULT	DEFAULT：该条目永久移到固件启动顺序的最前面。BOOT NEXT：它启动一次，之后恢复原状。
RISK	较低，并且始终可以从这个同一个标签页恢复，或者从你主板 自身的 UEFI 设置界面（通常是开机时按 Del 或 F2）恢复，如果有 什么东西以意料之外的方式启动了的话。

2.4 内核参数

GRUB 标签页的最后一节是纯文本形式的内核命令行——可像普通文本框 一样编辑，不需要任何 GRUB 语法——以及以秒为单位的菜单超时时间。 **SAVE** 会把它们写入 `/etc/default/grub`；**GRUB-MKCONFIG** 则会根据这个文件重新生成真正的启动菜单，正是这一步让已保存的参数在下次 启动时生效。面板在重新生成时总是会重新应用你当前的默认选择，因此 这个操作永远不会重置默认启动的系统。

```
▸ ADVANCED

Past this point: config file paths, raw commands, and what the panel is doing underneath.
```

2.5 Advanced

实际发生了什么

Odyssey 的 GRUB 配置是**扁平的**——没有 `submenu` 代码块。每个内核 都是一个顶层的 `menuentry`，这意味着一个稳定的 id（而不是数字位置） 总是能正确选中它，即使跨越了内核更新。你在选择器中看到的「Advanced options」分组纯粹是视觉上的便利，它是通过匹配一个操作系统的简单 条目与其各个内核变体之间共享的 UUID 构建出来的——GRUB 自身的文件 并没有这种分组。

设置默认项会在同一个操作中写入两个地方：`/etc/default/grub` 中 持久化的 `GRUB_DEFAULT`，以及——因为 `grub.cfg` 可能不会立即 重新生成——直接对 `/boot/grub/grub.cfg` 中生效中的 `set default="..."` 那一行进行精确的 `sed` 编辑，跳过 `grub-reboot` 所使用的一次性分支 `${next_entry}`。这就是为什么默认项在下次启动 时就会生效，而不需要先强制执行一次完整的 `grub-mkconfig`。

```
$ cat /etc/default/grub | grep GRUB_DEFAULT
$ grep 'set default=' /boot/grub/grub.cfg
```

手动操作 OS-PROBER

```
$ sudo xbps-install -y os-prober
# 启用扫描——编辑 /etc/default/grub 中的 GRUB_DISABLE_OS_PROBER
$ sudo sed -i 's/GRUB_DISABLE_OS_PROBER=true/GRUB_DISABLE_OS_PROBER=false/' /etc/default/
grub
$ sudo os-prober
$ sudo grub-mkconfig -o /boot/grub/grub.cfg
```

手动操作 EFI——EFIBOOTMGR

EFI 标签页是 `efibootmgr` 的前端界面。它所做的一切都对应一条命令：

```
# 列出所有固件条目，按当前启动顺序排列
$ sudo efibootmgr -v
# 重新排序：把条目 0002 放到第一位（= DEFAULT）
$ sudo efibootmgr -o 0002,0000,0001,0003
# 一次性启动进入条目 0003（= BOOT NEXT）
$ sudo efibootmgr -n 0003
# 取消一个待处理的 BootNext
$ sudo efibootmgr -N
```

孤立的引导程序

有时安装程序会把自己的引导加载程序写入 EFI System Partition，却没有为它注册固件条目——引导程序存在于磁盘上，但固件并不知道它 在那里。Odyssey 的面板会直接扫描 ESP，并将这些条目单独列为**「ON DISK, NOT IN FIRMWARE」**，每一项都带有一键 REGISTER 按钮。手动 操作的等价命令是 `efibootmgr -c -d /dev/sdX -p N -L "名称" -l '\EFI\名称\bootx64.efi'`——请注意，新创建的条目会被固件自身放在 启动顺序的**最前面**，所以如果你不希望它抢占主导地位，之后要在你 常用的条目上重新应用 DEFAULT。

此面板涉及的文件

文件	作用
`/etc/default/grub`	内核命令行、超时时间、GRUB_DEFAULT、GRUB_DISABLE_OS_PROBER
`/boot/grub/grub.cfg`	实际生成的启动菜单本身——绝不要手动编辑它；它会被 grub-mkconfig 覆盖
EFI 变量 (NVRAM)	固件的启动条目——它根本不是一个文件；完全通过 efibootmgr 读取和写入

CHAPTER 3

Kernel

四个标签页——**SYSCTL**、**CPU GOVERNOR**、**HUGEPAGES**、**MODULES**——外加 位于全部四者之上的一排一键预设，它会同时应用到全部四个标签页。

3.1 图解指南——预设

顶部显示：你当前运行内核的版本、活跃的调度器，以及四个预设按钮—— **DESKTOP**、**GAMING**、**SERVER**、**BALANCED**。

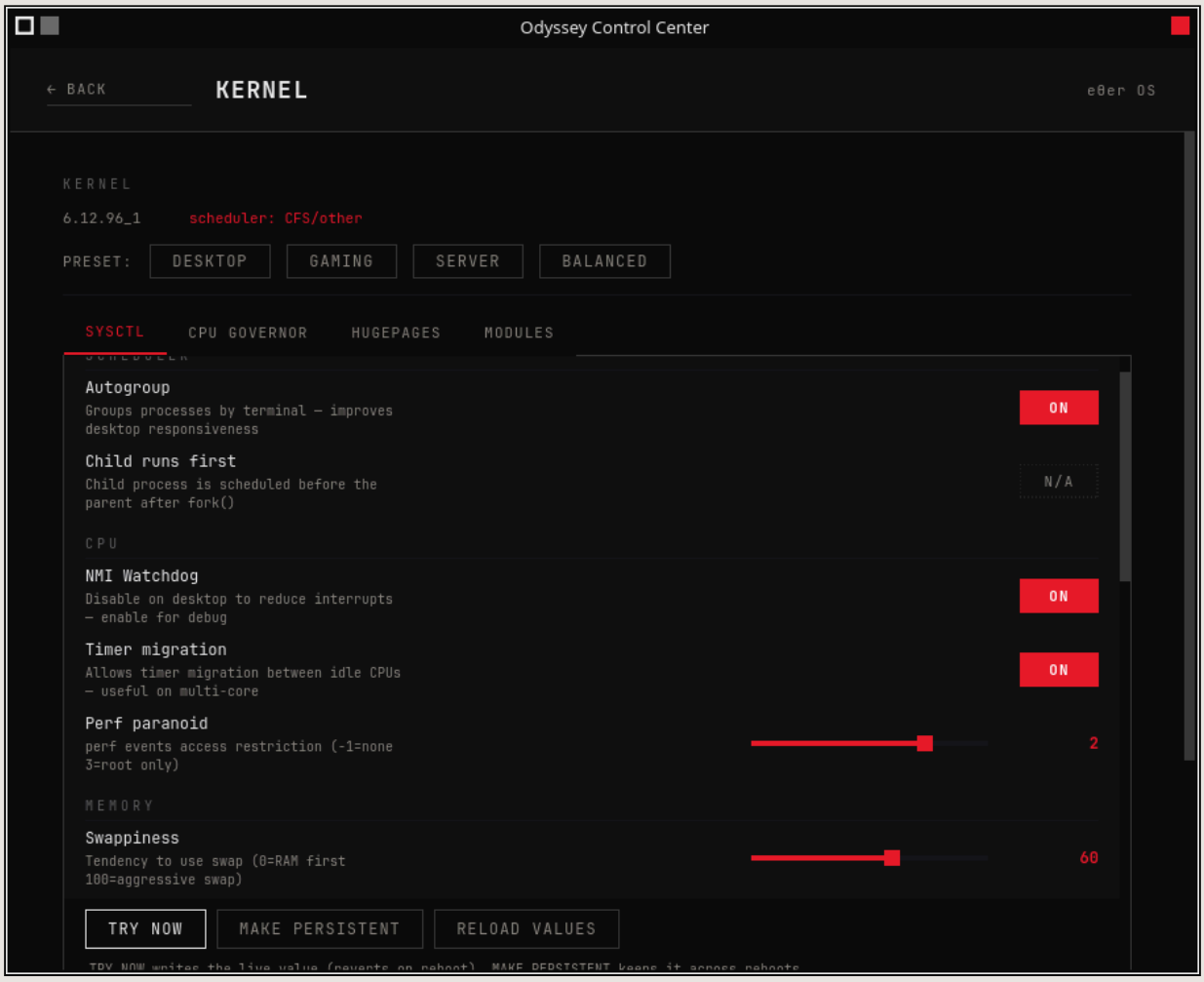

FIG · 刚打开的 SYSCTL 标签页：内核 6.12.96_1，调度器 CFS/other，四个 预设，以及下方的 Scheduler/CPU/Memory 分组。

预设	什么时候值得使用
DESKTOP	日常使用的机器——浏览网页、办公、轻度多任务。更低的 swappiness、适中的缓存压力、启用计时器迁移以实现空闲时的平稳能耗。对大多数人来说是合理的默认选择。

预设	什么时候值得使用
GAMING	swappiness 被压到接近零（把一切都留在 RAM 中），关闭计时器迁移（核心保持专用而不是迁移工作负载），放宽对 perf 事件的限制以配合 MangoHud 等性能分析工具。当帧时间比空闲功耗更重要时使用它。
SERVER	更高的 swappiness 和对缓存压力的容忍度，启用 NMI watchdog，对内核指针/日志施加更严格的限制。适用于优先考虑稳定性和安全性、而非交互响应速度的无显示器机器。
BALANCED	Desktop 和 Gaming 之间的折中——适用于一台时而是工作站、时而又是游戏机的电脑，而你不想不停地切换预设。

WHAT	Kernel → 点击 DESKTOP、GAMING、SERVER 或 BALANCED。
WHY	对于只想要「适合游戏的好设置」的人来说，手动调整十几个 sysctl 值要求太高——预设正是把这一切预先决定好的方案。
RESULT	SYSCTL 标签页中的每个值都会跳转到该预设的数值，准备就绪 且可见——你仍然需要 TRY NOW 或 MAKE PERSISTENT（见下文）才能 真正应用它们。
RISK	没有——选择预设只是填好了表单。

3.2 图解指南—SYSCTL，逐组讲解

向下滚动，每个参数都被归入一个标题下的分组，每一项都在面板中直接 给出通俗易懂的说明——你永远不需要在没有先在那里读到解释的情况下 就去理解 `vm.compaction_proactiveness` 到底是什么意思。

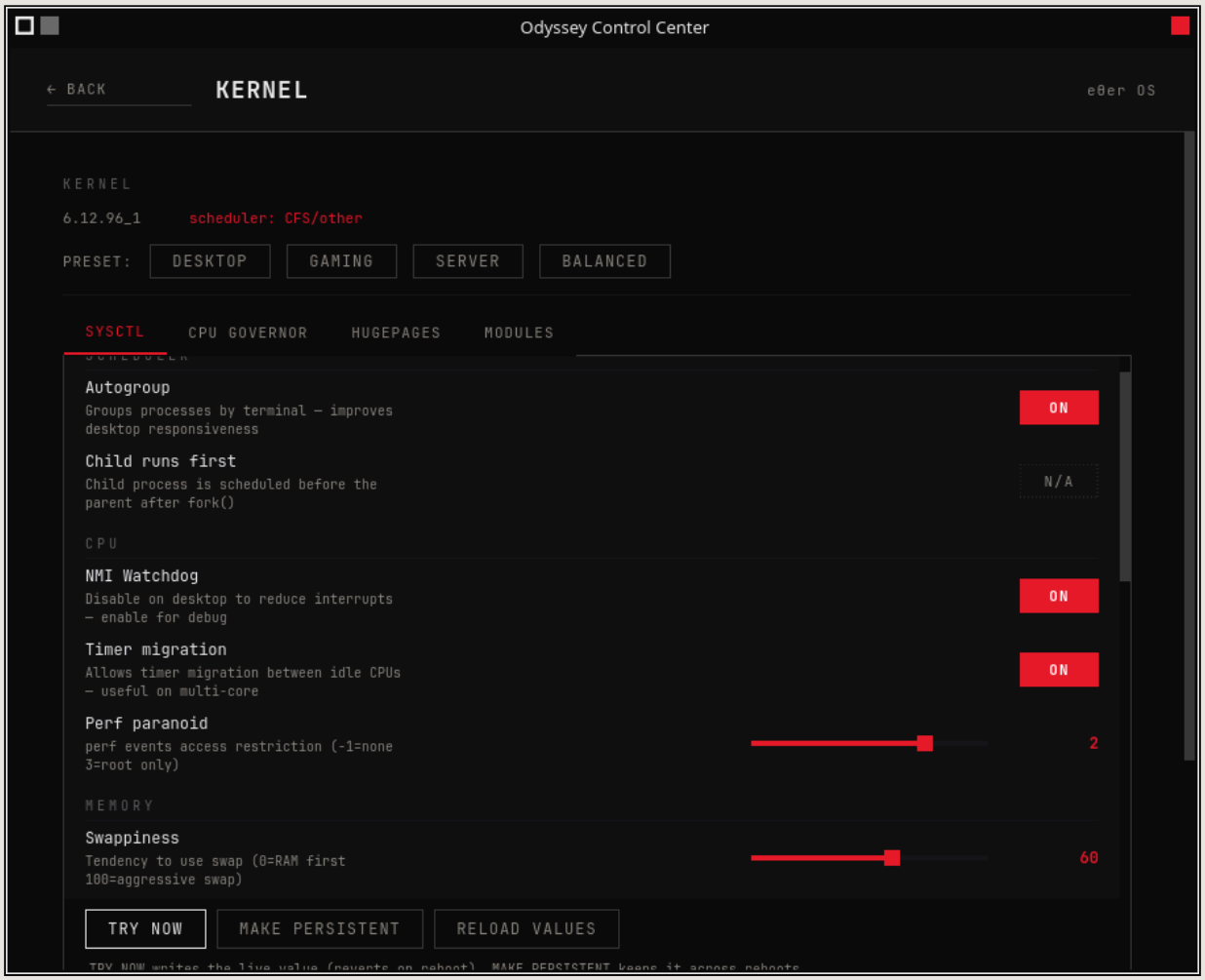

FIG · Scheduler 和 CPU 分组：Autogroup、Child runs first、NMI Watchdog、Timer migration、Perf paranoid——每一项都有自己的开关 或滑块，以及一行说明。

分组	包含内容
Scheduler	进程如何被分组和优先排序
CPU	Watchdog、计时器迁移、性能监控相关的限制
Memory	Swappiness、缓存压力、脏页回写阈值
Network	TCP Fast Open、backlog 队列大小、套接字缓冲区限制
Security	内核指针和日志消息是否对无特权用户隐藏

底部的两个按钮用于应用你的修改——它们被刻意设计成不同的：

WHAT	TRY NOW
WHY	以零风险测试一个值，不用担心它会在一次糟糕的重启后依然 存在。
RESULT	通过 `sysctl` 实时写入——立即生效，仅限本次会话。
RISK	没有——重启时会自动恢复原状。

WHAT	MAKE PERSISTENT
WHY	你已经测试过一个值（或整个预设），希望它能在重启后依然 保留。
RESULT	写入 `/etc/sysctl.d/99-odyssey.conf`，**同时**在同一次点击 中实时生效。
RISK	较低——它就是一个文本文件；删除该行或重新应用即可撤销。

RELOAD VALUES 会重新读取当前实际生效的内容，丢弃任何已输入但尚未 应用的修改。

3.3 图解指南—CPU governor

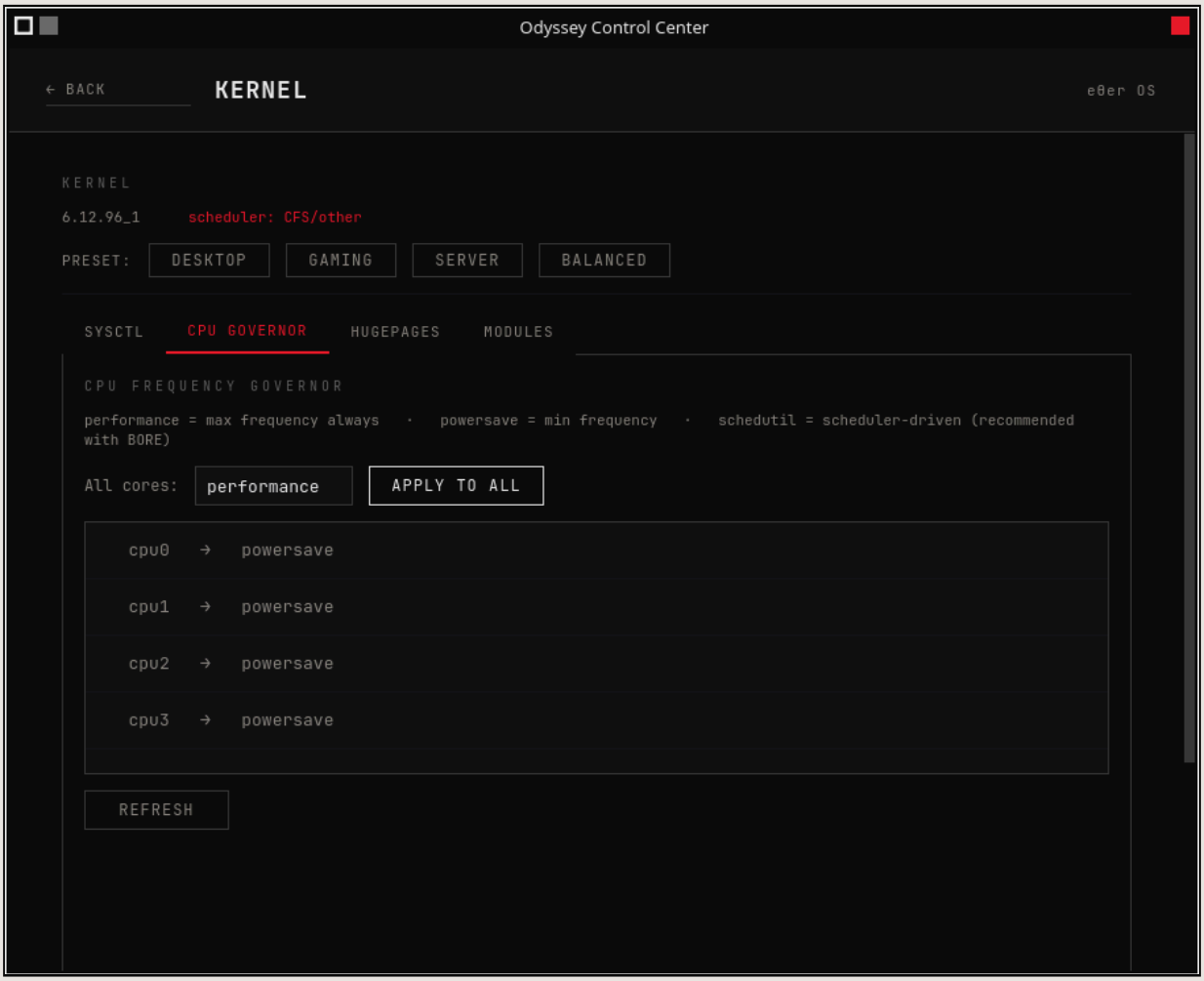

FIG · CPU GOVERNOR 标签页：「All cores」设置为 performance 并配有 APPLY TO ALL，而下方的逐核列表仍然全部显示为 powersave——一处 等待修正的不一致。

Governor 决定了你的 CPU 改变时钟频率的激进程度。一个下拉菜单可以 一次性把某个 governor 应用到所有核心；下方的列表则显示每个核心 **实时**的 governor 状态——这正是你发现某个核心卡在错误设置上的方法，正如上面的截图所示，「All cores」显示 performance，但下方每个核心 仍然显示 powersave，这说明 APPLY TO ALL 还没有被点击。

GOVERNOR	行为
performance	最高频率，始终如此——响应最快，功耗最高
powersave	最低频率，始终如此——最安静，响应最慢

GOVERNOR	行为
schedutil	由调度器自身对负载的判断来驱动频率——搭配 BORE 调度器时推荐使用

下拉菜单中只会出现你这块具体的 CPU 和内核实际支持的 governor。

3.4 图解指南——hugepages

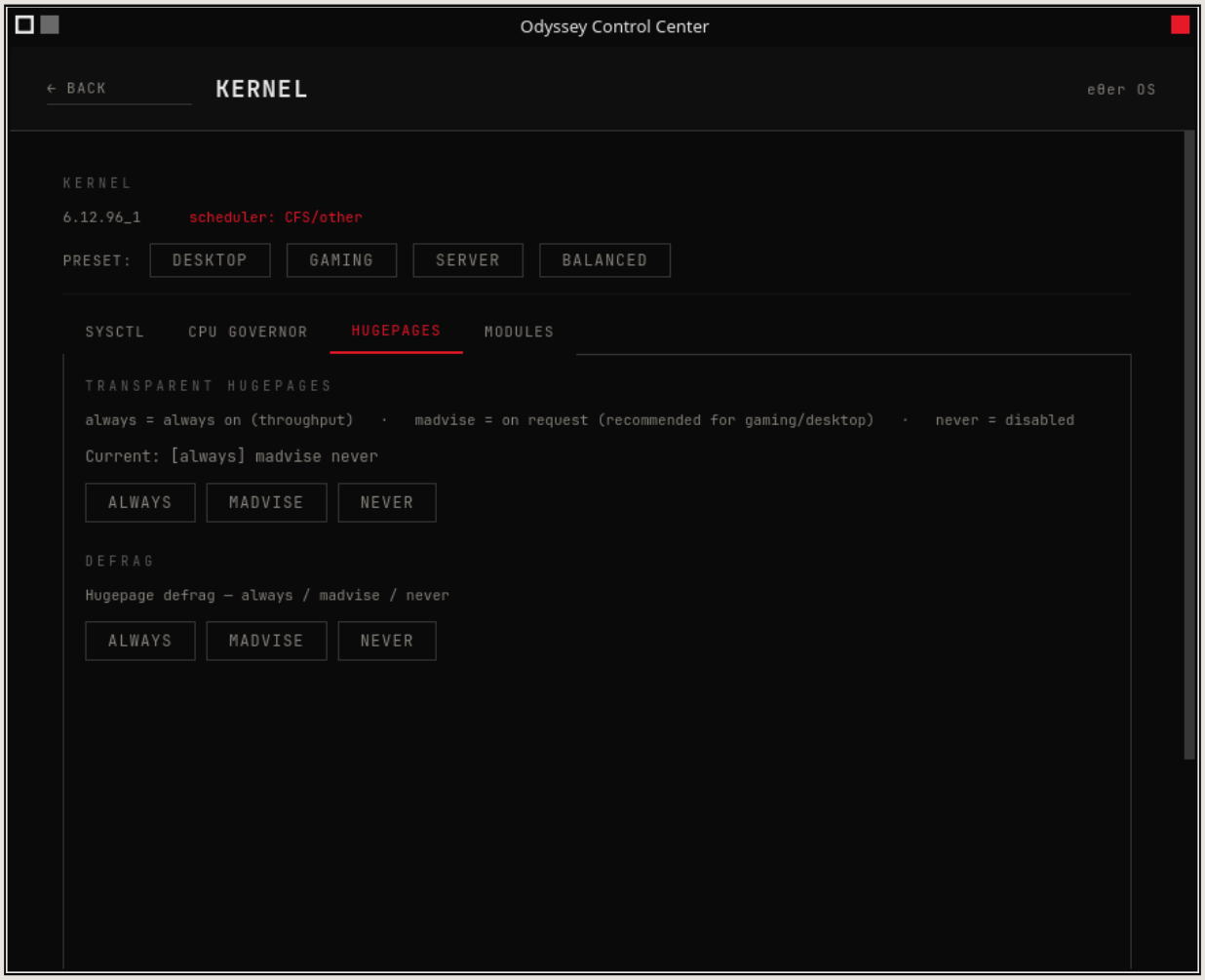

FIG · HUGEPAGES 标签页：Transparent Hugepages 显示「Current: [always] madvise never」，ALWAYS 被高亮，下方 Defrag 部分又是同样的三个 按钮一排。

两项独立的设置，各自是一排三个按钮，点击即刻生效——当前的实时值 总是以方括号的形式显示在按钮上方，就像截图中的 `[always] madvise never`。

设置	控制内容
Transparent hugepages	ALWAYS = 到处始终启用（最大吞吐量）；MADVISE = 仅在应用显式请求时启用（推荐用于游戏/桌面场景）；NEVER = 完全禁用
Defrag	内核在按需**形成** hugepages 时压缩内存的激进程度——同样的三个选项，独立于上面的设置

3.5 图解指南——modules

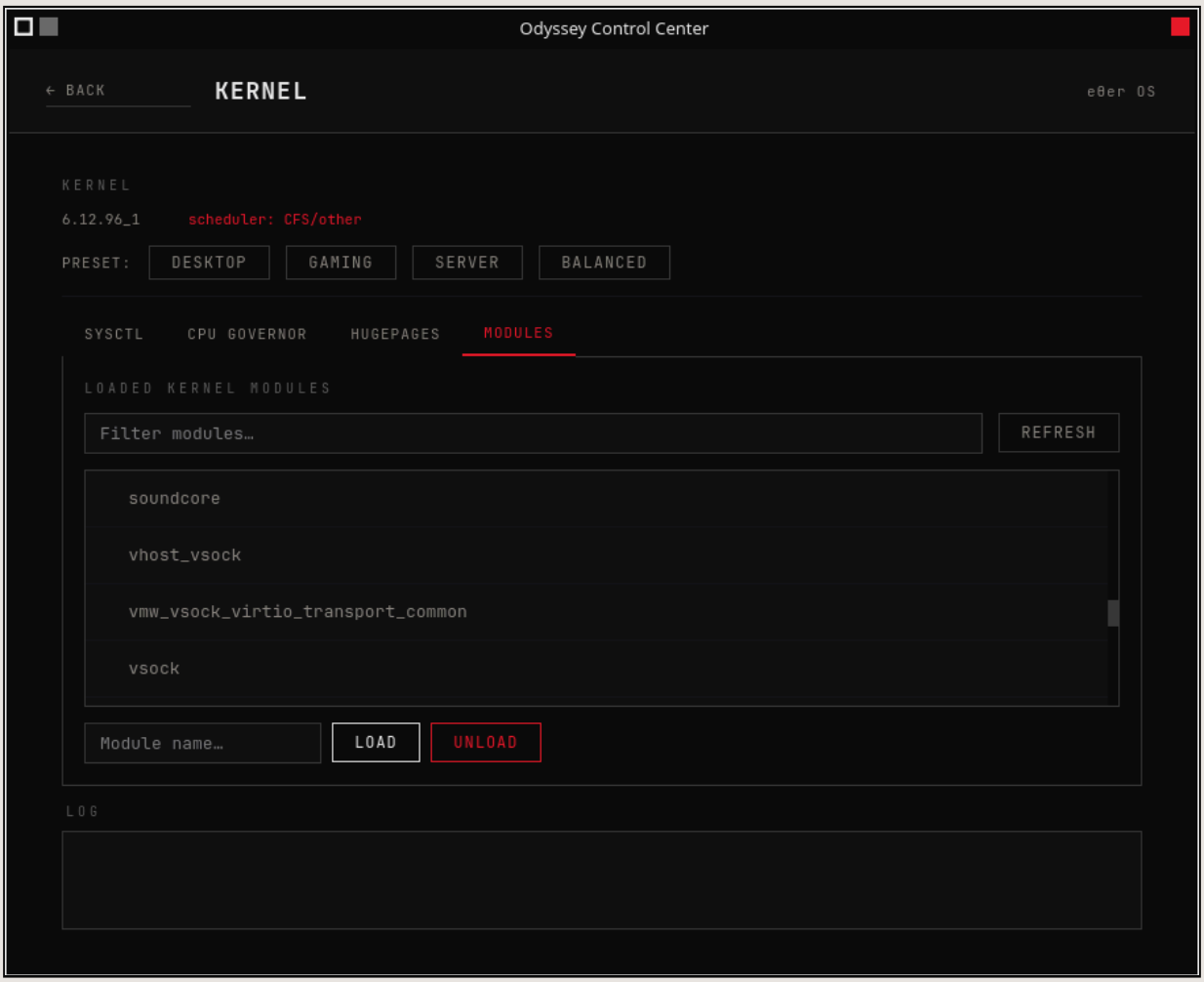

FIG · MODULES 标签页：一个过滤字段、一个实时列表（soundcore、 vhost_vsock、vsock……），以及底部用于按名称输入模块的 LOAD/UNLOAD 一排按钮。

一份当前已加载的每个内核模块的实时、可过滤列表——在过滤框中输入以 缩小范围，点击 REFRESH 重新读取。与其他三个标签页不同，这个标签页 允许你直接操作：输入一个模块名称，然后点击 **LOAD** 或 **UNLOAD**。

WHAT	Kernel → MODULES 标签页 → 输入一个模块名称 → LOAD 或 UNLOAD。
WHY	有时你需要在不重启的情况下让某个特定驱动或功能模块处于 激活（或移除）状态——测试硬件、释放被另一个模块占用的设备。
RESULT	该模块会立即被插入或从正在运行的内核中移除。
RISK	对 UNLOAD 而言风险适中——移除一个有其他东西依赖它的模块，可能会破坏正在使用它的东西的稳定性（显示驱动、文件系统）。LOAD 总体上是安全的；内核会直接拒绝加载一个不兼容的模块，而不是把 它应用到一半。

▸ ADVANCED

Past this point: config file paths, raw commands, and what the panel is doing underneath.

3.6 Advanced

确切的 SYSCTL 键

键	面板中的标签
kernel.sched_autogroup_enabled	Autogroup
kernel.sched_child_runs_first	Child runs first
kernel.nmi_watchdog	NMI Watchdog
kernel.timer_migration	Timer migration
kernel.perf_event_paranoid	Perf paranoid
vm.swappiness	Swappiness
vm.vfs_cache_pressure	VFS cache pressure
vm.dirty_ratio / vm.dirty_background_ratio	Dirty ratio % / Dirty background ratio %
vm.compaction_proactiveness	Compaction proactiveness
vm.watermark_boost_factor	Watermark boost
net.ipv4.tcp_fastopen	TCP Fast Open
net.core.netdev_max_backlog	Netdev max backlog
net.core.rmem_max	Socket rmem max
kernel.kptr_restrict / kernel.dmesg_restrict	kptr restrict / dmesg restrict

预设数值，DESKTOP VS GAMING

```
# DESKTOP
$ sysctl vm.swappiness=10 vm.vfs_cache_pressure=50 kernel.timer_migration=1
# GAMING
$ sysctl vm.swappiness=1 vm.vfs_cache_pressure=50 kernel.timer_migration=0
# 完整的预设数值表可以直接在 panel_kernel.py 中读到——
# 如果你想查看每一个字段，请打开 Control Center 的源代码
```

手动操作，不使用面板

```
# 实时，临时
$ sudo sysctl vm.swappiness=10
# 持久化——与面板写入的是同一个文件
$ echo “vm.swappiness=10” | sudo tee -a /etc/sysctl.d/99-odyssey.conf
$ sudo sysctl -system
# CPU governor，按核心
$ echo schedutil | sudo tee /sys/devices/system/cpu/cpu*/cpufreq/scaling_governor
# hugepages
$ echo madvise | sudo tee /sys/kernel/mm/transparent_hugepage/enabled
$ echo madvise | sudo tee /sys/kernel/mm/transparent_hugepage/defrag
# 模块
$ sudo modprobe MODULE_NAME
$ sudo modprobe -r MODULE_NAME
```

为什么 TRY NOW 和 MAKE PERSISTENT 是两个独立的操作

`sysctl` 的写入本质上永远只是实时的——内核本身没有内置的持久化 机制。MAKE PERSISTENT 是 Odyssey 选择了一个唯一的规范文件，`/etc/sysctl.d/99-odyssey.conf`，并且把实时写入和文件写入放在一起 同时执行，让两者永远不会产生分歧。直接编辑这个文件、然后运行 `sysctl --system`，其效果和点击按钮是完全一样的。

CHAPTER 4

Memory & Swap

只有一个屏幕，三个部分：一份只读摘要、swappiness，以及 ZRAM。

4.1 图解指南

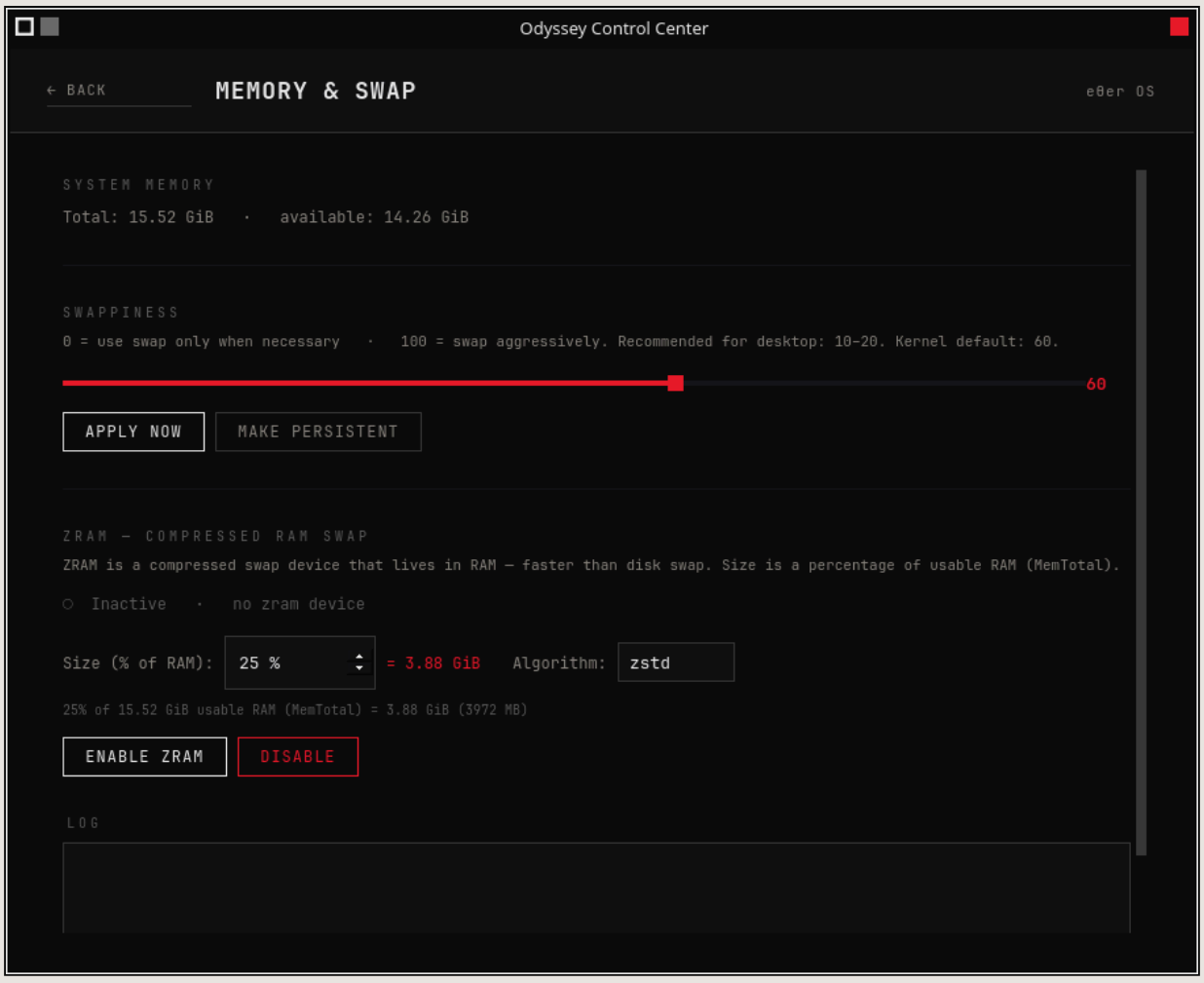

FIG · 完整面板：System memory（共 15.52 GiB，可用 14.26 GiB），Swappiness 滑块位于 60，配有 APPLY NOW / MAKE PERSISTENT，下方的 ZRAM 显示「Inactive · no zram device」，带有大小、算法和 ENABLE/DISABLE。

SYSTEM MEMORY

顶部只是一份摘要——总 RAM 以及当前可用量。没有任何控制项；它是下方 两个部分的背景信息。

SWAPPINESS

一个从 0 到 100 的单一滑块。0 表示「只有在真的别无选择时才使用 swap」；100 表示「积极使用 swap，远在 RAM 用完之前就开始」。内核的 默认值是 60（正是截图所显示的）；面板自身的提示建议桌面场景使用 10–20——这也正是 Kernel 面板中 DESKTOP 预设所设定的同一个数字。

WHAT	拖动滑块 → 点击 APPLY NOW 进行测试，满意后点击 MAKE PERSISTENT。
WHY	更低的值会把你正在积极使用的内容留在快速的 RAM 中，而不是 把它挪到更慢的 swap 中——只要安装了足够的内存，这通常正是你 想要的效果。
RESULT	APPLY NOW 会实时改变它，立即生效，仅限本次会话。MAKE PERSISTENT 会把它写入配置文件，使其在重启后也能保留。
RISK	很低。在一台真正缺乏 RAM 的机器上设置一个极低的值，可能 会让它在重负载下更容易突然耗尽内存，而不是平稳地进行 swap—— 如果发生这种情况，把它调高即可。

ZRAM—RAM 中的压缩交换区

ZRAM 是一种存在于 RAM **内部**的交换设备，经过压缩——因为完全不涉及 磁盘，所以明显快于磁盘上的 swap。它的大小是你可用总 RAM 的一个 百分比，而不是一个固定数字，因此这个设置能够合理地适应不同的机器；大小字段旁边的实时读数会为你把百分比换算成 GiB（截图中：25% = 在 15.52 GiB 系统上为 3.88 GiB）。

WHAT	设定大小百分比并选择一种算法（zstd、lz4、lzo 或 deflate）→ 点击 ENABLE ZRAM。
WHY	RAM 中的压缩 swap 能以远低于磁盘 swap 的性能代价来吸收内存 压力——在任何机器上都有用，在 RAM 有限的机器上尤其如此。
RESULT	一个 `zram0` 设备会以所选的大小和算法被创建并激活； 状态行会从「Inactive」变为激活状态。
RISK	较低。zstd（默认值）在压缩率和 CPU 开销之间为大多数机器 取得了良好的平衡。DISABLE 可以随时干净地将其关闭。

▸ ADVANCED

Past this point: config file paths, raw commands, and what the panel is doing underneath.

4.2 Advanced

这里「取决于硬件」到底意味着什么

这两项设置都与你具体的硬件相关，而不只是简单的「RAM 越多越好」：

- **高速 NVMe SSD**：swappiness 在 10–20 范围内的代价很小，即使发生 了 swap 也是如此——磁盘足够快，几乎察觉不到。ZRAM 在延迟和 SSD 磨损方面仍然占优，但差距比机械硬盘要小。
- **作为 swap 使用的机械 HDD 或 eMMC**：把 swappiness 保持在较低值 （0–10）——在慢速介质上进行磁盘 swap 会导致被称为「thrashing」 的卡顿。正是在这种情况下，ZRAM 带来的提升要大得多。
- **CPU 余量**：ZRAM 用磁盘 I/O 换取用于压缩和解压缩的 CPU 周期。 在 CPU 受限的机器上（较旧的硬件，或在游戏时已经满载的机器）， `lz4` 的压缩速度比 `zstd` 更快，但压缩率更差——如果 CPU 时间是 更紧张的资源，这是一个合理的取舍。在任何现代硬件上，`zstd` 更好的压缩率值得它稍高一点的代价。
- **RAM 余量**：在一台总共只有 8 GiB 的机器上，使用较大的 ZRAM 百分比（比如 50%）会为应用程序留下更少的真实 RAM，因为 ZRAM 本身也要消耗真实内存来存储压缩后的页面。在 16 GiB 或更多的机器 上，25% 是一个留有余地的舒适默认值。

手动操作

```
# swappiness，实时
$ sudo sysctl vm.swappiness=15
# swappiness，持久化——与面板写入的是同一个文件
$ echo “vm.swappiness=15” | sudo tee /etc/sysctl.d/99-odyssey.conf
# zram，手动配置
$ sudo modprobe zram
$ echo zstd | sudo tee /sys/block/zram0/comp_algorithm
$ echo 4G | sudo tee /sys/block/zram0/disksize
$ sudo mkswap /dev/zram0
$ sudo swapon /dev/zram0 -p 100
# 确认它已激活
$ swapon -show
```

为什么面板会重置 ZRAM0 而不是实时调整它的大小

ZRAM 设备的大小无法在它处于激活状态、正被用作 swap 时更改——内核 要求先将其禁用并重置。面板的 ENABLE ZRAM 按钮会自动执行这一整套 流程（swapoff、reset、重新配置、swapon），这就是为什么在一个已经 激活的设备上更改大小或算法，看起来仍然是一次干净利落的操作，而 不是一个错误。

CHAPTER 5

Firewall

Firewall 面板在 **Security and Hardening Guide** 中拥有自己专门的、 完整深入的章节——逐行解释的规则集、附带各自风险说明的全部 21 个 预设、原始配置编辑器，以及命令行等效操作。这一章则是图解引导版本。

5.1 图解指南

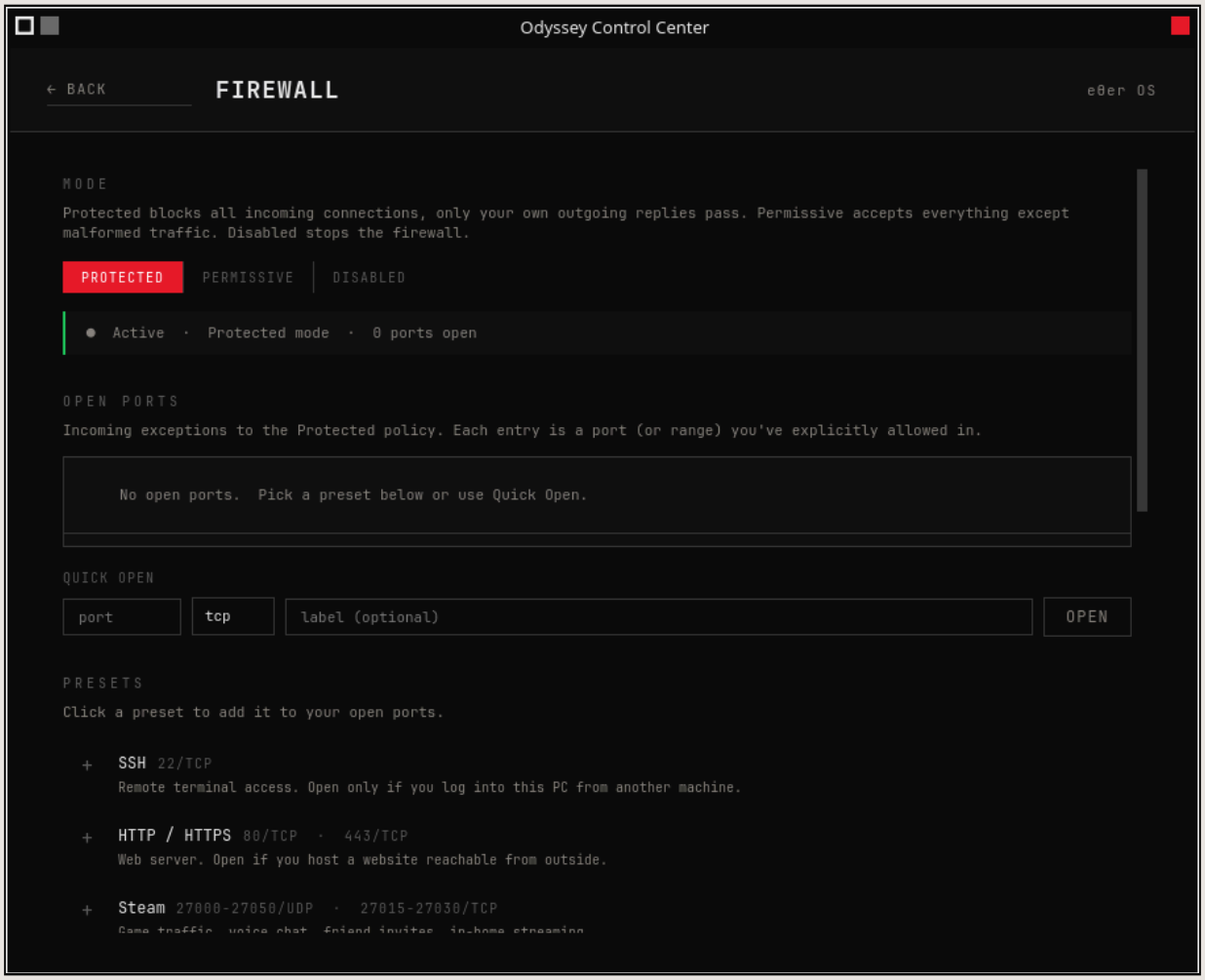

FIG · PROTECTED 模式处于激活状态，「0 ports open」，Quick Open 字段， 以及以 SSH、HTTP/HTTPS 和 Steam 开头的预设方块。

MODE

顶部有三个按钮，面板在它们正上方用一行文字分别解释了每一个：

模式	作用
PROTECTED	默认模式。阻止所有入站连接；只有你自己的出站响应可以通过。
PERMISSIVE	接受除格式错误流量之外的一切——一个用于诊断「是不是防火墙把这个搞坏了?」的设置，而不是应该长期停留的地方。
DISABLED	完全停止防火墙。

下方的状态行确认了实际生效的内容——在截图中，是 **「Active · Protected mode · 0 ports open」**。

OPEN PORTS 与预设

OPEN PORTS 列出你明确设置的例外——默认为空，如图所示。**QUICK OPEN** 直接接受端口、协议和可选标签，用于任何不在预设列表中的情况。 下方的 **PRESETS** 提供八个常见服务，每个只需一次点击，每一项都附有 一行文字，解释你何时真正需要它（SSH：「远程终端访问，只有当你从 另一台机器登录这台 PC 时才打开它」），此外在可展开的「MORE PRESETS」下还有另外十三个。

WHAT	Firewall → 点击一个预设方块，或者填写 QUICK OPEN → SAVE & APPLY。
WHY	你正在运行某个东西——一个游戏服务器、一个媒体服务器、一个 开发服务器——需要被另一台机器访问到。
RESULT	该端口被打开；其余所有端口在 PROTECTED 策略下仍保持 关闭。
RISK	完全取决于具体的预设——在打开 SSH 或 HTTP/HTTPS 之外的 任何端口、且该机器可从互联网访问之前，请查阅 Security and Hardening Guide 中针对各预设的风险说明。

在你点击 **SAVE & APPLY** 之前，任何更改都不会生效——更改会先进入 待定状态，让你能够在确认之前准确检查自己即将打开的是什么。

```
▸ ADVANCED

Past this point: config file paths, raw commands, and what the panel is doing underneath.
```

5.2 Advanced

完整的技术参考——逐条解释的 `nftables.conf` 默认规则集、全部 21 个预设各自的具体暴露面和缓解措施、仅限局域网访问的配置方案、`nft` 命令行等效操作，以及如何解决被锁在外面的 SSH 会话——都在 **Security and Hardening Guide 的第 2 章**中（安装到你系统上后位于 `/usr/share/doc/odyssey/`）。本手册刻意不重复这些内容；把那一章 视为本节的延续。

```
# 现在就值得记住的一条命令：
$ sudo nft list ruleset
# 精确显示内核中实际加载的内容，与面板无关
```

CHAPTER 6

AppArmor

五个标签页：**STATUS**、**PROFILES**、**COMMUNITY**、**VIOLATIONS**、 **EDITOR**。与 Firewall 一样的交叉参考：深入的参考资料——强制访问 控制到底是什么、完整的限制流程、一个实际案例——都在 Security and Hardening Guide 中。这里是图解指南。

6.1 图解指南——STATUS

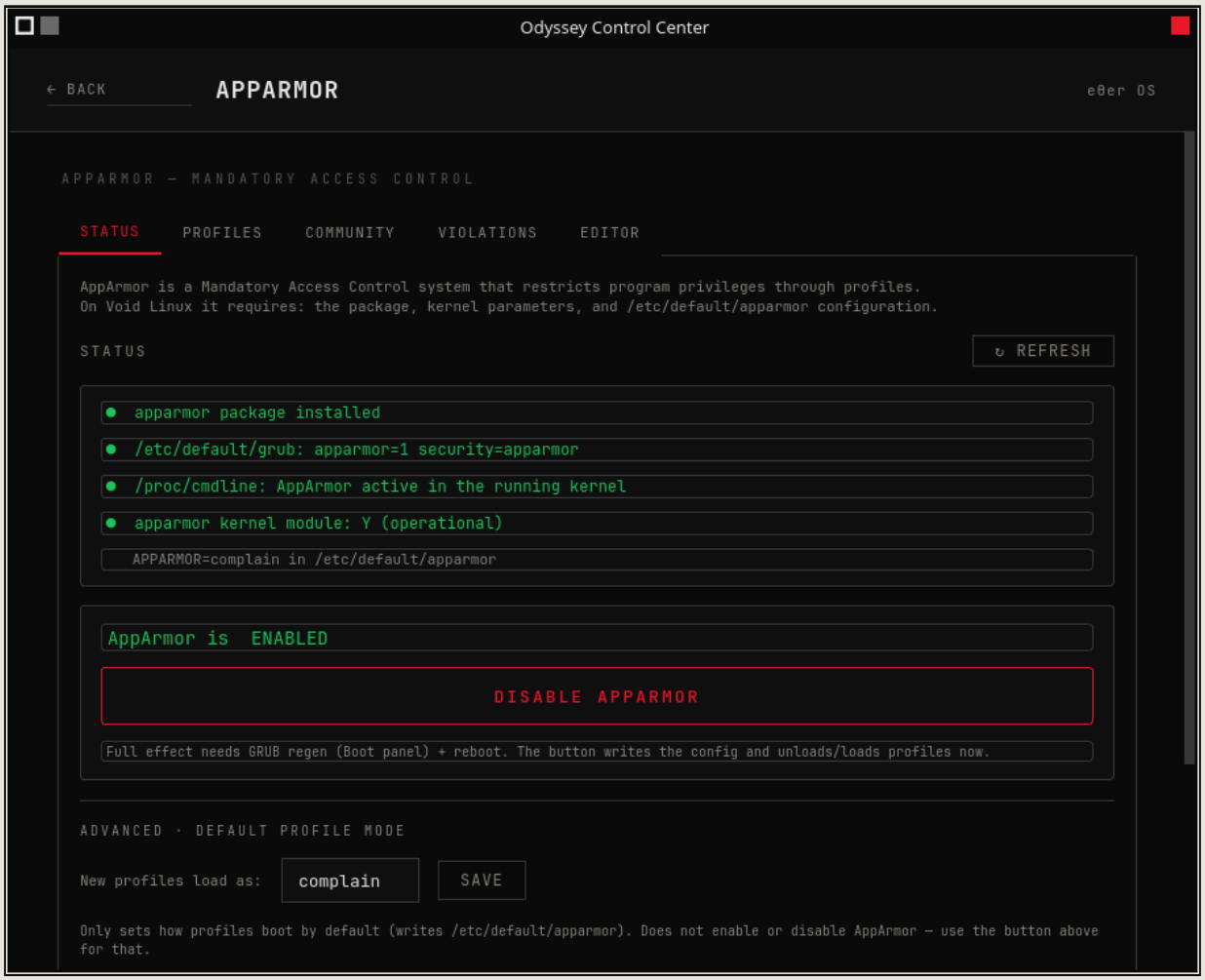

FIG · STATUS 标签页：四个绿色对勾（软件包已安装、GRUB 参数、正在运行 的内核、内核模块），「AppArmor is ENABLED」配有一个红色的 DISABLE APPARMOR 按钮，默认的配置文件模式被设为 complain。

四项检查从上到下确认了这个框架确实在正常工作：`apparmor` 软件包 已安装，GRUB 中的内核参数已设置，**正在运行的**内核确实是带着这些 参数启动的（这一项在你启用参数后如果没有重启，就可能滞后），以及 内核模块本身运行正常。下方是一个大的启用/禁用按钮，以及 **ADVANCED · DEFAULT PROFILE MODE**——新加载的配置文件是以 complain 还是 enforce 模式启动。

WHAT	AppArmor → STATUS → 确认全部四项检查都是绿色的。
WHY	如果某项检查是红色或黄色，这个面板里的其他任何东西都无法 正常工作——当 AppArmor 出现问题时，这永远是第一个该查看的 地方。
RESULT	确信这个框架确实是激活的，而不只是安装了而已。
RISK	没有——只读验证。

6.2 图解指南——PROFILES

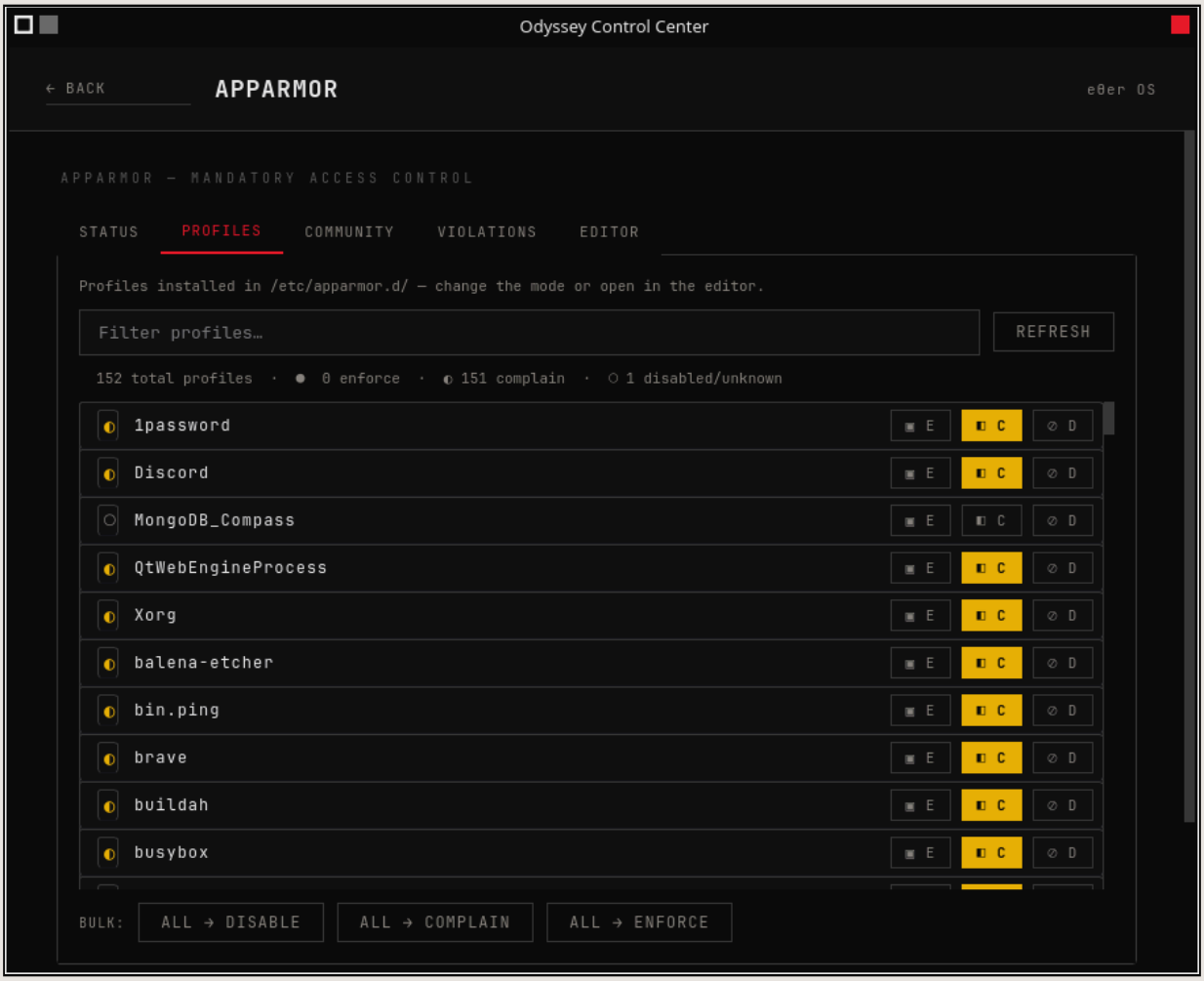

FIG · PROFILES 标签页：共 152 个配置文件，0 个 enforce，151 个 complain，1 个 disabled/unknown——一个可过滤的列表，包含 1password、Discord、MongoDB_Compass 等，每一项都有 E / C / D 按钮，底部是 ALL→DISABLE / ALL→COMPLAIN / ALL→ENFORCE。

每个已加载的配置文件占一行，顶部有过滤字段和计数（截图中：152 个 配置文件，除一个之外全部处于 complain 模式——这是 Odyssey 的默认 状态）。每行三个按钮：

按钮	效果
E	将该配置文件切换为 enforce——违规行为会被阻止，而不仅仅是被记录
C	切换为 complain——记录违规行为，但不阻止任何东西
D	完全禁用该配置文件——程序在无限制状态下运行

底部的三个 **BULK** 按钮会同时对列表中的每一个配置文件应用相同的 切换操作——在从 COMMUNITY 批量安装之后很有用，但请对 ALL→ENFORCE 格外谨慎；请参见下一节中的操作说明。

6.3 图解指南——COMMUNITY

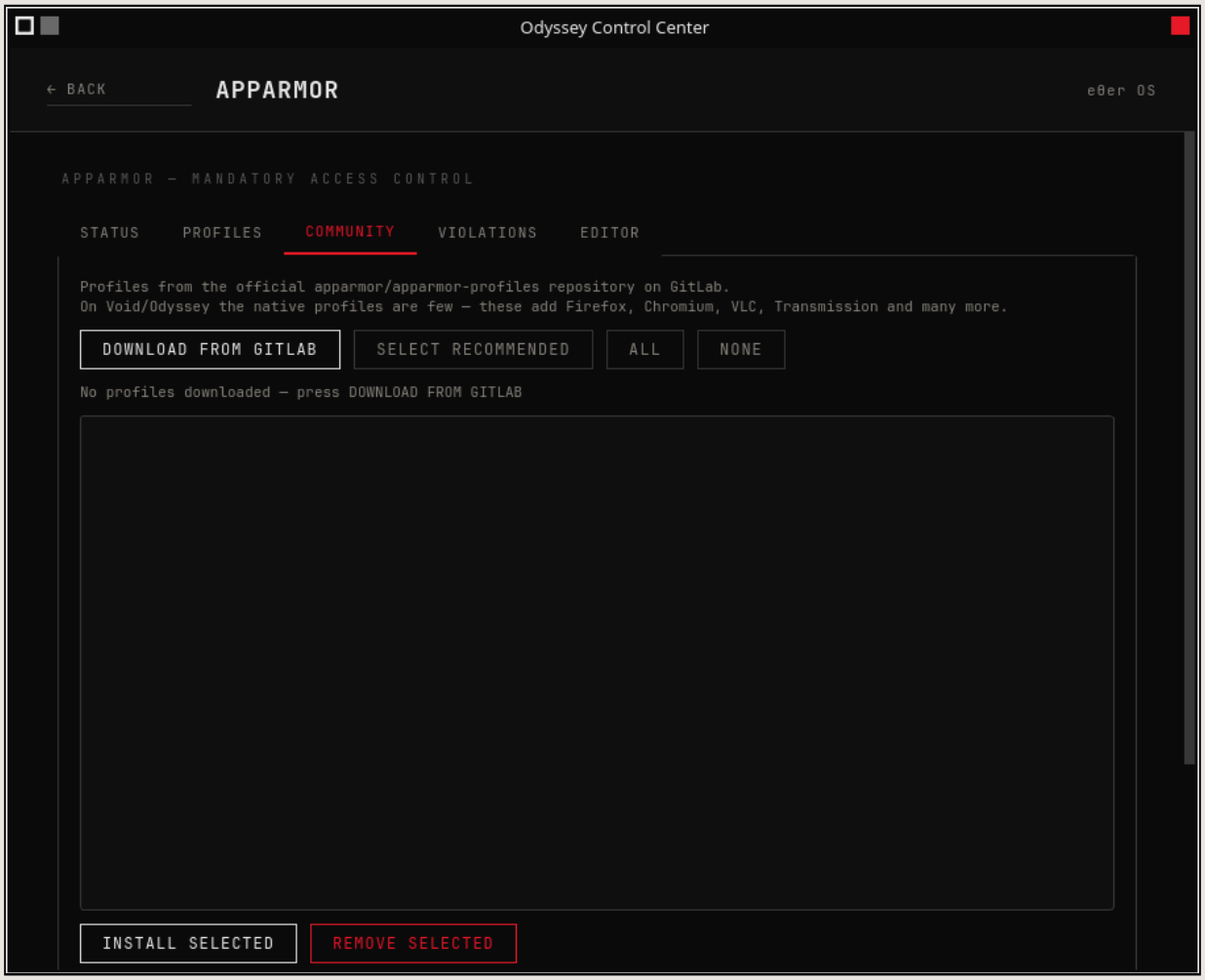

FIG · COMMUNITY 标签页：「No profiles downloaded — press DOWNLOAD FROM GITLAB」，配有 DOWNLOAD FROM GITLAB、SELECT RECOMMENDED、ALL 和 NONE 按钮，底部是 INSTALL SELECTED / REMOVE SELECTED。

出厂时的 AppArmor 主要限制系统组件——你真正关心的桌面应用程序（浏览器、聊天应用、媒体播放器）需要来自社区的配置文件。这个标签页 直接从 GitLab 上的官方仓库 `apparmor/apparmor-profiles` 获取它们。

1. **DOWNLOAD FROM GITLAB**

   获取当前可用配置文件的列表—— 此时还没有安装任何东西，只是填充这个列表。

2. **SELECT RECOMMENDED（或 ALL / NONE）**

   勾选一组适合桌面 场景的合理默认集合——Firefox 系浏览器、Chromium、VLC、 Transmission 等类似应用。

3. **INSTALL SELECTED**

   安装已勾选的配置文件——它们和其他所有 配置文件一样以 complain 模式加载，为下面「先观察、后应用」的循环 做好准备。

WHAT	AppArmor → COMMUNITY → DOWNLOAD FROM GITLAB → SELECT RECOMMENDED（或单独选择）→ INSTALL SELECTED。
WHY	那些最容易暴露于不可信输入的应用程序——渲染恶意页面的 浏览器、打开精心构造的文件的媒体播放器——恰恰是配置文件保护 效果最好的对象。
RESULT	新的配置文件会出现在 PROFILES 中，处于 complain 模式， 记录下它们本会阻止的行为，但目前尚未阻止任何东西。
RISK	安装时没有任何风险——在你于 PROFILES 中做出决定之前， 不会应用任何限制。

6.4 图解指南—VIOLATIONS

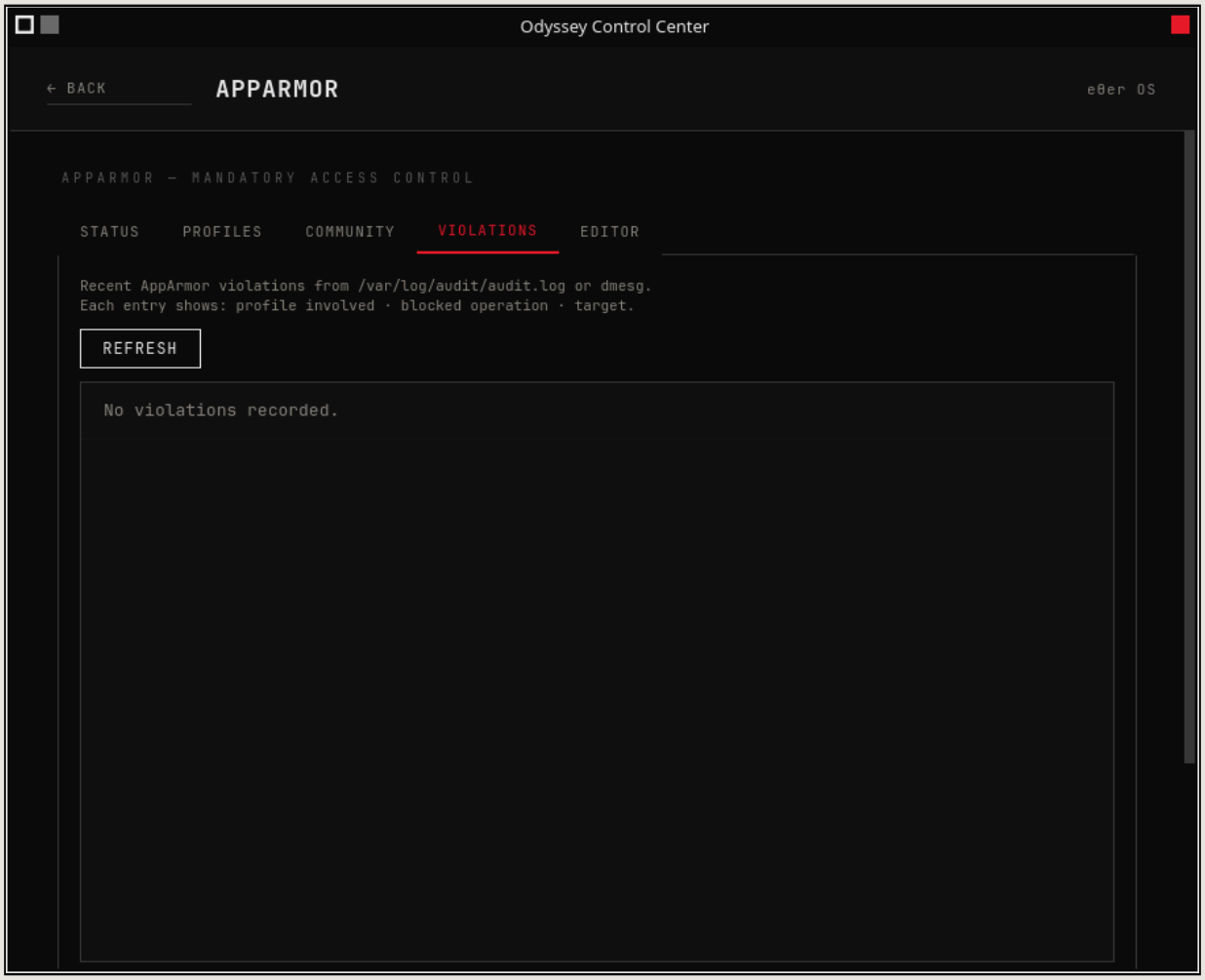

FIG · VIOLATIONS 标签页：「No violations recorded」，配有一个 REFRESH 按钮，以及一条说明，指出条目来自 /var/log/audit/audit.log 或 dmesg。

这个标签页读取系统的 AppArmor 审计日志，显示某个配置文件**本会**阻止的行为（在 complain 模式下）或**已经**阻止的行为（在 enforce 模式下）——每条记录都指明了配置文件、被阻止的操作，以及目标文件 或资源。

空列表是健康系统上预期看到的画面

截图展示的正是你通常会看到的情况：没有违规记录。这不是一个 bug，也不是面板未正常工作的迹象——它意味着自上次检查日志以来，每个 被限制的程序都始终待在允许的边界之内。当某个被限制的程序尝试做 它的配置文件不允许的事情时，违规记录就会在此刻出现在这里；去 使用一下相应的程序，回来，再点击 REFRESH。

一旦出现记录，双击其中任何一条都是进入 EDITOR 的快捷方式，届时那个 确切的配置文件已经被打开——详见下文。

6.5 图解指南—EDITOR

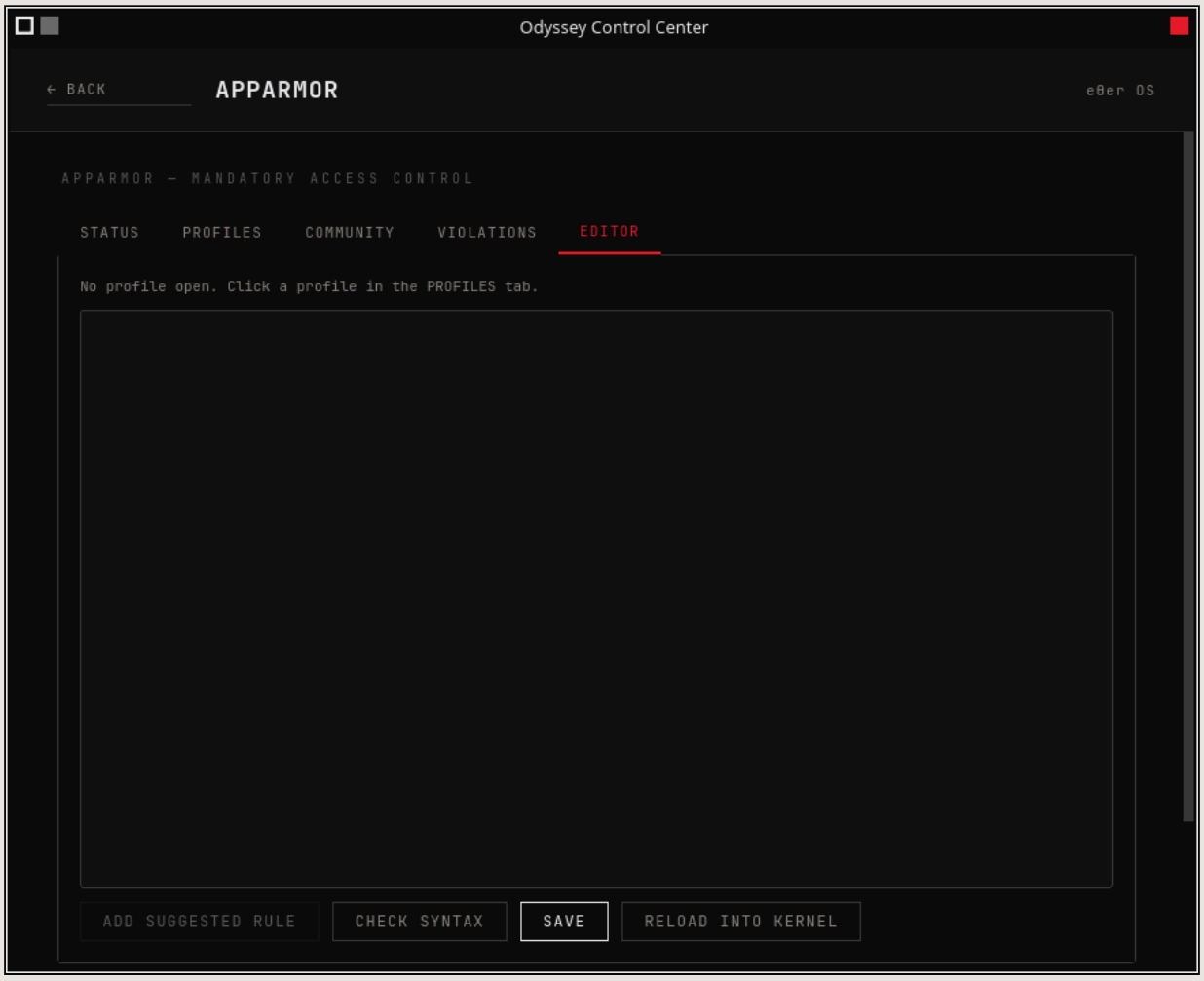

FIG · EDITOR 标签页：「No profile open. Click a profile in the PROFILES tab」，一片空白文本区域，以及四个按钮—ADD SUGGESTED RULE、CHECK SYNTAX、SAVE、RELOAD INTO KERNEL—除 SAVE 外全部 处于禁用状态。

编辑器故意以**空白**状态打开，如图所示——它一次只编辑一个配置文件， 你可以从 PROFILES（点击某一行）或 VIOLATIONS（双击某条记录）中 选择要编辑哪一个。这可以防止不小心编辑错误文件这种容易犯的错误。

按钮	作用
ADD SUGGESTED RULE	仅在你是从某条 VIOLATIONS 记录进入时才可用——插入原本会允许该记录动作的确切规则行
CHECK SYNTAX	在不加载配置文件的情况下验证它——在打字错误造成问题之前捕捉到它
SAVE	把你的修改写入磁盘上的配置文件

按钮	作用
RELOAD INTO KERNEL	让已保存的修改立即生效，不需要等待下一次启动或服务重启

WHAT	PROFILES → 点击某个配置文件的名称 → EDITOR 会打开它 → 进行修改 → CHECK SYNTAX → SAVE → RELOAD INTO KERNEL。
WHY	这就是整个限制流程在一行文字中的体现：先在 complain 模式 下观察（VIOLATIONS），修正真正缺失的部分（EDITOR），然后才把 这个配置文件切换为 enforce（PROFILES）。
RESULT	配置文件会精确按照编辑后的内容实时生效，无需重启。
RISK	错误地编辑一个配置文件，可能会在 enforce 模式下之后阻止 某些合法的操作——CHECK SYNTAX 会在 SAVE 之前捕捉到格式错误的 规则，但捕捉不到逻辑错误。如果你编辑过的配置文件在 enforce 模式下开始造成问题，就从 PROFILES 把它切回 complain，再次 检查 VIOLATIONS。

▸ ADVANCED

Past this point: config file paths, raw commands, and what the panel is doing underneath.

6.6 Advanced

命令行等效操作

```
$ sudo aa-status
# 与 PROFILES 标签页头部相同的统计数字，来自终端
$ sudo aa-enforce /etc/apparmor.d/PROFILE_FILE
$ sudo aa-complain /etc/apparmor.d/PROFILE_FILE
$ sudo aa-disable /etc/apparmor.d/PROFILE_FILE
# 相当于手动编辑之后的 RELOAD INTO KERNEL
$ sudo apparmor_parser -r /etc/apparmor.d/PROFILE_FILE
```

阅读原始审计日志

```
$ sudo grep apparmor /var/log/audit/audit.log | tail -20
# 或者，如果没有专门的审计守护进程在运行：
$ sudo dmesg | grep -i apparmor
```

手动阅读一行日志时值得了解的字段：`apparmor="DENIED"`（enforce）或 `apparmor="ALLOWED"`（complain，仅记录）、`operation=`（尝试 执行的操作）、`profile=`（是哪个配置文件做出的判定——对应 EDITOR 打开的那个）、`name=`（路径或资源）、`comm=`（进程）。

为什么批量 ALL→ENFORCE 有风险

在某个配置文件的 VIOLATIONS 日志沉寂了一段时间之后再逐个将其升级， 是安全的做法——你能够逐个配置文件地确认，正常使用不会触发它。 ALL→ENFORCE 会同时跳过所有配置文件的这段观察期，包括那些你可能 刚从 COMMUNITY 安装、却从未真正使用过的配置文件。一个还没有被 真正「过日子」验证过的配置文件一旦被强制执行，就可能在你第一次 使用某个合法功能的那一刻将其阻止。更倾向于逐一进行的循环；把批量 操作留给 ALL→COMPLAIN（始终安全——只会放松限制）或者在排查问题 时使用 ALL→DISABLE。

此面板涉及的文件

路径	作用
`/etc/apparmor.d/`	每个被限制的程序对应一个配置文件——正是 PROFILES 所列出、EDITOR 所打开的内容
`/etc/default/apparmor`	启动时的默认模式（`APPARMOR=complain`）以及 enable/disable 状态
`/etc/default/grub`	STATUS 所检查的内核参数 `apparmor=1 security=apparmor`

CHAPTER 7

Privacy

两个独立的工具，各自拥有自己的三步流程 install → enable → running，两者都在运行起来之后还有一座桥梁把它们连接在一起。

7.1 图解指南——SearXNG，逐步讲解

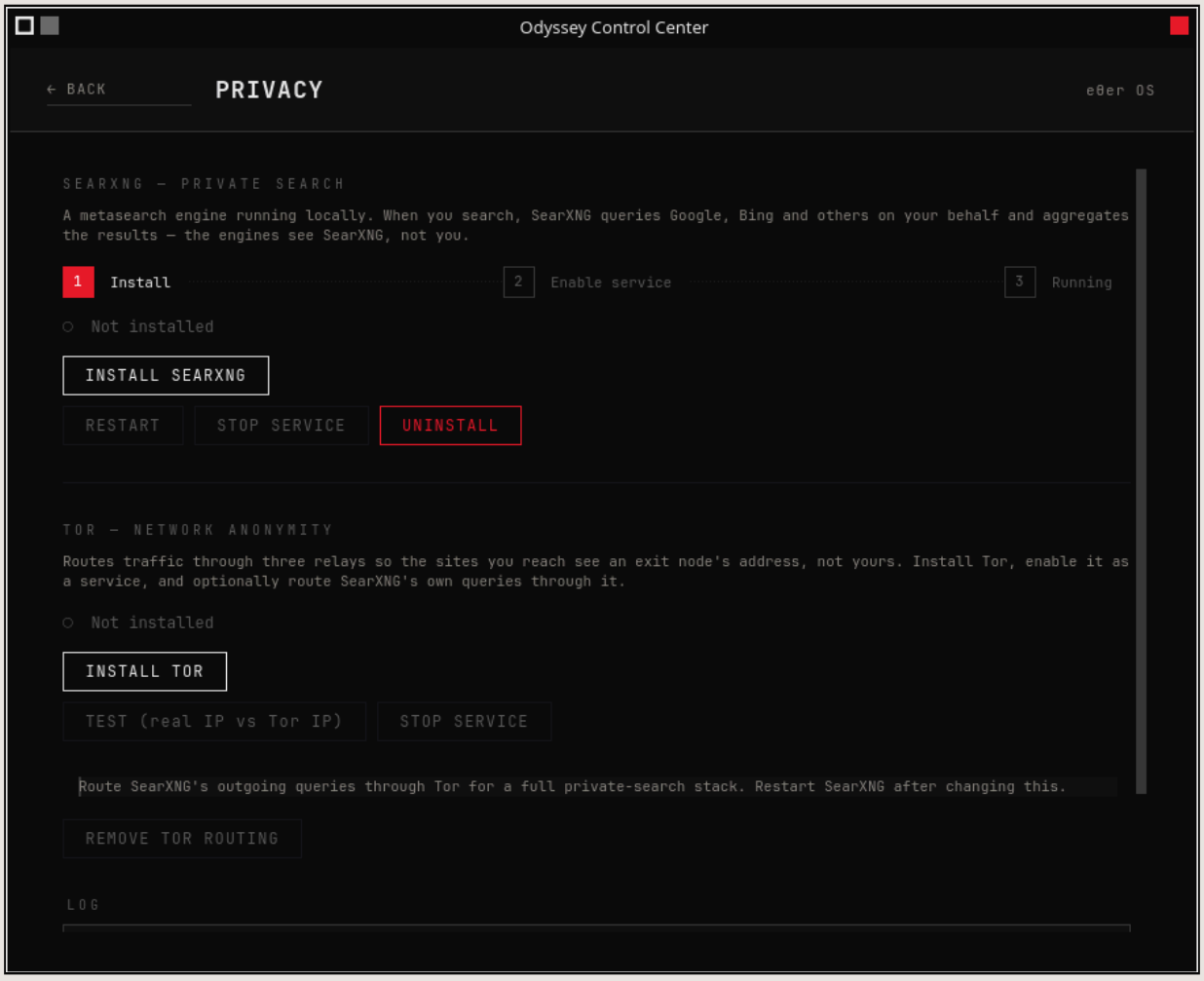

FIG · 面板处于静止状态：SearXNG 的追踪器处在第 1 步「Install」，状态为「Not installed」，只有一个 INSTALL SEARXNG 按钮；Tor 的追踪器同样显示「Not installed」，下方是 INSTALL TOR。

SearXNG 是一个在你自己机器上本地运行的元搜索引擎——当你搜索时， 是 SearXNG 代表你去查询 Google、Bing 等等，因此这些搜索引擎看到的 是 SearXNG 发出的请求，而永远不是你本人直接发出的。

1 INSTALL SEARXNG

点击它。日志会显示真实的安装过程正在 进行——克隆源代码、创建一个系统用户、配置一个 Python 虚拟 环境。

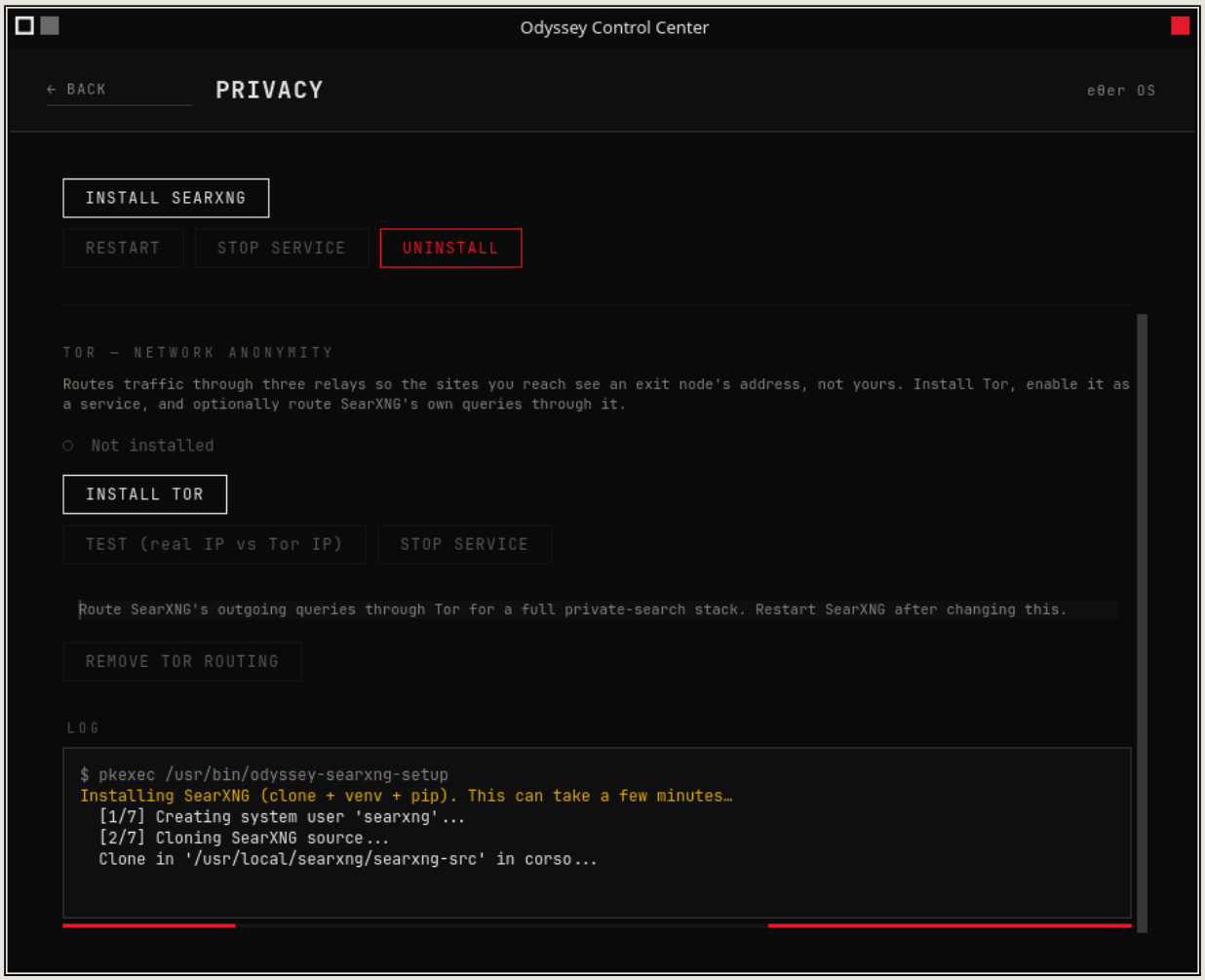

FIG · 安装过程中：日志正在传输真实的输出——「Installing SearXNG (clone + venv + pip). This can take a few minutes…」，可以 看到 1/7 和 2/7 的步骤。

2 ENABLE SERVICE

安装完成后，追踪器进入第 2 步，状态行 变为黄色：「Installed — service not enabled」。现在只有一个 按钮：ENABLE SERVICE。

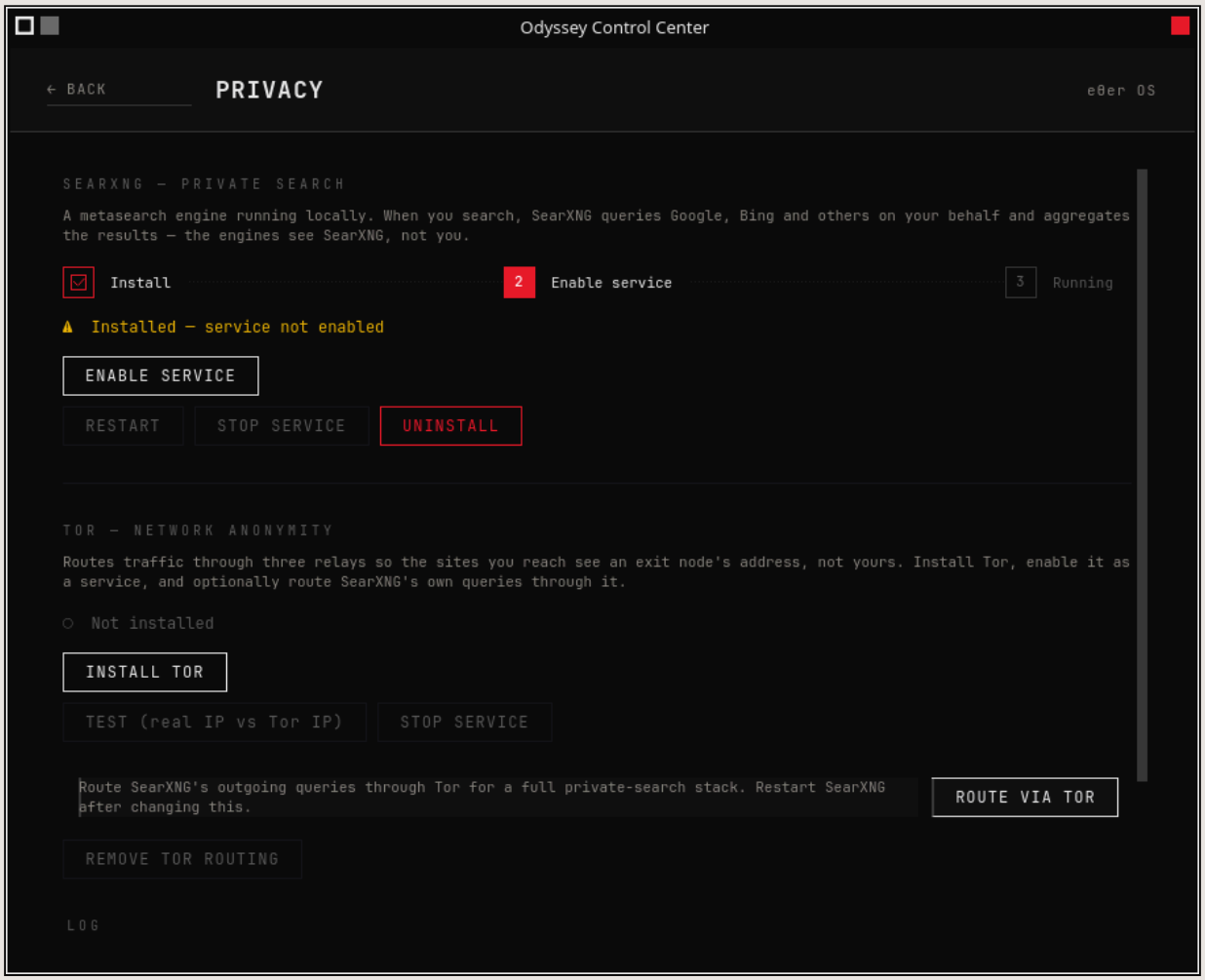

FIG · 第 2 步处于激活状态：「Installed — service not enabled」， ENABLE SERVICE 是唯一激活的按钮，RESTART/STOP SERVICE/UNINSTALL 在下方仍处于禁用状态。

3 Running

启用之后，追踪器到达第 3 步，整排控制按钮 都会被激活。
一旦服务运行起来，**OPEN IN BROWSER** 会直接带你过去——SearXNG 默认 监听在 `http://localhost:8888`。

WHAT	Privacy → INSTALL SEARXNG → ENABLE SERVICE → OPEN IN BROWSER。
WHY	每一个商业搜索引擎都会随着时间的推移根据你的查询建立起 一份画像；一个本地的元搜索实例意味着只有你自己能够看到那个 未经汇总的原始查询。
RESULT	一个运行在你自己机器上的私密搜索引擎，可以在浏览器中 访问，代表你去查询各大搜索引擎，而不会让你直接暴露在它们面前。
RISK	较低——它就是一个普通的本地 web 服务。如果它出现异常就 RESTART，STOP SERVICE 可以在不移除它的情况下暂停它， UNINSTALL 则会将其完全移除。

7.2 图解指南—Tor，逐步讲解

同样的三步结构，独立的追踪器，就在 SearXNG 部分的正下方。

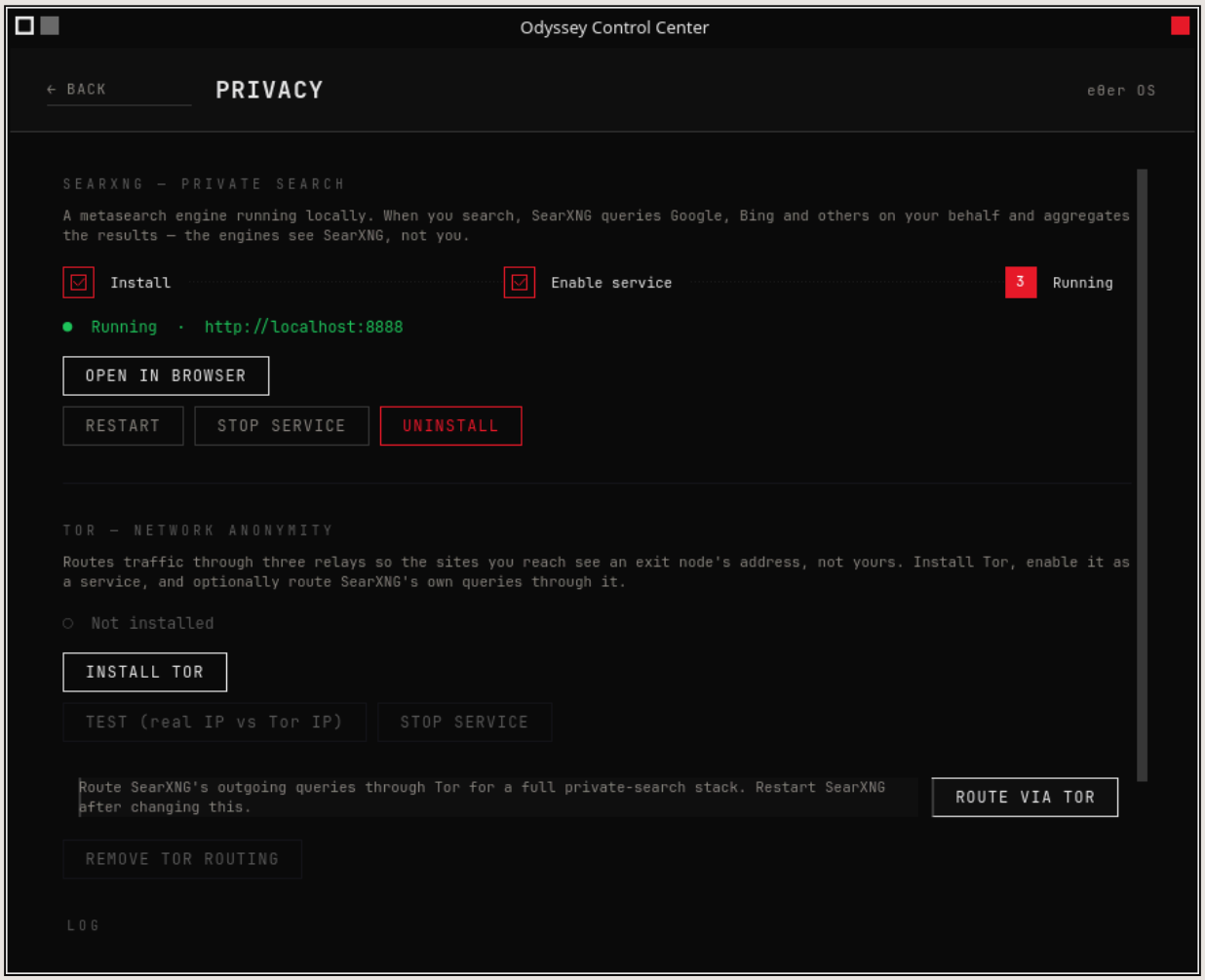

FIG · 上方 SearXNG 已完全运行；下方的 Tor 部分仍处于「Not installed」 状态，INSTALL TOR 是唯一激活的按钮。

Tor 会在你的流量到达目的地之前，通过三个中继节点来转发它，因此你 访问的网站看到的是出口中继节点的地址，而永远不是你自己的。

1 **INSTALL TOR**

日志确认软件包已安装，并显示该服务已链接 进 runit。

2 **ENABLE SERVICE**

状态变为「Installed — service not enabled」——与 SearXNG 相同的模式。

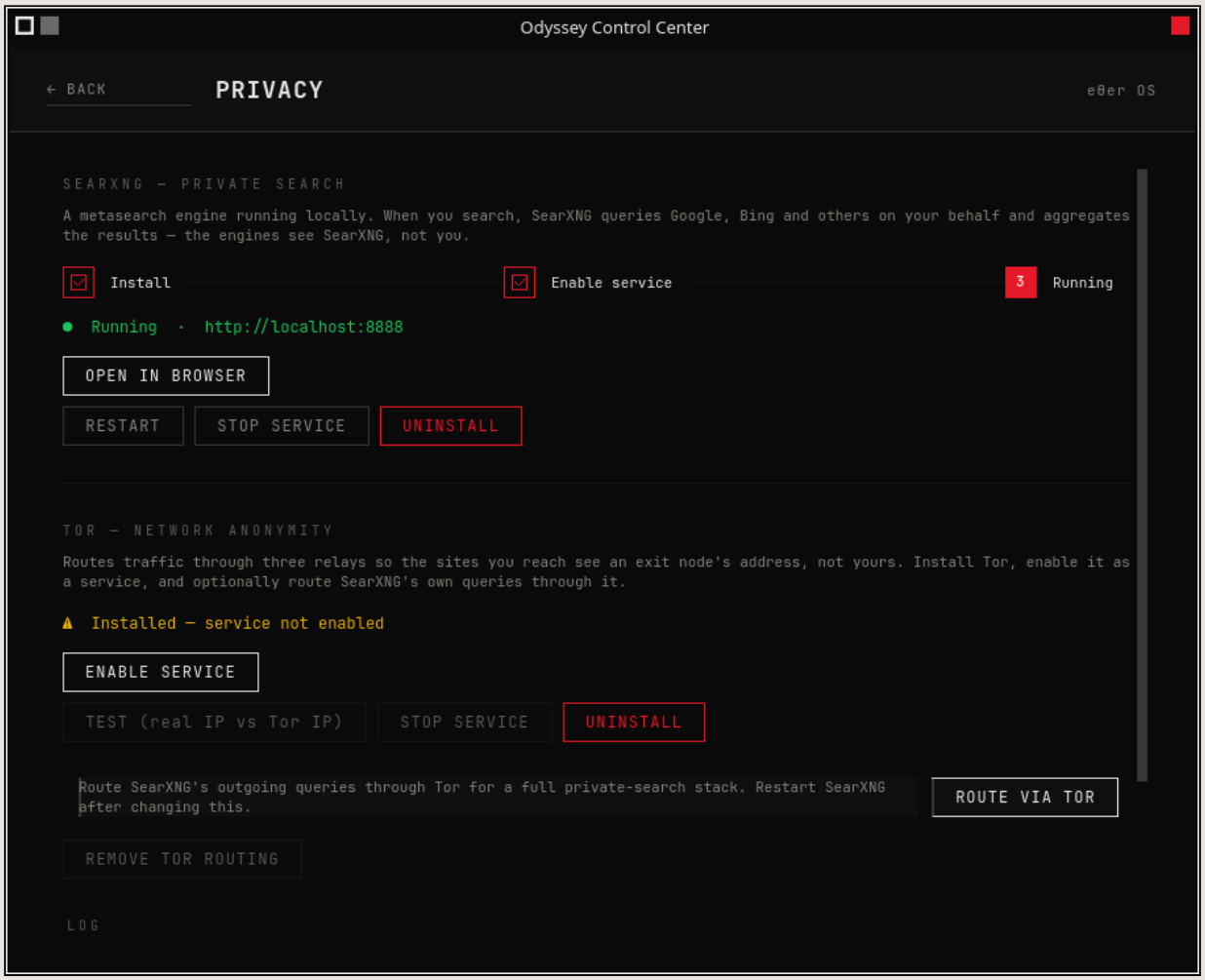

FIG · Tor 处于第 2 步：「Installed — service not enabled」，ENABLE SERVICE 已激活，TEST 和 STOP SERVICE 仍处于禁用状态。

3 Running，并对其进行验证

启用之后，会出现一个专门的 **TEST (real IP vs Tor IP)** 按钮——使用它。它会并排比较你的真实 地址和 Tor 呈现出的地址，让你获得它确实在起作用的真实证据，而 不只是一个只能被动相信的状态指示灯。

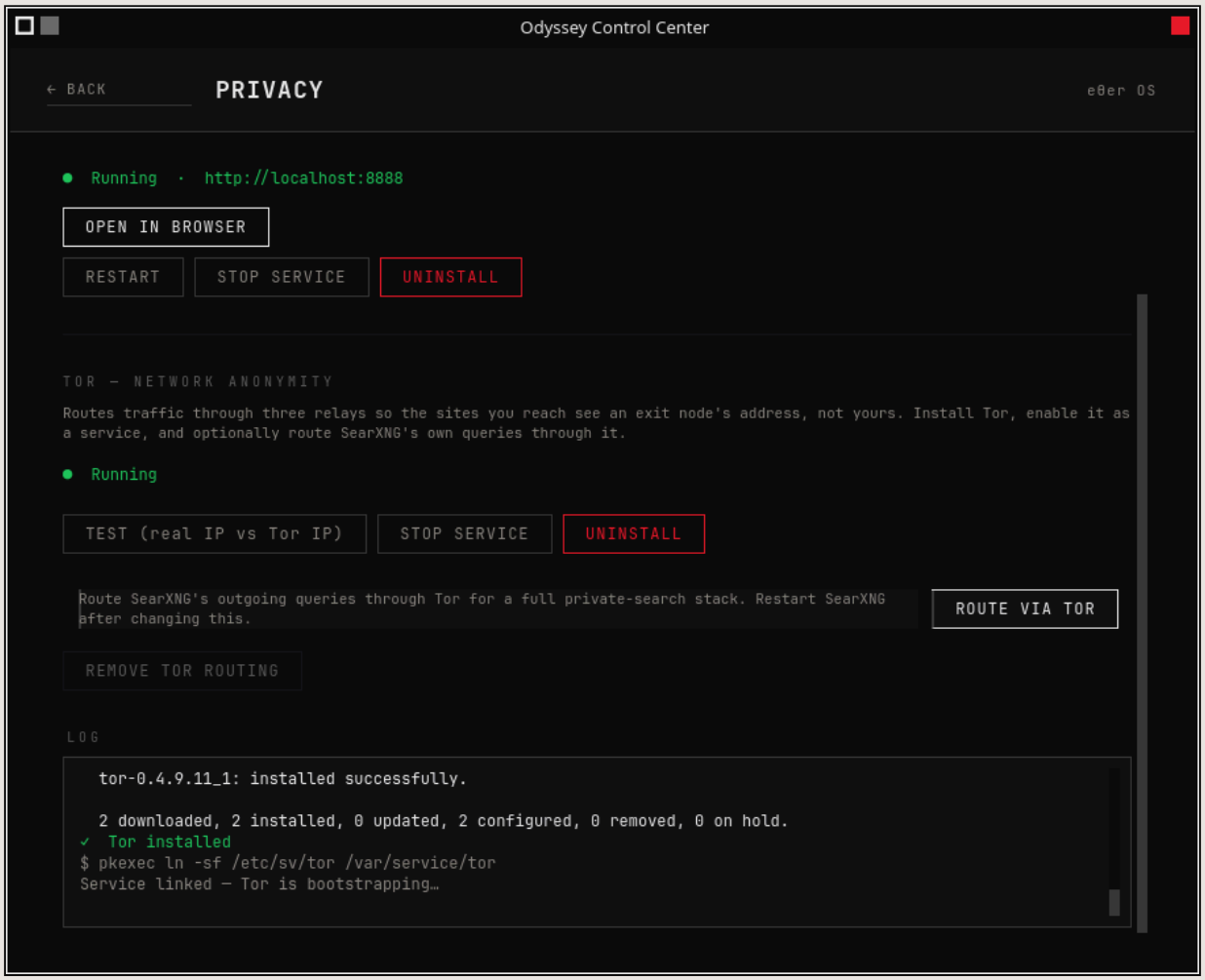

FIG · 两个部分都是绿色且正在运行：SearXNG 位于 localhost:8888，配有 OPEN IN BROWSER；Tor 正在运行，TEST / STOP SERVICE / UNINSTALL 全部处于激活状态。

WHAT	Privacy → INSTALL TOR → ENABLE SERVICE → TEST (real IP vs Tor IP) 进行确认。
WHY	把敏感的浏览行为通过 Tor 转发，可以对你访问的网站隐藏你的 真实地址——TEST 为你提供的是证据，而不仅仅是一句断言。
RESULT	一个正在运行的 Tor 服务，其他应用程序可以通过它进行 转发；如果一切正常，TEST 会显示两个确实不同的地址。
RISK	通过 Tor 的流量明显更慢——这是匿名性所付出的代价。STOP SERVICE 会暂停它；UNINSTALL 会移除它。

7.3 图解指南——将两者结合起来

一旦两者都在运行，就会出现一条内嵌提示，提议把 SearXNG **自身**的 查询也通过 Tor 转发——不只是你发给 SearXNG 的请求，还有 SearXNG 代表你发给 Google 和 Bing 的请求。

WHAT	Privacy → 「Route SearXNG's outgoing queries through Tor」 提示 → ROUTE VIA TOR → 在 SearXNG 部分点击 RESTART 使其 生效。
WHY	如果不这样做，SearXNG 本身仍然会直接向各大搜索引擎发出 查询——对你这一侧而言是私密的，但对它们那一侧而言仍是可见的。 把 SearXNG 自身的流量也通过 Tor 转发，弥补了这个缺口，构成一个 完全私密的搜索链路。
RESULT	SearXNG 的出站查询，而不仅仅是你发给它的查询，都会经过 Tor。
RISK	由于同样的原因——任何经过 Tor 转发的流量都会变慢——结果的 加载会明显更慢。REMOVE TOR ROUTING 可以撤销这一设置，同样需要 随后重启服务。

▸ ADVANCED

Past this point: config file paths, raw commands, and what the panel is doing underneath.

7.4 Advanced

INSTALL SEARXNG 实际执行的内容

```
$ pkexec /usr/bin/odyssey-searxng-setup
# 创建一个专用的系统用户 'searxng',
# 将源代码克隆到 /usr/local/searxng/searxng-src,
# 构建一个 Python venv 并用 pip 安装依赖
```

服务与端口，手动操作

```
# SearXNG 自己的 runit 服务
$ sudo sv status searxng
$ sudo sv restart searxng
# Tor 的 runit 服务——面板的启用步骤本质上就是这个
$ sudo ln -sf /etc/sv/tor /var/service/tor
$ sudo sv status tor
# 确认 Tor 确实在进行转发
$ curl -socks5 localhost:9050 https://check.torproject.org/api/ip
```

手动将 SEARXNG 通过 TOR 转发

Tor 的 SOCKS 代理默认监听在 `127.0.0.1:9050`。要让 SearXNG 通过它 转发，意味着要把它的出站 HTTP 客户端指向这个代理——这项设置存在于 SearXNG 自己的 `settings.yml` 文件中，位于出站请求配置部分，而 不是某个系统级的代理变量中。面板的 ROUTE VIA TOR 按钮会直接编辑 这个文件并重启服务；手动完成的话，意味着要找到并编辑 `/usr/local/searxng/searxng-src/searx/settings.yml` 中同样的 `outgoing:` 代码块。

> **为什么路由更改之后 SEARXNG 需要重启**
>
> SearXNG 只在启动时读取一次它的代理配置——这不是一个可以在运行时 热重载的设置。无论你是通过面板还是手动更改它，这一点都成立， 这也是为什么两条路径都以同样的 RESTART 步骤结束。

此面板涉及的文件

路径	作用
/usr/local/searxng/searxng-src/	SearXNG 自身的安装内容——源代码、venv、配置
/etc/sv/searxng、/etc/sv/tor	runit 服务定义
/var/service/	启用某个服务时，它会被链接进正在运行的监督树中的位置

CHAPTER 8

Containers

从头到尾的容器管理：引擎设置、正在运行的容器列表、一个用于起步的 精选目录、针对目录中没有的内容的原始命令，以及磁盘占用情况——全部 都在一个可滚动的面板中。

8.1 图解指南——引擎

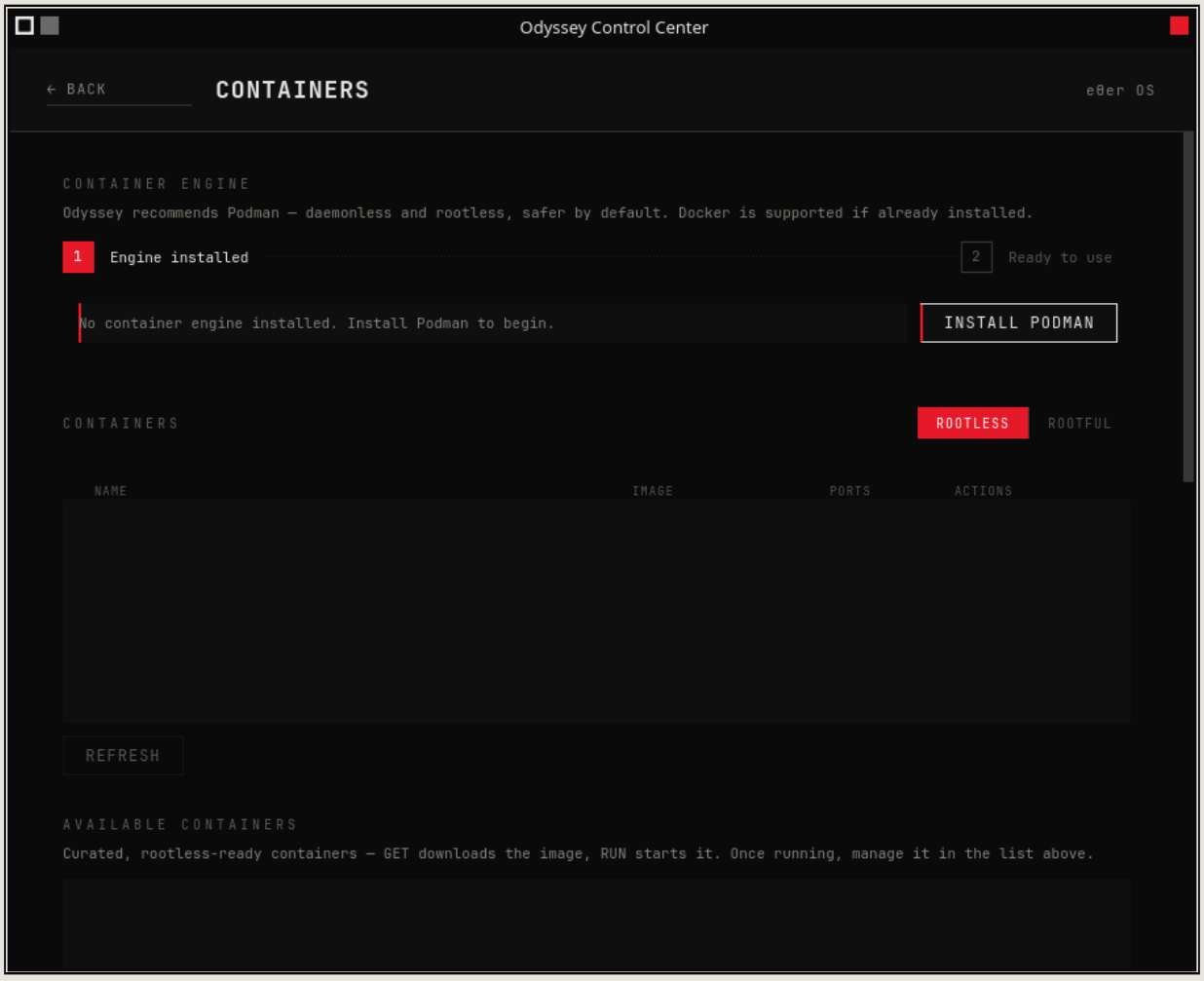

FIG · 第 1 步，共 2 步：「No container engine installed. Install Podman to begin」，INSTALL PODMAN 是唯一激活的操作。下方的 Containers list 和 Available containers 部分可见，但处于空 状态。

Odyssey 推荐使用 **Podman**——无守护进程、rootless，这意味着容器 运行时不需要一个常驻的后台服务，并且默认情况下没有 root 权限。如果 你已经在使用 **Docker**，它同样受支持。

1 **INSTALL PODMAN**

日志显示真实的软件包安装过程。

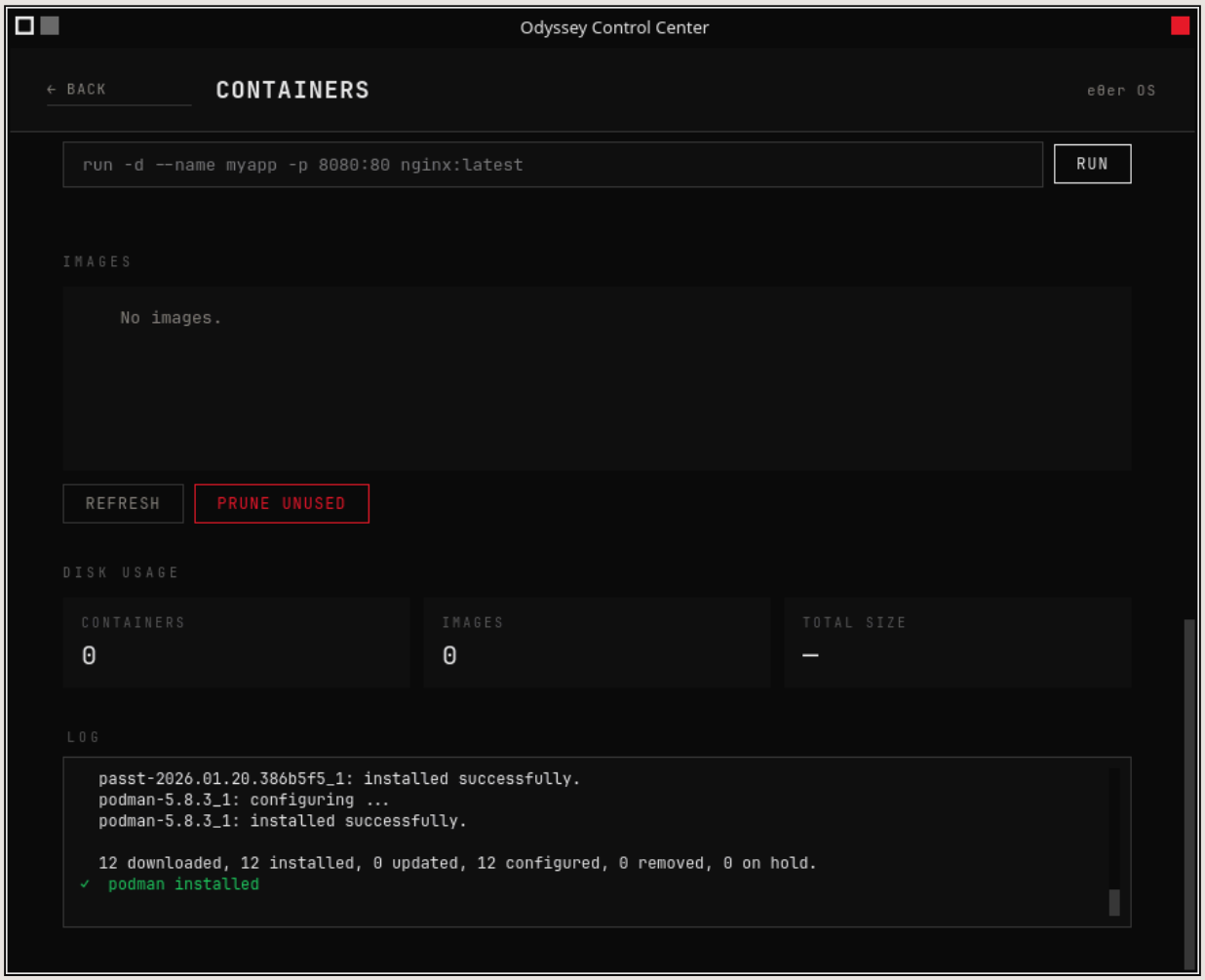

FIG · 安装完成后：「Engine installed」以红色打勾，进入第 2 步「Ready to use」，信息行显示「podman version 5.8.3 · Podman installed · Docker not installed」，还有一个 INSTALL DOCKER TOO 按钮，供 同时想要 Docker 的用户使用。

2 Ready to use

引擎已激活。如果你想让两个引擎并存， **INSTALL DOCKER TOO** 仍然可用——它们能够互不冲突地共存。

8.2 图解指南——rootless vs. rootful，以及空状态

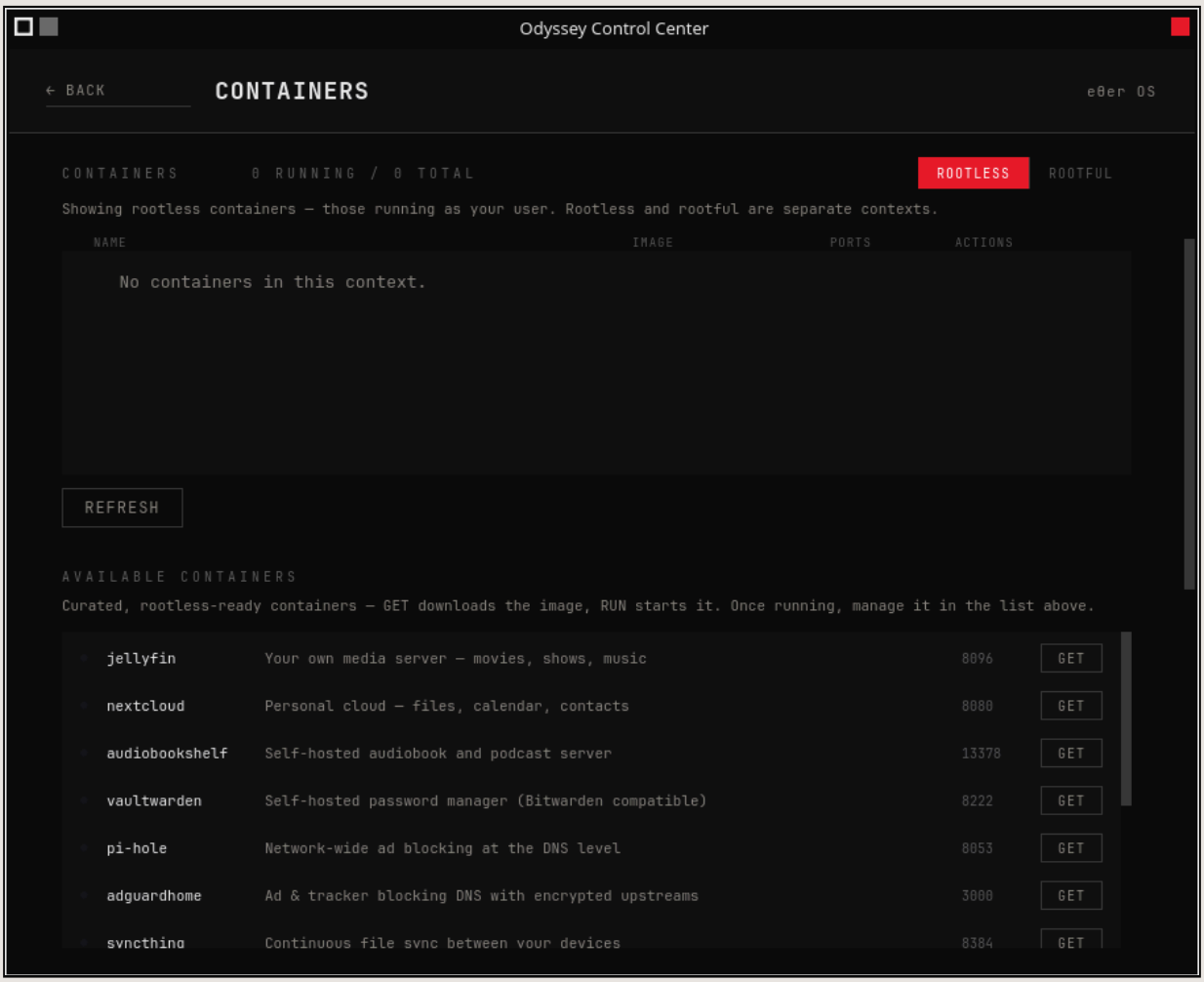

FIG · ROOTLESS 被选中（以红色高亮），旁边是 ROOTFUL，「0 RUNNING / 0 TOTAL」，NAME / IMAGE / PORTS / ACTIONS 列标题下方显示「No containers in this context.」。

顶部的 **ROOTLESS / ROOTFUL** 开关会切换你正在查看和管理的上下文—— 这两者是完全独立的容器命名空间，而不是对某个合并列表的过滤器。 Odyssey 默认显示 ROOTLESS，这个默认选择是刻意为之的：

> **你应该使用哪一个**
>
> Rootless 是更安全的选择，也是面板引导你使用的方向：一个在 rootless 模式下逃脱限制的容器，反正也做不了任何你自己的用户账户 本来就做不到的事情。Rootful 是为那些确实需要它的特定场景而存在 的——某些编号较低的网络端口、某些硬件访问需求——它应该是例外， 而不是习惯。

8.3 图解指南——精选目录，从头到尾

滚动到 **Available containers**——一份已经准备好的列表，这样面板 就永远不会是一个空盒子，等着你已经背下所有镜像名称才能使用。每一行 都显示该容器、一行说明，以及它所使用的端口。

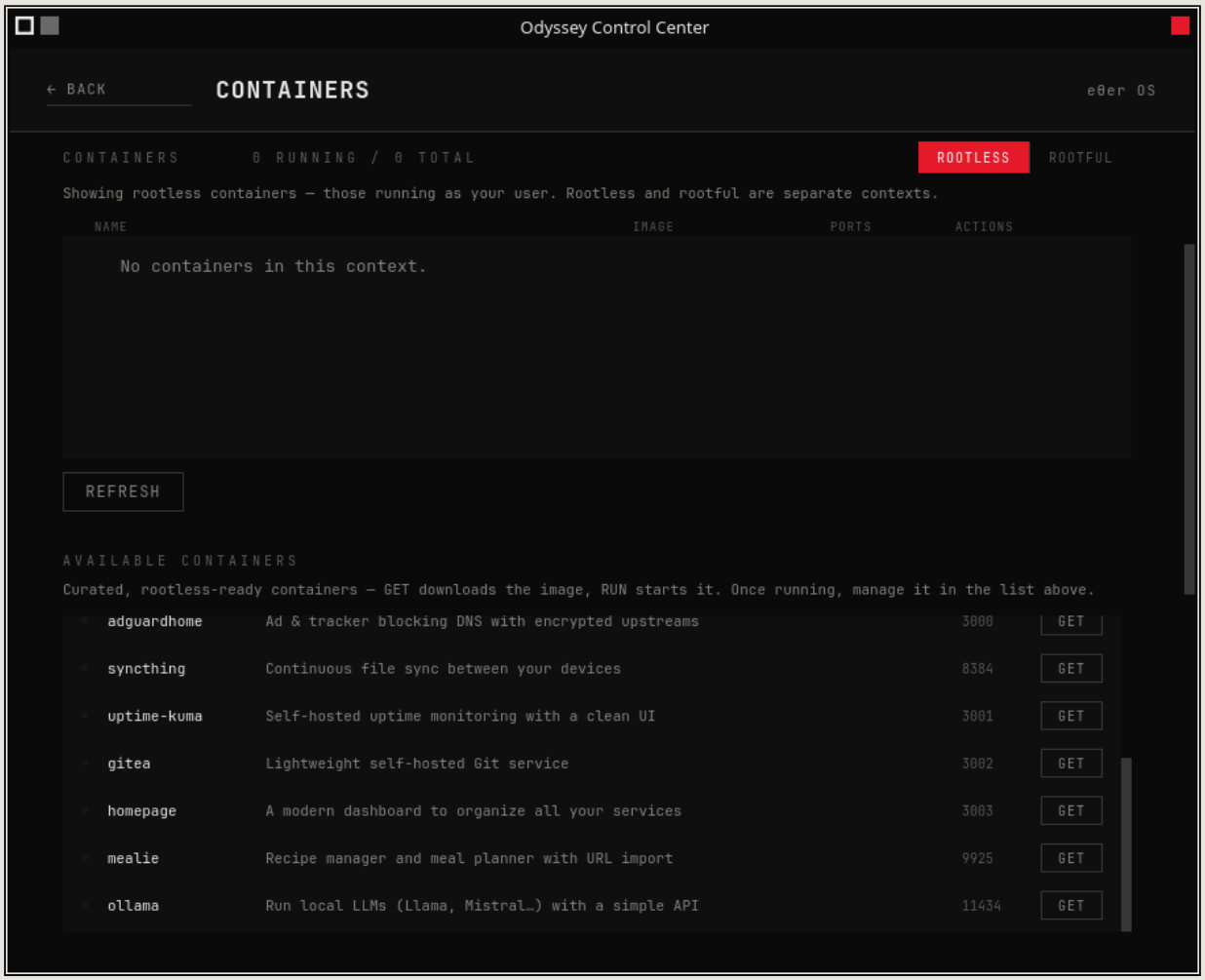

FIG · 目录列表：adguardhome（端口 3000）、syncthing（8384）、 uptime-kuma（3001）、gitea（3002）、homepage（3003）——每一行都有一个 GET 按钮。

这个端口号就是该服务运行起来之后你将用来访问它的端口——例如 homepage 配合 3003，意味着一旦激活，就是 http://localhost:3003。

1 GET

在 homepage 上点击 GET。这只会下载容器镜像—— 此时还没有任何东西在运行。

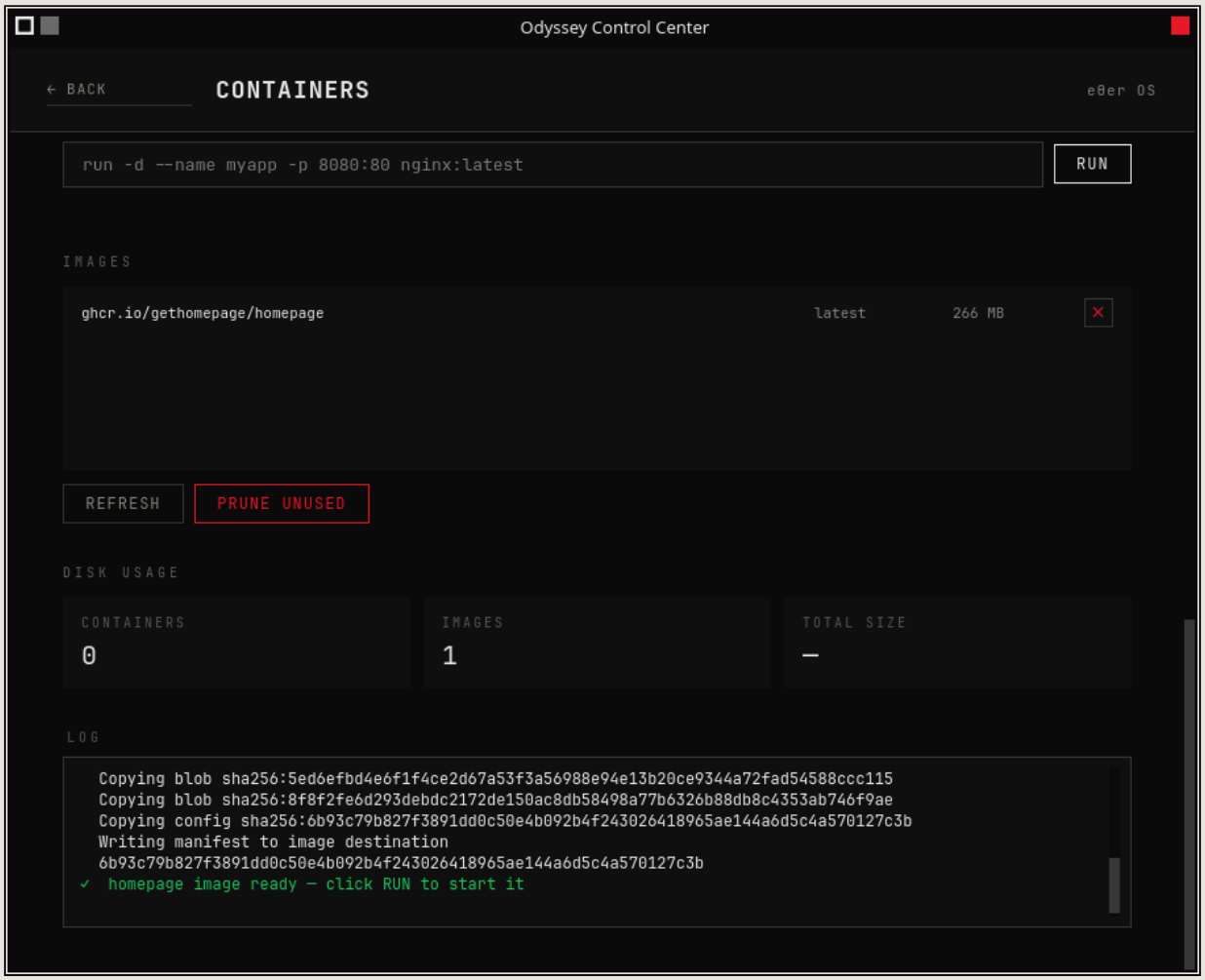

FIG · 日志正在传输真实的镜像下载过程：「Copying blob sha256:5ed6…」，「Writing manifest to image destination」， 最后以绿色结尾——「homepage image ready — click RUN to start it」。

2 RUN

GET 按钮所在的那一行现在改为显示 RUN——点击它， 真正启动这个容器。

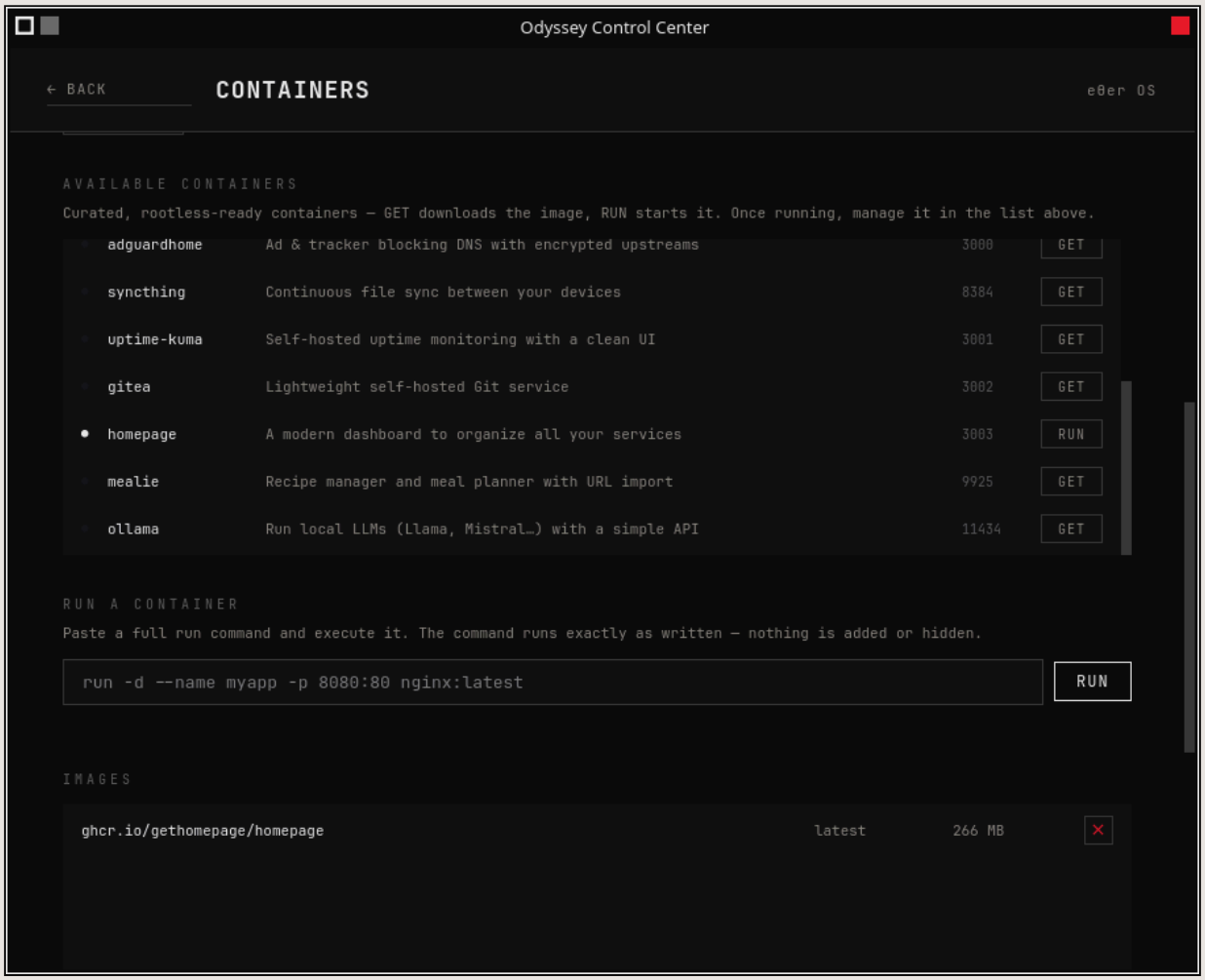

FIG · homepage 现在出现在上方的 CONTAINERS 部分，绿色圆点，镜像路径 ghcr.io/gethomepage/homepage，端口「0.0.0.0:3003→…」，拥有自己的一行操作图标；在下方的 Available Containers 中依然可见，并带有一个新的 RUN 按钮， 用于启动另一个实例。

一旦运行起来，homepage 会同时出现在两个地方：作为上方 **Containers** 列表中的一条激活记录（带有它的端口和操作图标），以及 仍然留在下方目录中，以备你之后想要再启动第二个实例。

WHAT	Containers → Available containers → 在目录中的某项上点击 GET → 下载完成后点击 RUN。
WHY	分成两个步骤（先下载，再启动）意味着你可以提前获取一个 镜像，而不必立即承诺运行它——在网速较慢，或者你只是在浏览有 哪些可用内容时很有用。
RESULT	一个正在运行的容器，出现在主列表中，带有它的端口映射， 从那时起可以在那里进行管理。
RISK	较低——精选目录中的自托管服务经过挑选，默认对 rootless 友好且风险较低。它的暴露面与运行任何一个 web 服务是一样的：无论它监听哪个端口，都要依照你的 Firewall 设置来决定是否可以被访问。

8.4 图解指南——容器列表及其操作

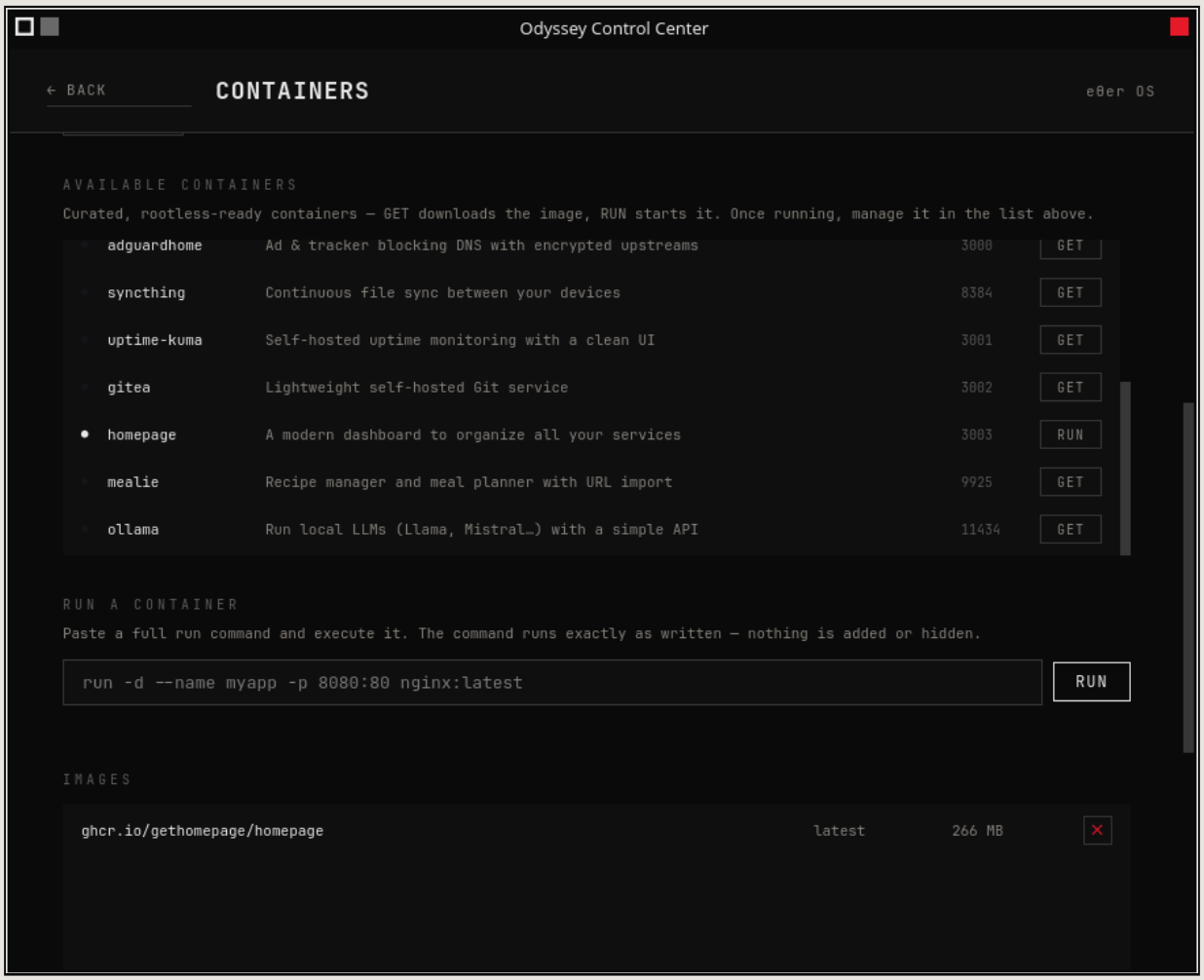

FIG · Containers 列表中 homepage 所在的行，右侧可见它的五个操作 图标——下面会一一解释。

每个正在运行的容器所在的行末尾都有一组图标——把鼠标悬停在任意一个 上面都能看到它的标签，以下是每个图标的作用：

图标	操作
打开（箭头）	直接在浏览器中打开该服务——一旦有东西在运行，这是最快的方式
停止 / 启动（方块 / 三角形）	在不移除容器的情况下将其停止，或者重新启动它
重启（环形箭头）	一次操作中先停止再启动——用于「正在运行但不响应」情况的解决方案
日志（线条图标）	打开该容器自己的日志输出——如果有什么东西不工作，这是应该首先查看的地方
移除（×）	彻底移除该容器——镜像本身仍会保留已下载状态，单独列在 Images 中

8.5 图解指南——运行一个容器，并打开它

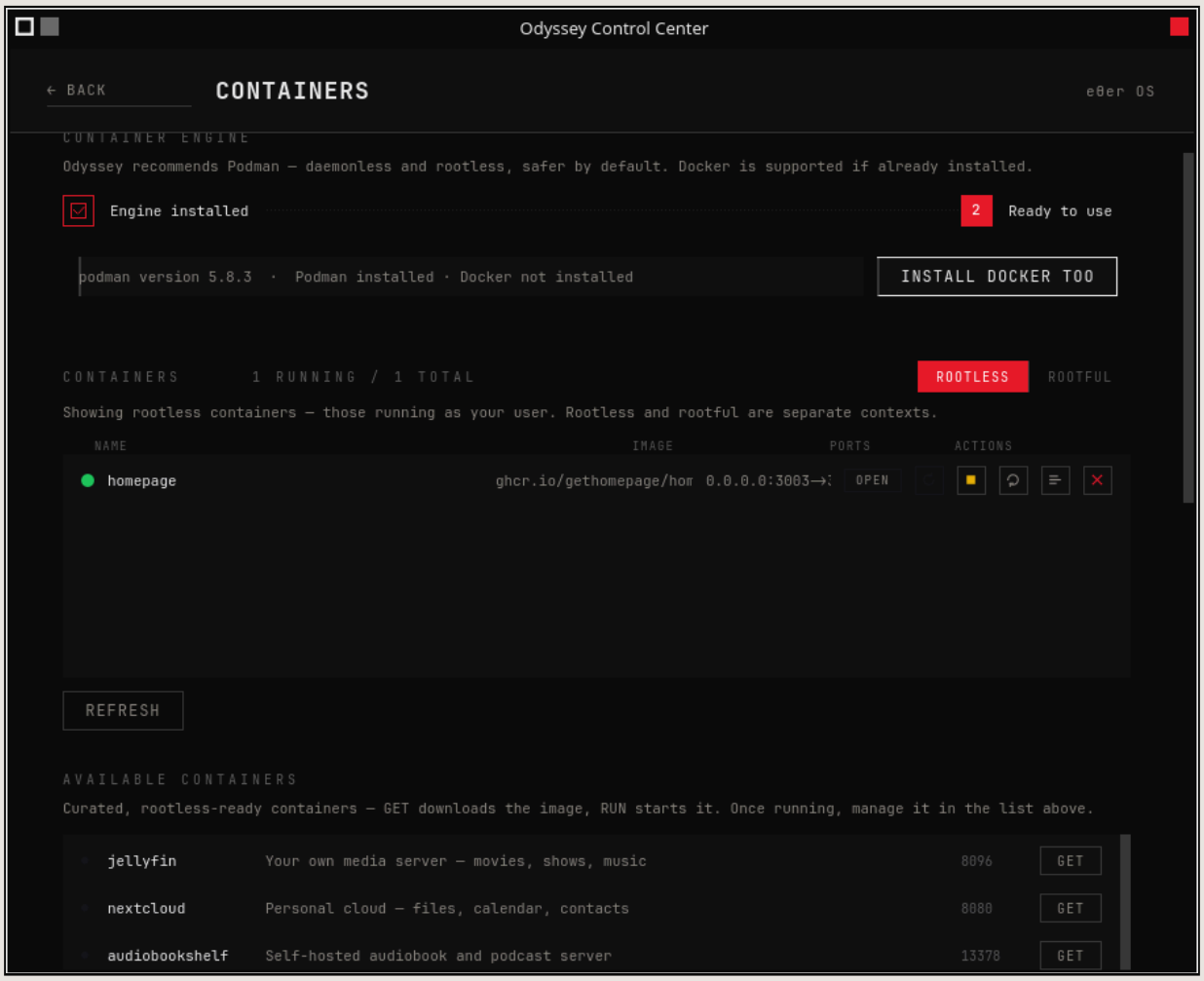

FIG · 「Run a container」字段中显示一条示例命令作为占位提示文字， RUN 按钮，下方的 Images 部分列出 ghcr.io/gethomepage/homepage · latest · 266 MB，附有一个移除 按钮。

对于目录中没有的任何内容，**Run a container** 会接受一条完整的 `podman run`（或 `docker run`）命令，并按它写出的样子精确执行—— 不会添加任何内容，也不会隐藏任何内容。

WHAT	Containers → 「Run a container」→ 粘贴一条完整的 run 命令（例如 `run -d --name myapp -p 8080:80 nginx:latest`）→ RUN。
WHY	精选目录覆盖了常见的使用场景；这是针对目录未提供的特定 镜像、特定参数、特定端口的退路。
RESULT	容器会精确按照命令指定的方式启动，一旦运行起来就会加入 主 Containers 列表。
RISK	取决于命令本身做了什么——这是一次忠实的传递，而不是一个 沙盒。请以在终端中输入同一条命令那样的谨慎态度来对待它。

下方的 **Images** 会列出每一个已下载的镜像，无论当前是否有容器正在 使用它，配有 **REFRESH** 和 **PRUNE UNUSED**，用于从不再被任何东西 引用的镜像中回收空间。

打开一个正在运行的容器

在正在运行的容器所在的行上点击 OPEN——与上方图标行中相同的打开 操作——会直接把你的默认浏览器带到它的地址和端口，例如 `homepage` 对应的 `http://localhost:3003`。一旦它在运行，你就 不需要自己记住端口号了。

8.6 磁盘占用

面板最底部，是三个实时汇总数字——容器数量、镜像数量、总大小——让你 无需打开终端就总能知道容器在磁盘上占用了多少空间。

```
▸ ADVANCED

Past this point: config file paths, raw commands, and what the panel is doing underneath.
```

8.7 Advanced

命令行等效操作

```
# 面板中所说的「GET」
$ podman pull ghcr.io/gethomepage/homepage
# 面板中对目录条目所说的「RUN」
$ podman run -d -name homepage -p 3003:3000 ghcr.io/gethomepage/homepage
# 列出正在运行的容器——rootless 上下文
$ podman ps
# 同样的操作，rootful
$ sudo podman ps
# 日志、停止、重启、移除——图标行，手动操作
$ podman logs -f homepage
$ podman stop homepage
$ podman restart homepage
$ podman rm homepage
# 镜像与磁盘占用
$ podman images
$ podman system df
$ podman image prune
```

ROOTLESS VS. ROOTFUL，底层原理

Rootless Podman 会以你自己的用户身份运行容器，处于一个用户命名 空间之内——容器内的「root」被映射到主机上一段没有特权的 UID 范围， 因此即使容器进程被完全攻破，它依然被限制在你自己账户的权限之内。Rootful Podman（`sudo podman ...`）以传统方式运行，容器的 root 被 映射到主机的真实 root。面板上的这个开关，实际上只是在运行普通的 `podman` 还是 `sudo podman` 之间切换——两个上下文各自拥有真正独立 的状态目录、独立的容器列表，默认情况下彼此之间没有任何共享。

```
# 每个上下文各自把自己的状态保存在哪里
$ podman info -format '{{.Store.GraphRoot}}'
# rootless:  /.local/share/containers/storage
# rootful: /var/lib/containers/storage
```

为什么有些目录条目需要开放特定端口

一个容器的端口映射（在 homepage 示例中是 `3003:3000`）默认**只**让 该服务能从本机访问——除非另有指定，否则 Podman 会将其绑定到 `localhost`。要从你网络中的另一台设备访问它，还需要在 Firewall 面板（第 4 章）中开放这个端口——这两个系统是相互独立的层级，一个 带有端口映射的正在运行的容器，并不会仅仅因为 Podman 在监听它，就 自动可以从互联网或局域网访问到。

CHAPTER 9

Gaming Mode

顶部有一个主开关，一行实时状态，以及四个标签页：**PACKAGES**、**GAMEMODE**、**MANGOHUD**、**PROTON**。

9.1 图解指南——主开关

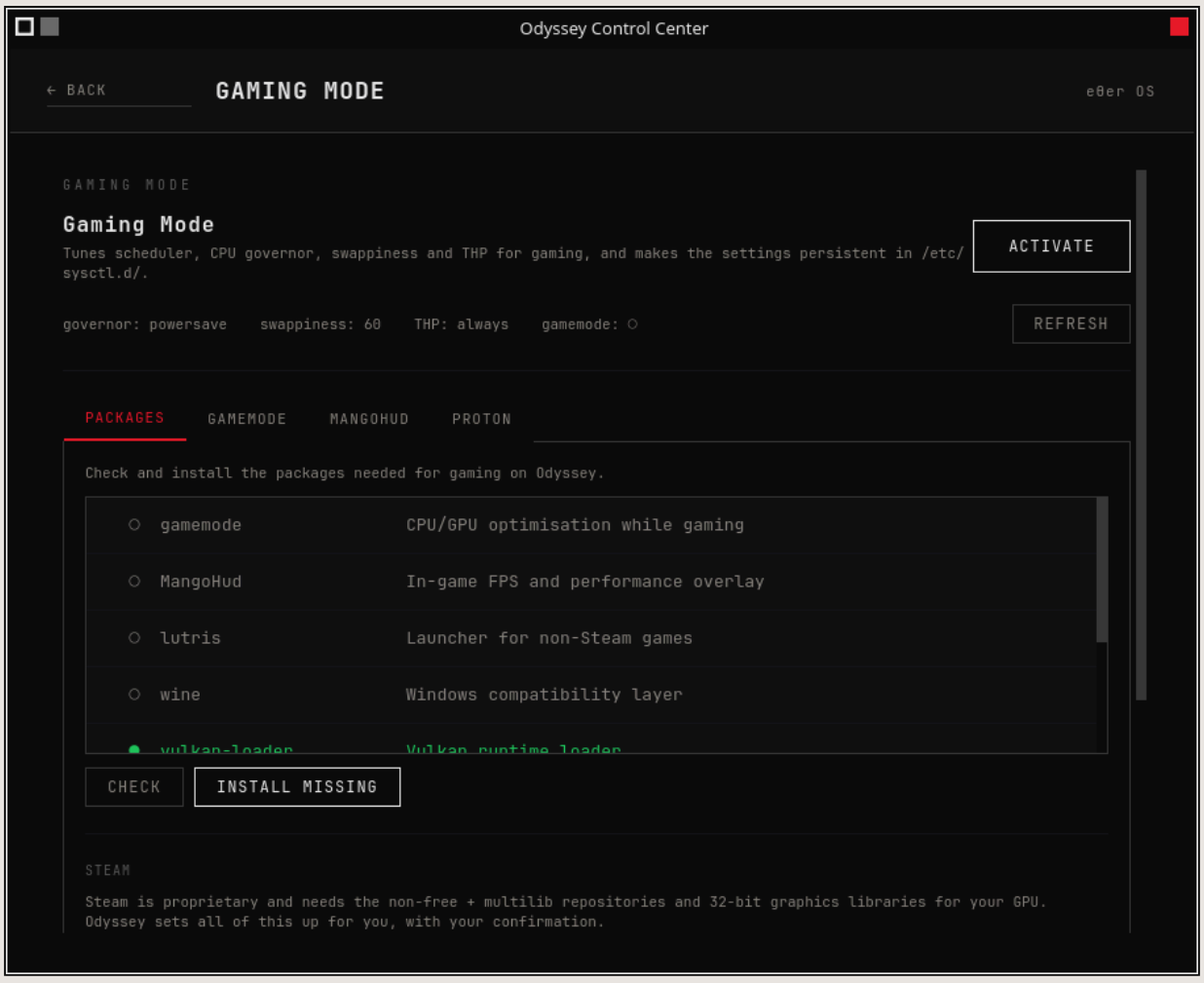

FIG · 面板的顶部区域：「Gaming Mode — Tunes scheduler, CPU governor, swappiness and THP for gaming, and makes the settings persistent in /etc/sysctl.d/」，一个 ACTIVATE 按钮，一个 REFRESH 按钮，以及 实时状态行：governor powersave, swappiness 60, THP always, gamemode off。

这行状态显示的是与 Kernel 和 Memory 面板相同的实时数值——governor、swappiness、THP 状态，以及 `gamemode` 服务是否在运行——这样你就能 一眼确认激活操作是否真正生效，而不必分别去查看那两个面板。

WHAT	Gaming Mode → ACTIVATE。
WHY	手动前往 Kernel 和 Memory，手工设置一个适合游戏的 governor、swappiness 和 THP 值，正是那种一个专门的模式应该 消除掉的多步骤麻烦事。
RESULT	Scheduler、governor、swappiness 和 THP 全部切换到适合 游戏的值，立即生效并被持久化——状态行会更新以反映这个变化。
RISK	较低，且可逆——就是你原本要在 Kernel/Memory 中手动设置的 同样那些值，只是被打包进了一次点击。

9.2 图解指南—Packages 与 Steam

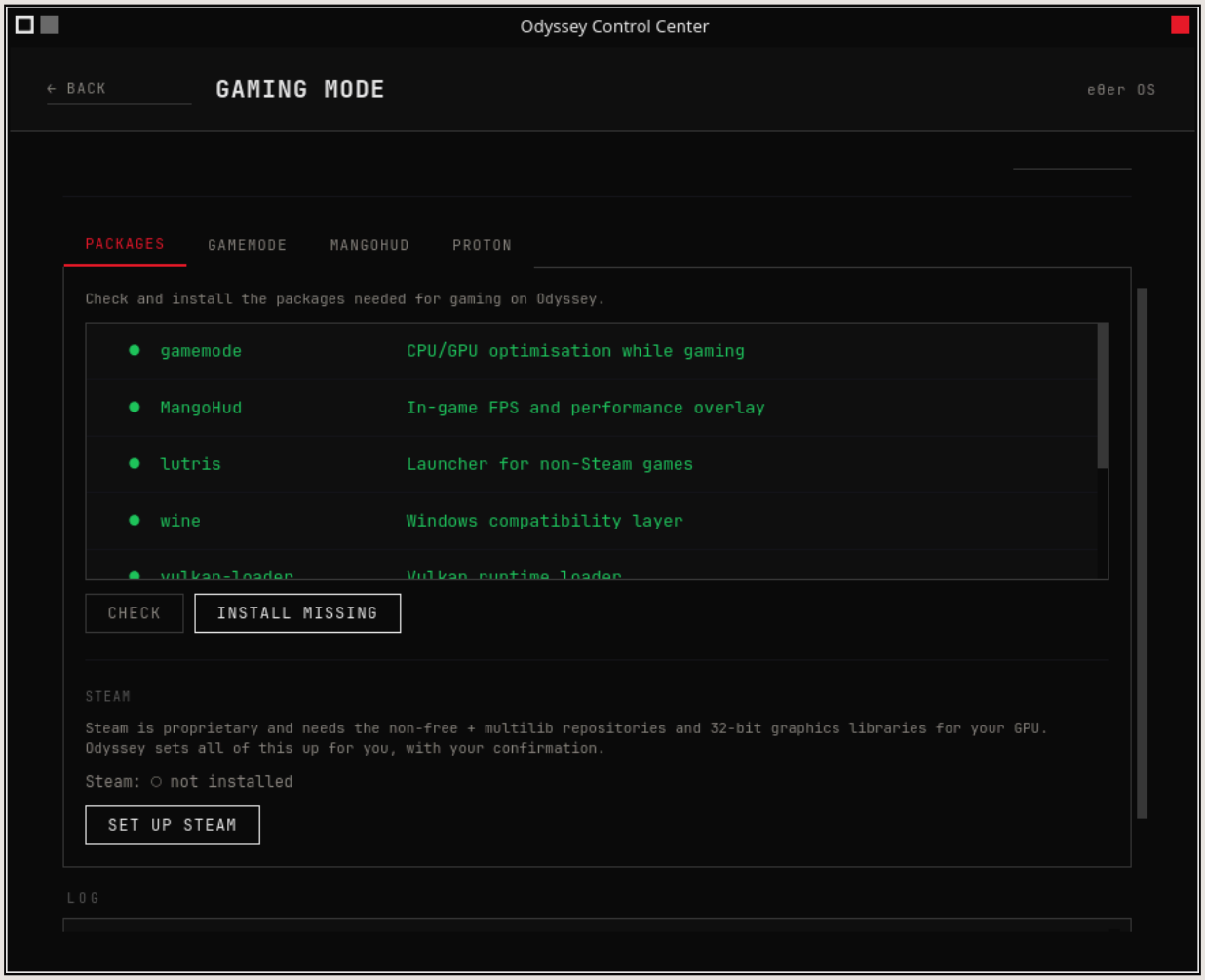

FIG · PACKAGES 标签页：gamemode、MangoHud、lutris、wine、 vulkan-loader 全部都是实心绿色圆点（已安装）；上方是 CHECK 和 INSTALL MISSING；下方的 Steam 部分显示「Steam: ● installed」， 配有 REINSTALL STEAM 按钮。

一份游戏通常所需软件包的检查清单——CHECK 能看出缺少什么，INSTALL MISSING 可以一次性补齐这些缺口。一旦一切都显示为实心绿色圆点， 就像截图那样，你就已经完全配置好了。

Steam 在检查清单下方拥有自己专属的流程，因为它需要的不只是一次 简单的软件包安装。

1 SET UP STEAM

点击它——此时还不会安装任何东西，会先 打开一个提供说明的确认对话框。

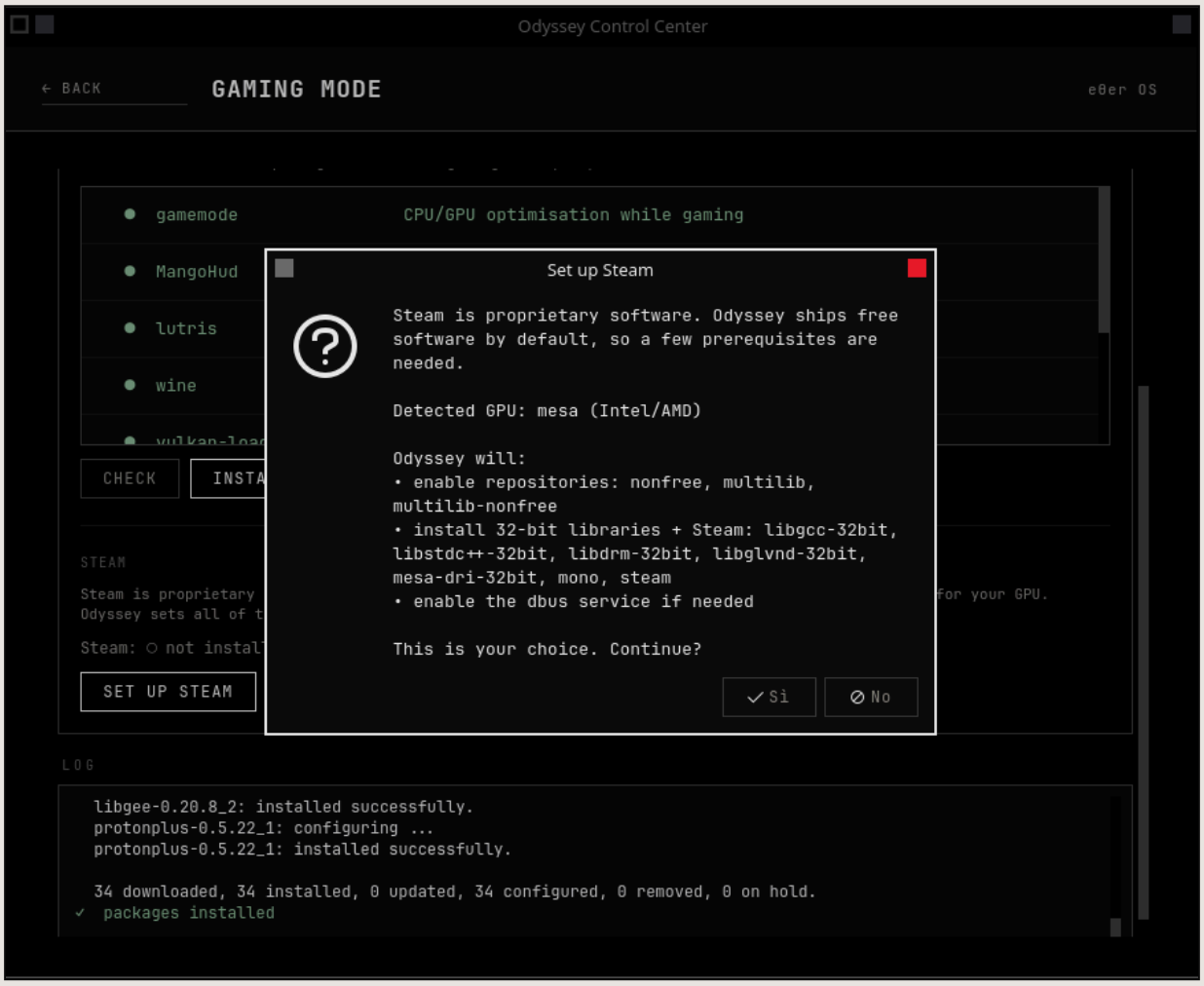

FIG · 确认对话框：「Steam is proprietary software. Odyssey ships free software by default, so a few prerequisites are needed. Detected GPU: mesa (Intel/AMD).」一份项目符号列表，列出了将会 具体启用和安装什么，最后是「This is your choice. Continue?」， 配有 Sí / No 按钮。

在你同意之前，这个对话框会准确告诉你即将发生什么：哪些仓库会被 启用（nonfree、multilib、multilib-nonfree），根据你**检测到** 的 GPU 会安装哪些 32 位库，以及如果 dbus 服务尚未启用，它将会被 启用。没有任何东西被隐藏在这个确认之后。

2 **Sí**

确认它。日志会传输真实的安装过程——软件包下载、 配置、完成。

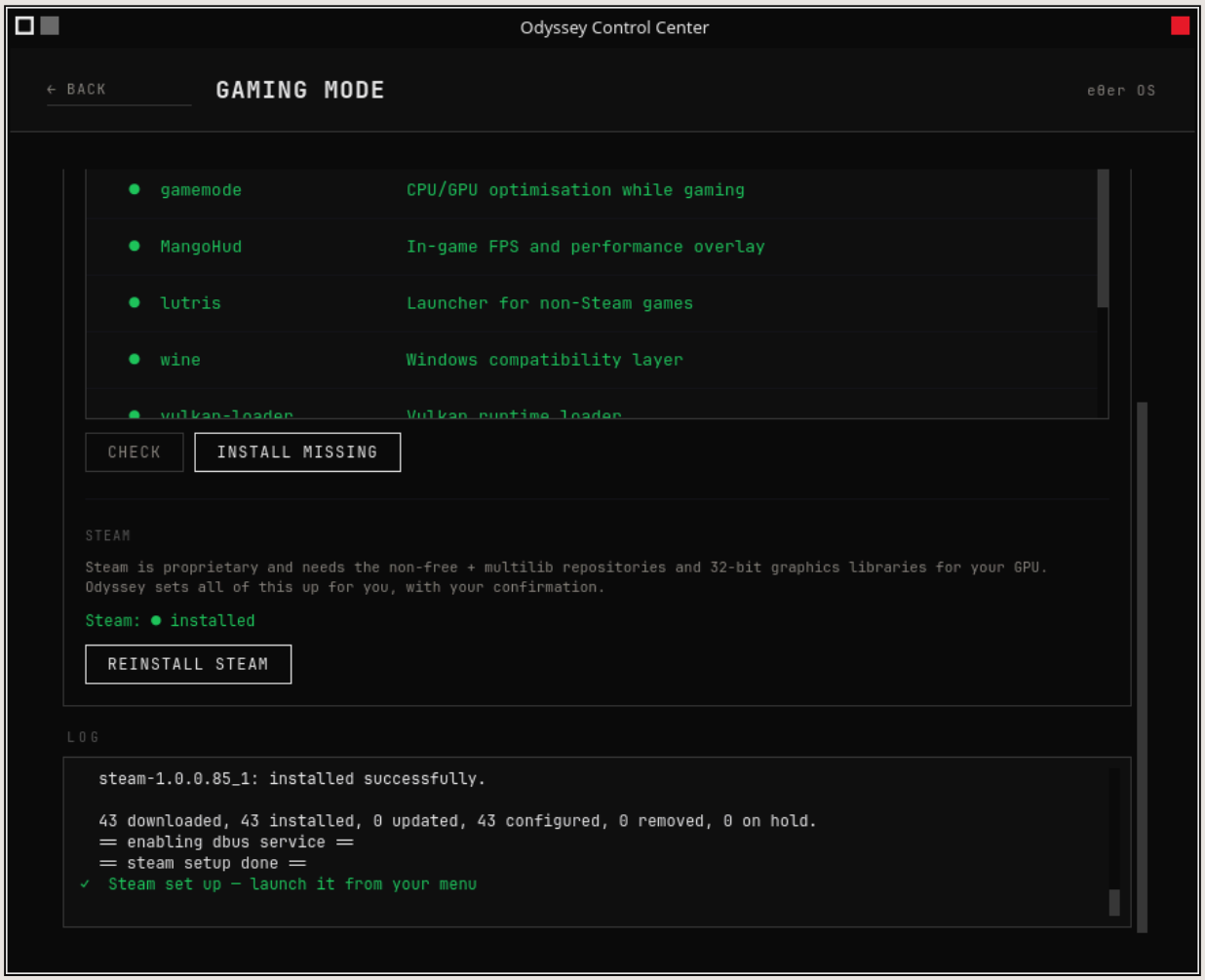

FIG · 配置完成后的 PACKAGES 标签页：所有软件包都是绿色的，Steam 显示「● installed」，配有 REINSTALL STEAM，日志最后写着「Steam set up — launch it from your menu.」

WHAT	Gaming Mode → PACKAGES 标签页 → SET UP STEAM → 查看对话框 → Sí。
WHY	Steam 是专有软件，需要 non-free/multilib 仓库，以及 Void 默认不启用的、针对具体 GPU 的 32 位库——对于刚开始使用基于 Void 的系统的人来说，手动把这一切配置正确是一个常见的障碍。
RESULT	所需的仓库、Steam 本身，以及与你具体 GPU 相匹配的 32 位图形库，全部安装完成——之后就可以从应用程序菜单启动 Steam 了。
RISK	较低——这只会启用仓库并添加软件包；不会移除或重新配置 任何已有的东西。在你确认之前，对话框已经展示了将会发生的 完整清单。

9.3 GameMode 与 MangoHud

GameMode（拥有自己的标签页）是 Feral Interactive 开发的按游戏 性能守护进程——通过它启动的游戏，会仅为自己的进程请求一次临时的 性能提升（CPU governor、I/O 优先级），当游戏关闭时会自动归还。**MangoHud**（拥有自己的标签页）是一个屏幕叠加层，在游戏过程中显示 FPS、帧时间、CPU/GPU 负载和温度。这两个标签页都提供安装/检查 控制，MangoHud 还提供对叠加层外观的快速配置。

9.4 Proton

用于管理通过 Steam 运行 Windows 游戏的兼容层——检查已安装和可用 的内容，并在某个特定游戏需要 Valve 默认不分发的 Proton 版本时（ 包括像 Proton-GE 这样的社区构建版本）安装额外的 Proton 版本。

```
▸ ADVANCED

Past this point: config file paths, raw commands, and what the panel is doing underneath.
```

9.5 Advanced

ACTIVATE 写入的内容

与 Kernel 和 Memory 面板的 MAKE PERSISTENT 操作使用的是同样的底层 文件——Gaming Mode 是一个打包好的组合，而不是一个独立的机制：

```
# 已持久化的 sysctl 值
$ cat /etc/sysctl.d/99-odyssey.conf
# CPU governor——不是 sysctl，基于 sysfs，不会通过这个机制被
# 持久化；如果你依赖它，重启后需要重新应用
$ cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_governor
```

SET UP STEAM 实际执行的内容

确认对话框本身几乎就是一份完整的技术说明，但具体来说，在一台 Mesa（Intel/AMD）系统上：

```
$ sudo xbps-install -y void-repo-nonfree void-repo-multilib void-repo-multilib-nonfree
$ sudo xbps-install -Sy steam libgcc-32bit libstdc++-32bit libdrm-32bit libglvnd-32bit
mesa-dri-32bit mono
# NVIDIA 系统会额外下载 nvidia-libs-32bit 而不是 mesa-dri-32bit——
# 面板会检测你的 GPU 厂商，并相应地调整软件包列表
```

手动操作 GAMEMODE 与 MANGOHUD

```
# 以激活的 GameMode 启动任意游戏
$ gamemoderun ./game_binary
# 或者，对于 Steam 游戏，启动选项：
$ gamemoderun %command%
# MangoHud 叠加层，启动选项：
$ mangohud %command%
# MangoHud 自己的配置文件，用于调整叠加层显示的内容
$ $EDITOR  /.config/MangoHud/MangoHud.conf
```

安装特定版本的 PROTON-GE

PROTON 标签页封装了 `protonplus` 或一个功能相当的版本管理器——手动 操作的等价方式是从 GitHub 上的 `GloriousEggroll/proton-ge-custom` 下载一个发行版，并把它解压到 Steam 的兼容工具目录中：

```
$ mkdir -p  /.steam/root/compatibilitytools.d
$ tar -xf GE-Proton-VERSION.tar.gz -C  /.steam/root/compatibilitytools.d
# 然后在 Steam 中按游戏逐一选择: Properties → Compatibility
```

CHAPTER 10

Telemetry

尽管名字如此，这个面板并不是一个需要你去配置的设置——它是一个实时 监视器。Odyssey 不会收集也不会发送任何关于你的信息，所以这里没有 什么需要关闭的东西。相反，这个面板所做的，是实时向你展示你**自己** 系统正在建立的每一条出站连接——这样「我的机器此刻正在和谁通信」就变成了一个你可以通过亲眼查看来回答的问题，而不是依赖某种说法去 相信。它把自己称为 **Reverse Telemetry**，正是出于这个原因：它把观察这件事翻转了过来，代表你，把它对准了系统本身。

10.1 图解指南

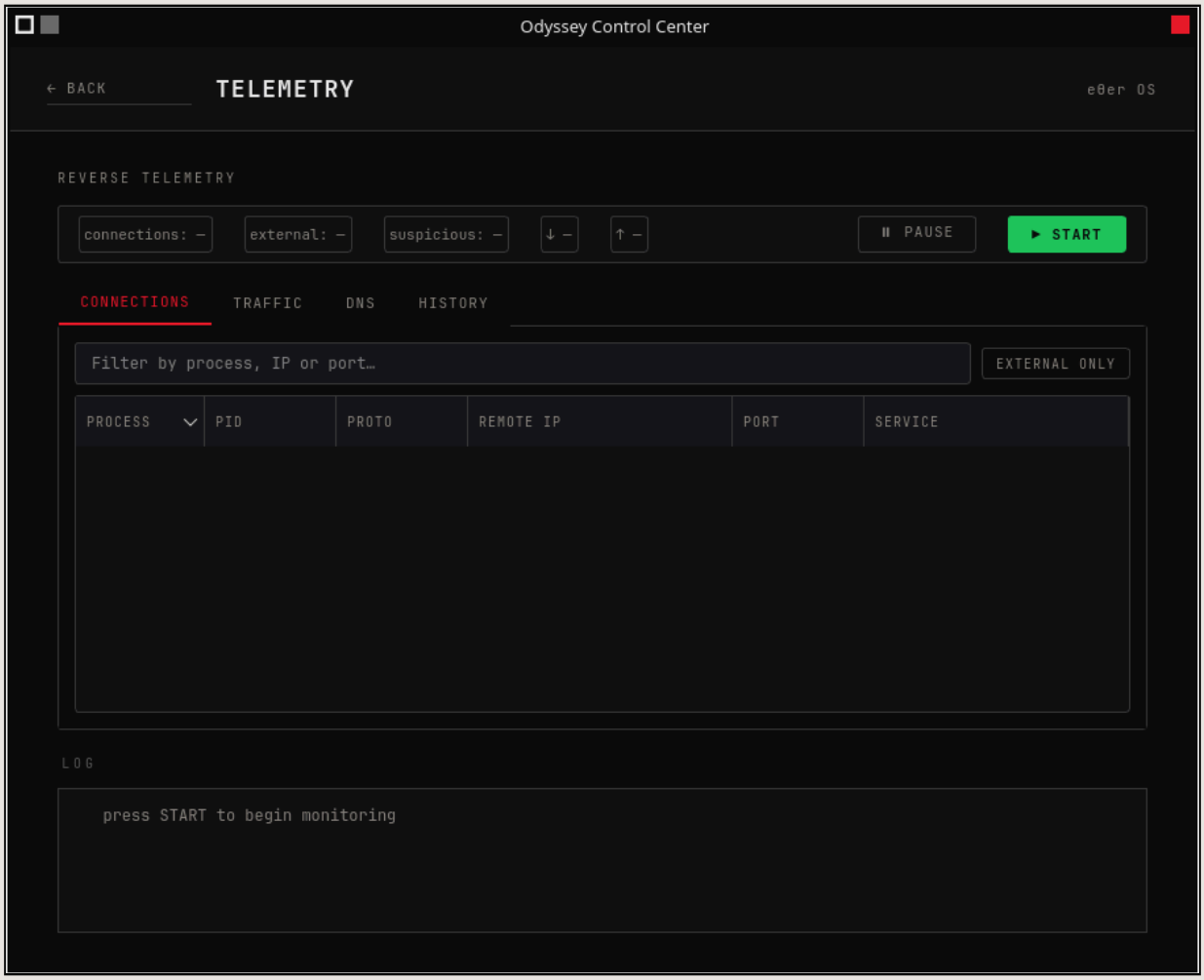

FIG · 启动之前：connections/external/suspicious 全部显示「–」，一个 绿色的 START 按钮，日志写着「press START to begin monitoring」。 四个标签页——CONNECTIONS、TRAFFIC、DNS、HISTORY——都是空的， 上方是一个尚未填充的表格。

1 **START**

开始实时监视——状态栏和当前激活的标签页会立即 填充起来。

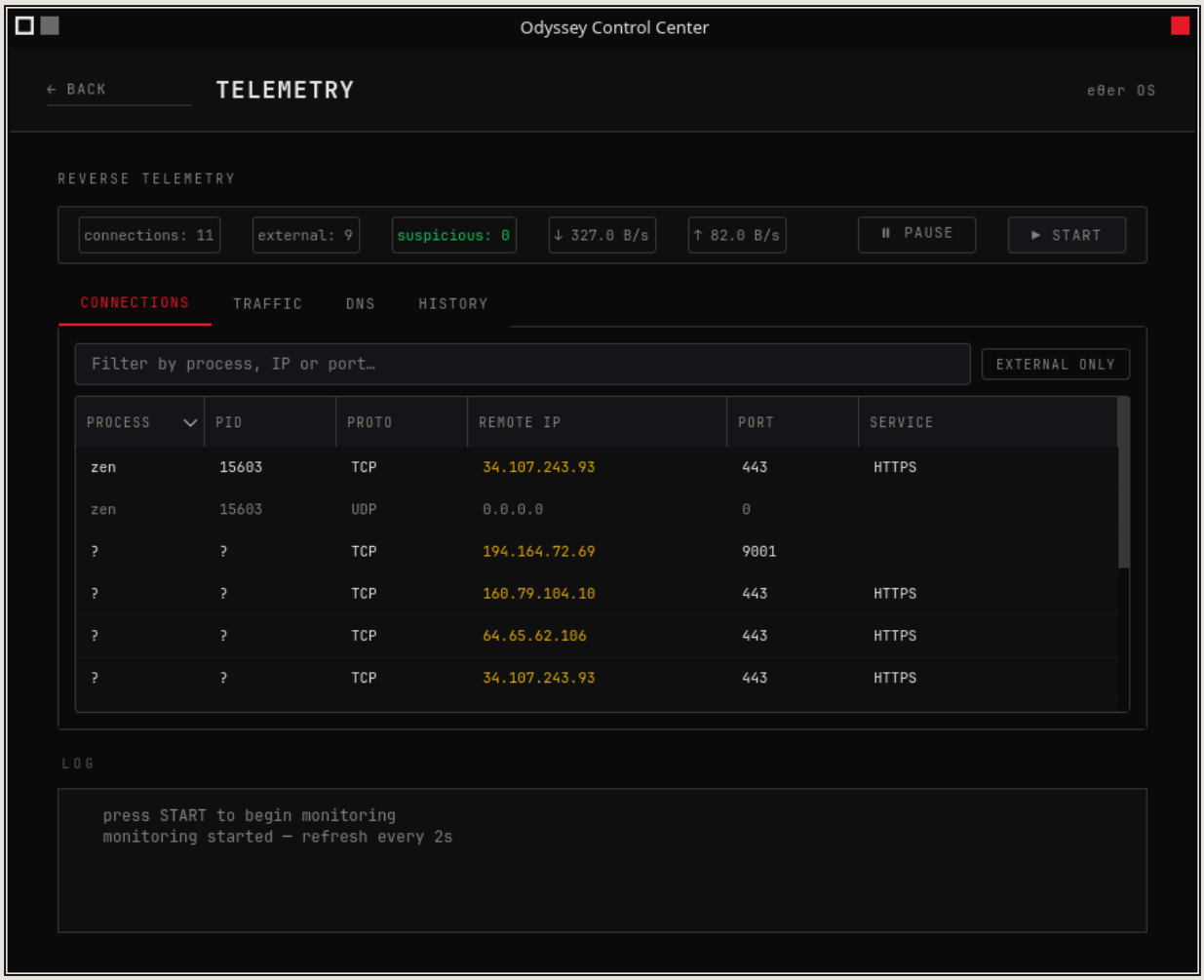

FIG · 启动之后：「connections: 11 · external: 9 · suspicious: 0 · ↓327.0 B/s · ↑82.0 B/s」，PAUSE 现在可用，CONNECTIONS 表格里 填满了真实的行：进程「zen」通过 TCP 连接到 34.107.243.93:443（HTTPS），有几行进程/PID 标记为「?」， 连接到其他地址。

顶部的状态栏提供了实时的汇总数据——连接数、外部连接（相对于本地 连接）、可疑连接、当前的下载/上传速度——**PAUSE** 会冻结当前视图而 不丢失已捕获的数据，**START** 则会恢复它。

四个标签页

标签页	显示内容
CONNECTIONS	每条激活连接对应一行：进程、PID、协议、远程地址、端口，以及该端口所对应的已知服务（443 → HTTPS，22 → SSH）。可以按任意列排序。
TRAFFIC	按接口划分的吞吐量——当前的下载/上传速度，以及自监视开始以来累计的总量。
DNS	域名解析的实时流，随发生随显示——通常比一个赤裸裸的 IP 更清晰，因为主机名告诉你的是**什么**，而不只是**在哪里**。
HISTORY	会话所捕获的全部内容，以可滚动日志的形式呈现——用于捕捉那些在你查看时已经不再是激活连接的短暂连接，非常有用。

进程一栏显示为 ? 的行（截图中可见）意味着这条连接已被捕获，但在 捕获的那一刻，无法根据 PID 解析出源进程——这在生存期极短的连接中 很常见。表格上方的**过滤**字段可以按进程、IP 或端口进行缩小范围，**EXTERNAL ONLY** 则会隐藏纯本地流量。

WHAT	Telemetry → START → 观察 CONNECTIONS，或按 PROCESS 或 REMOTE IP 排序以调查某个具体的东西。
WHY	这是对「这台机器上是否有什么东西在访问它不该访问的网络」 这个问题最直接、最具体的回答——直接从内核自身的连接表中读取， 而不是来自某个供应商对自家软件的说法。
RESULT	每一条出站连接的实时、精确画面，只要监视处于运行且未 暂停状态，就会持续更新。
RISK	没有——完全只读。

那些没有寻常理由自行访问网络的进程所发起的连接——一个裸露的 shell、一个独立运行的脚本解释器——会在状态栏中自动标记为 「suspicious」，这样异常情况就会被凸显出来，而不是被淹没在浏览器 和系统服务的正常流量之中。

▸ ADVANCED

Past this point: config file paths, raw commands, and what the panel is doing underneath.

10.2 Advanced

数据来自哪里

面板读取的是与任何标准工具会读取的相同的内核级连接表，在监视处于 激活状态时以短间隔进行刷新——没有任何专属的特殊驱动或内核模块。 命令行等效操作：

```
# 每一个打开的套接字，附带其所属进程——与 CONNECTIONS 相同的字段
$ sudo ss -tupn
# 按接口划分的吞吐量——与 TRAFFIC 相同的数字
$ ip -s link
# 一次性的实时视图，如果你想在面板之外查看它
$ sudo nethogs
```

手动查看 DNS 解析

```
# 实时捕获出站 DNS 查询（需要一个抓包工具）
$ sudo tcpdump -i any -n port 53
```

为什么有些行的 PROCESS 显示为「?」

要把一条实时连接匹配到它所属的进程，需要在捕获的那一刻读取每个 候选进程的 `/proc/PID/fd`——如果连接的打开和关闭速度比面板的刷新 间隔更快，或者所属进程已经退出，面板就没有这个时间窗口来完成这种 匹配，于是显示 `?`，而不是去猜测。这是一种时序上的限制，而不是 表示有什么东西被隐藏起来了。

CHAPTER 11

About

一份关于这台机器的详尽诊断清单，每次打开这个标签页时都会刷新，因此它总能反映出其他面板中发生的变化，而不需要重新打开 Control Center。

11.1 图解指南

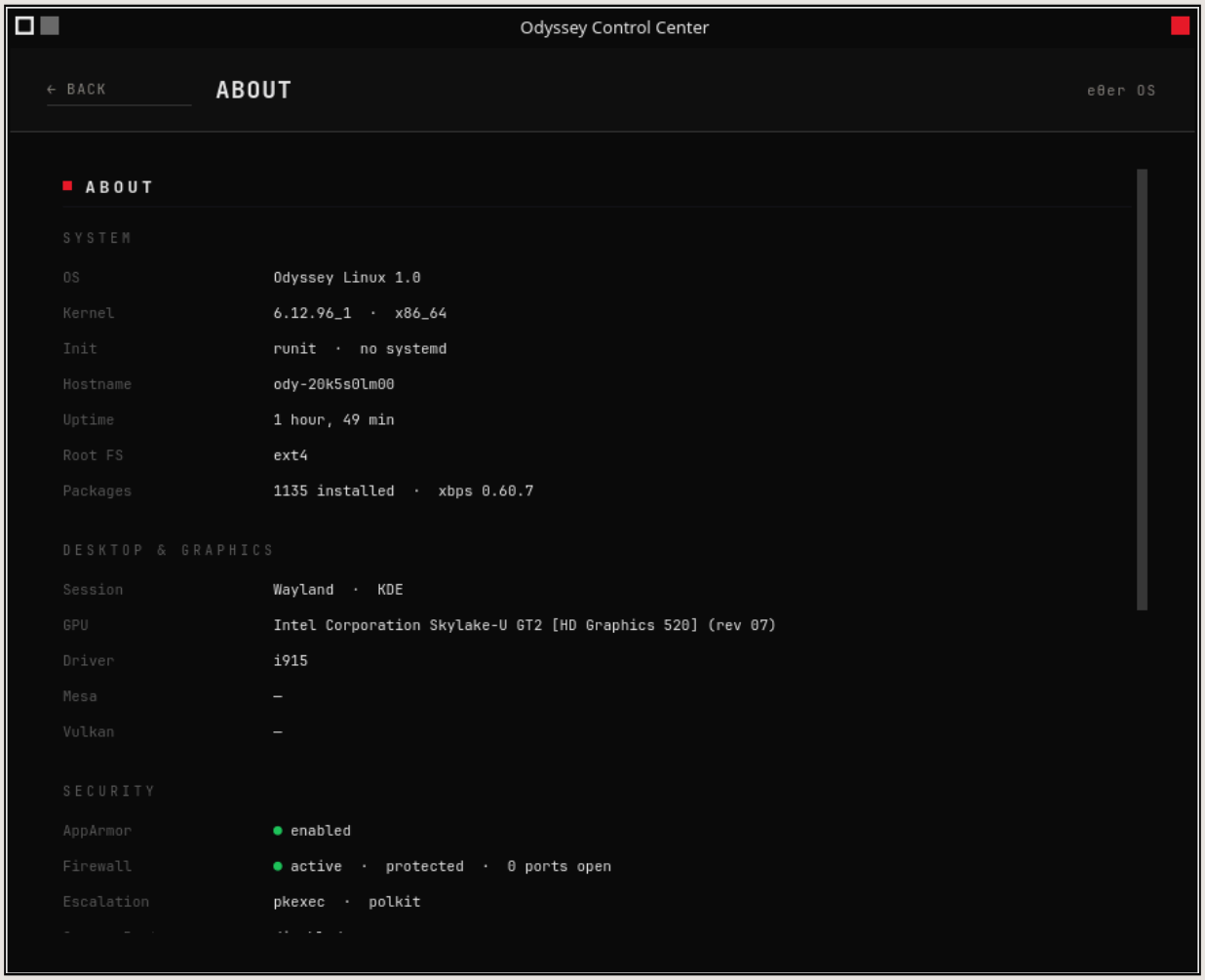

FIG · 面板滚动到最顶部：SYSTEM 部分显示 OS「Odyssey Linux 1.0」、 Kernel 6.12.96_1 · x86_64、Init runit · no systemd、Hostname、 Uptime、Root FS ext4、Packages「1135 installed · xbps 0.60.7」；下方是 DESKTOP & GRAPHICS，包含 Session、GPU、 Driver、Mesa、Vulkan；SECURITY 开始显示 AppArmor「● enabled」 和 Firewall「● active · protected · 0 ports open」。

三个部分，按顺序排列：**SYSTEM**（内核、init、主机名、运行时间、 文件系统、软件包数量）、**DESKTOP & GRAPHICS**（会话类型、GPU、驱动、Mesa/Vulkan 支持情况），再往下是 **SECURITY**（AppArmor 和 Firewall 的状态，用彩色指示器而不仅仅是文字表示——绿色代表已启用/ 激活）以及 **HARDWARE**（主板、音频）。

WHAT	About → 滚动浏览这三个部分，检查任意数值——内核版本、 正在使用的驱动、AppArmor 和防火墙是否真的处于激活状态。
WHY	这是在一个地方确认整个系统状态最快的方式，不需要打开五个 不同的面板。
RESULT	一份完整、最新的快照——每次访问该标签页都会刷新，因此 它从不会显示来自上一次会话的过时信息。
RISK	没有——完全只读。

11.2 Copy report

面板顶部只有一个按钮，**COPY REPORT**，会把整份清单以纯文本形式 放入剪贴板——每一个部分，每一个数值——可以直接粘贴进论坛帖子或 bug 追踪系统的工单中。

WHAT	About → COPY REPORT → 粘贴进你的错误报告或论坛帖子。
WHY	错误报告的用处取决于其所附带的系统细节，而手动输入内核 版本、驱动、AppArmor 状态以及其他十几个字段，正是那种让很多人 干脆跳过这一步的麻烦。
RESULT	一份完整、格式正确的系统报告出现在剪贴板中，可以直接 粘贴使用——维护者原本还需要一个字段一个字段向你询问的那些信息， 现在都齐了。
RISK	对你的系统没有任何风险。这份报告是诊断信息（版本、状态、 硬件型号）——不包含个人文件或凭据——但和任何诊断信息导出一样，如果你不确定它里面到底包含什么，在把它公开发布到某处之前， 先重新读一遍。

▸ ADVANCED

Past this point: config file paths, raw commands, and what the panel is doing underneath.

11.3 Advanced

每个字段来自哪里

这个面板中没有任何内容是凭空编造或从安装那一刻起就被缓存的——每 一个数值都是在标签页打开的每一次都实时查询的：

```
$ uname -r # Kernel
$ cat /proc/sys/kernel/osrelease
$ hostnamectl 2>/dev/null || hostname
$ uptime -p
$ xbps-query -l | wc -l # Packages installed
$ xbps-query -v # xbps version
$ lspci -k | grep -A3 VGA # GPU + Driver
$ glxinfo | grep “OpenGL version” # Mesa
$ vulkaninfo -summary # Vulkan
```

具体到 Security 部分：

```
$ sudo aa-status -enabled && echo enabled
$ sudo nft list ruleset | head -1 # firewall active/mode
```

为什么值得先检查 ABOUT

因为它每次访问都会重新查询系统，About 是确认在其他面板中所做的 更改是否真正生效的最快方式——在 Firewall 中应用某个防火墙模式， 然后去查看 About 的 Security 部分，而不是重新打开 Firewall 本身。 它的设计初衷，就是为 Control Center 的其余部分充当「这真的生效了 吗?」这一交叉检查。

CHAPTER 12

NVIDIA driver

> **等待整合**
>
> 这一章是一个占位章节。NVIDIA 模块只有在检测到 NVIDIA 硬件的机器 上才会出现在 Control Center 中，而它的截图也只能在这样一台机器 上获取——它们将单独提供，这一章会在本指南未来的修订版中完成。它被特意放在最后，位于 About 之后，正是为了让日后添加它时，不会 改变任何其他章节的页码或小节引用。

已知的信息、将会成为完整版这一章基础的内容如下：面板带有一个醒目的 **BETA** 徽章，以及一条前置说明，指出驱动本身是专有软件——由 Void 基于 NVIDIA 官方安装程序打包，带有一个固定的校验和，但无法进行 独立审计，这与 Odyssey 其他部分对自身所要求的诚实标准是一致的。它会将你具体的 GPU 与一个内置数据库进行比对，自动选择正确的驱动 分支（current/图灵架构及更新，或者三个旧分支之一用于更早期的显卡， 在没有任何专有驱动适用时回退到开源驱动 `nouveau`）。安装过程以 一个被追踪的七步流水线运行，带有可见的进度指示，而且面板会在你点击 安装之前就预先打印出它自己的恢复命令——从文本控制台执行 `odyssey-nvidia --revert`。在混合显卡笔记本电脑上，会自动出现一个 PRIME render-offload 部分，允许某个特定的游戏或程序在 NVIDIA GPU 上运行，同时其余一切仍留在集成 GPU 上，以保证电池续航。