ODYSSEY LINUX · リファレンスドキュメント

Odyssey Control Center — 実践ガイド

各パネルをセクションごとに解説。何をするのか、何が変わるのか、
何をクリックすればいいのか──初めて使う人のために書かれています。

Launch Edition

Don’t trust me. Verify me.

— nobody

Odyssey Linux — Odyssey Control Center：実践ガイド
Launch Edition。Launch Edition の ISO に収録されている Control Center を対象とし、Boot から About までの十のパネルを完全に 網羅しています。加えて NVIDIA ドライバーモジュールについては 暫定的な章を設けています（NVIDIA 搭載ハードウェアでのみ表示 され、それを所有するユーザーからのスクリーンショット提供を 待っています）。

Copyright © 2026 the Odyssey Linux project.
Creative Commons Attribution-ShareAlike 4.0 International (CC BY-SA 4.0) ライセンスのもとで公開されています。

本ガイドは Security and Hardening Guide の実践編にあたります。同ガイドは Firewall と AppArmor パネルを技術的に完全な深さで 扱っており、本書ではそれらについて視覚的なステップバイ ステップガイドと相互参照のみを示します。

ここに記載されているすべての画面、ボタン、動作は、記憶では なく、Control Center の実際のソースコードと実際のスクリーン ショットから確認されたものです。

ボタンのラベル、パネル/タブの名前、ファイルパス、コマンドは 英語のまま表示されます。これはプログラムの実際のインター フェースであるためで、Control Center 内で翻訳された表記を 探しても見つかりません。

目次

1 Control Center とは何か 1
1.1 すべてのパネルに共通するパターン 2
2 Boot 3
2.1 ビジュアルガイド──別の OS を見つけてデフォルトを設定する 3
2.2 ビジュアルガイド──デフォルトで起動するものを選ぶ 5
2.3 ビジュアルガイド──EFI タブ 7
2.4 カーネルパラメーター 9
2.5 Advanced 9
3 Kernel 11
3.1 ビジュアルガイド──プリセット 11
3.2 ビジュアルガイド──SYSCTL、グループごとに 12
3.3 ビジュアルガイド──CPU governor 14
3.4 ビジュアルガイド──hugepages 15
3.5 ビジュアルガイド──modules 16
3.6 Advanced 17
4 Memory & Swap 19
4.1 ビジュアルガイド 19
4.2 Advanced 20
5 Firewall 22
5.1 ビジュアルガイド 22
5.2 Advanced 23
6 AppArmor 24
6.1 ビジュアルガイド──STATUS 24
6.2 ビジュアルガイド──PROFILES 25
6.3 ビジュアルガイド──COMMUNITY 26
6.4 ビジュアルガイド──VIOLATIONS 28
6.5 ビジュアルガイド──EDITOR 29
6.6 Advanced 30
7 Privacy 32
7.1 ビジュアルガイド──SearXNG を段階的に 32
7.2 ビジュアルガイド──Tor を段階的に 35
7.3 ビジュアルガイド──両者を組み合わせる 37
7.4 Advanced 38
8 Containers 40
8.1 ビジュアルガイド──エンジン 40
8.2 ビジュアルガイド──rootless vs. rootful、そして空の状態 42
8.3 ビジュアルガイド──厳選されたカタログ、最初から最後まで 42
8.4 ビジュアルガイド──コンテナリストとその操作 46
8.5 ビジュアルガイド──コンテナを実行し、それを開く 47
8.6 ディスク使用量 48
8.7 Advanced 48
9 Gaming Mode 50
9.1 ビジュアルガイド──メインスイッチ 50
9.2 ビジュアルガイド──Packages と Steam 51
9.3 GameMode と MangoHud 53
9.4 Proton 54
9.5 Advanced 54
10 Telemetry 56
10.1 ビジュアルガイド 56
10.2 Advanced 58
11 About 60

11.1 ビジュアルガイド 60
11.2 Copy report 61
11.3 Advanced 61
12 NVIDIA driver 63

CHAPTER 1

Control Center とは何か

Odyssey Control Center は、ひとつのランチャーの裏にある十個の小さな 独立プログラムです——システムの各領域にひとつずつ対応しています：Boot、Kernel、Memory & Swap、Firewall、AppArmor、Privacy、Containers、Gaming Mode、Telemetry、About、そして（NVIDIA ハードウェア上では）NVIDIA ドライバーモジュール。

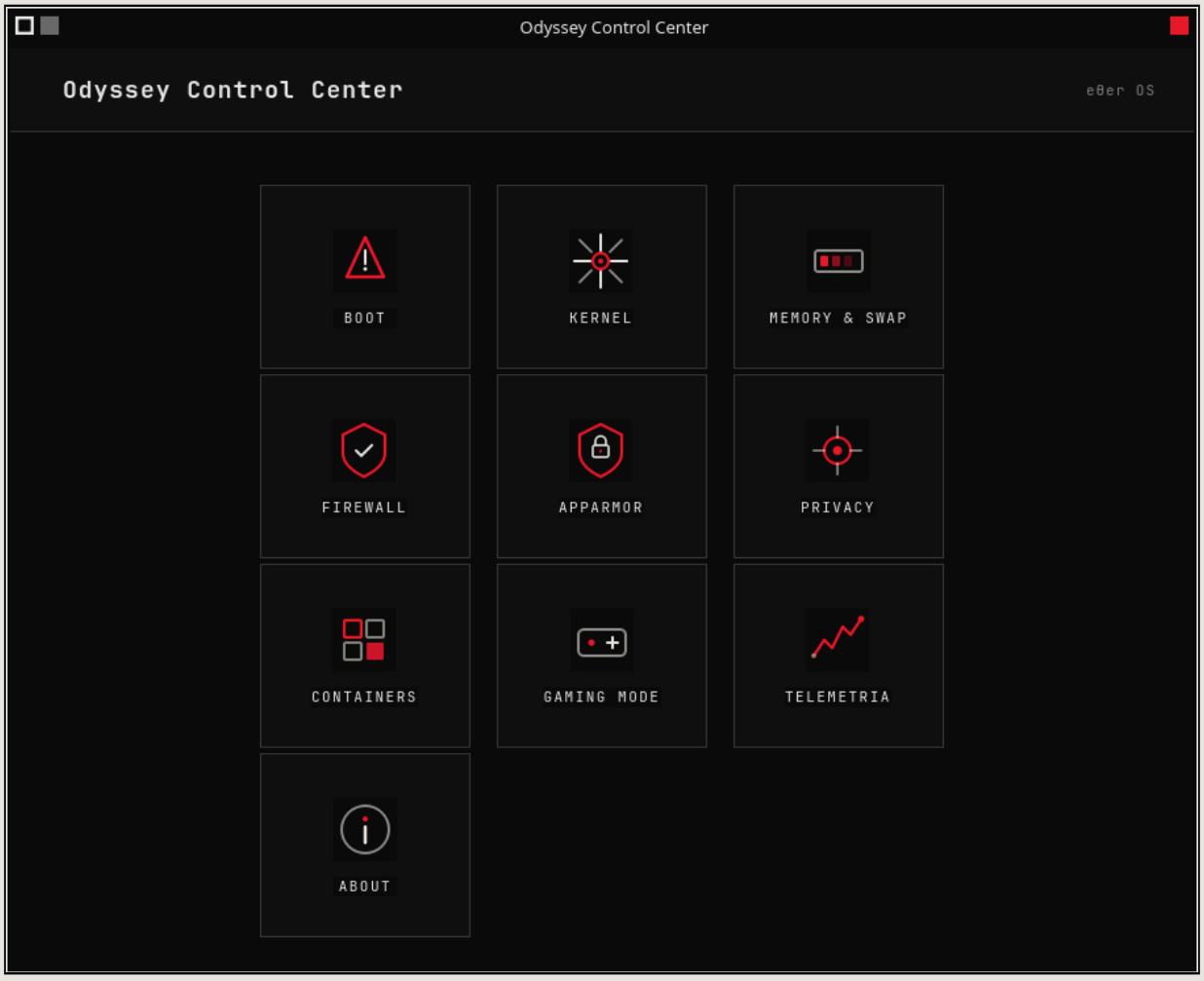


FIG · Control Center のホーム画面—各パネルにひとつずつタイルが 対応しています。

各パネルは、その下にある本物の Linux の仕組み——GRUB の設定、カー ネル自身の sysctl インターフェース、`nftables`、`podman`——の上に かぶさった薄いグラフィカルな層にすぎません。パネルは決して独自の 非公開な状態を作り出すことはありません——実際のファイルやコマンド を読み取り、それをあなたに見せ、あなたが頼んだことをそのまま実行 するだけです。パネルが行うことは何ひとつ隠されておらず、パネルが 行うことはすべて、本ガイドが各ボタンの横に示す正確なコマンドを 使って、ターミナルから手動で行うこともできます。

これは実践的なガイドであり、すべてのオプションを網羅したリファレ ンスではありません——「何をクリックすれば、何が起きるのか」に答える ものです。各章は同じ二部構成になっています：まず**ビジュアルガイド** ——スクリーンショットを追いながら、ボタンをひとつずつ確認していく、 Control Center を一度も

開いたことのない人のために書かれた部分—— 続いて、黒いバナーで示される **ADVANCED** セクションで、詳細に踏み 込みます：正確なファイルパス、各ボタンが裏側で実行する実際のコマ ンド、そしてパネルを完全に飛ばしたい場合にターミナルから何をすれば いいか。ビジュアルガイドだけを読んでもいいですし、すでに Linux に 慣れているなら Advanced に直接進んでも構いません——どちらの部分も それ単体で完結しています。

本当にセキュリティに関わる二つのパネル——Firewall と AppArmor——に ついては、詳しいリファレンスは **Security and Hardening Guide** に あります。ここではビジュアルガイドと要点、そして相互参照を示します。

1.1 すべてのパネルに共通するパターン

いくつかの約束事は十個すべてのパネルで繰り返し登場します——一度 覚えてしまえば、どのパネルにも当てはまります。

パネルごとに一度だけ求められるパスワード

ほとんどのパネルは、root が所有するファイル——GRUB の設定、 sysctl、ファイアウォールのルールセット——を読み書きする必要が あります。あるパネルがそれを初めて必要とするとき、認証ダイアログ が一度だけ表示されます。パネルは読み取り処理をまとめて行うように 作られています：たとえば Boot を開くと、GRUB の設定 **と** EFI ファームウェアの状態を一回のリクエストで同時に読み取るため、 パスワードの入力は一度で済み、二度求められることはありません。 同じセッション内でパネルを再度開いても、パスワードは再要求されま せん。続く操作（変更の適用、再読み込み）は、それぞれのパネル自身の ログに進行状況を表示します。

ステップトラッカー

何かを使えるようにする前にインストールの手順を案内してくれる パネル——Boot の os-prober、Privacy の SearXNG と Tor、Containers のコンテナエンジン——は、上部に小さな番号付きのトラッカーを表示 します：ステップの並びの中で現在のステップが強調表示され、常に **次のステップ**を実行する単一のアクションボタンがあります。何を 最初にクリックすべきか、推測する必要は一切ありません。

下部のログ

すべてのパネルは、スクロール可能な小さなログで締めくくられます。 これは装飾ではありません——実際に実行された正確なコマンド、その 出力、そして色分けされた結果（成功なら緑、失敗なら赤）です。パネル が理解できないことをしているように見えたら、本当の答えはこの ログの中にあります。

TRY VS. MAKE PERSISTENT

特に Kernel と Memory でよく見られる、繰り返し登場するボタンの 組み合わせがあります：**TRY NOW**（または **APPLY NOW**）は稼働中の システムを即座に変更し、それ以上何もしなければ次回の再起動時に 元に戻ります。**MAKE PERSISTENT** は同じ値を `/etc/` 以下の設定 ファイルに書き込み、再起動後も残るようにします。この区別は意図的 なものです——値を試すことには何のコストもかからず、二度言うまで は何も恒久的にはなりません。

CHAPTER 2

Boot

UEFI モードで起動するすべてのマシンにおいて、Boot パネルには二つの タブがあります：**GRUB**（起動メニューそのもの）と **EFI**（その下に あるファームウェア層）です。BIOS のみの古いマシンでは、下層に EFI 層自体が存在しないため、GRUB だけが存在します。

> **ひとつではなく、二つの層**
>
> **GRUB** は画面に表示される起動メニューです——GRUB 自体がすでに 動作している状態で、どのオペレーティングシステムを起動するかを 決定します。**EFI/UEFI ファームウェア**は GRUB よりひとつ下の層に あります——GRUB がまだロードされる前の段階で、マシンがそもそも 最初にどのブートローダーに制御を渡すかを決定します。ほとんどの 人は EFI タブに一切触れることはありません。これは、独立して 自分自身の起動エントリを管理する二つ目の Windows インストール のような、GRUB では対応できないケースのために存在しています。

2.1 ビジュアルガイド—別の OS を見つけてデフォルトを設定する

GRUB タブは **Other OS detection** セクションで開きます。Odyssey は デフォルトでは他のシステムをスキャンしません——ほとんどのインス トールではディスクに Odyssey しか存在せず、起動メニューを再生成 するたびにスキャンを行うのは無駄に時間がかかるだけだからです。これを有効にするのは、小さな番号付きトラッカーとして示される、 案内付きの三段階の手順です。

1. **`Installed`**

   スキャンツールである os-prober はまだ システムにインストールされていません。ボタンはひとつだけ：**INSTALL OS-PROBER**。

2. **`Enable scan`**

   ツールはインストール済みですが、GRUB は まだそれを無視するよう設定されています——これが Odyssey の デフォルトです。ボタンはひとつだけ：**ENABLE SCAN**。

3. **`Scan`**

   これで GRUB はそれを使用するようになります。 ボタンはひとつだけで、今後別のシステムを追加または削除するたびに 再び使うことになるボタンです：**SCAN & REGENERATE**。

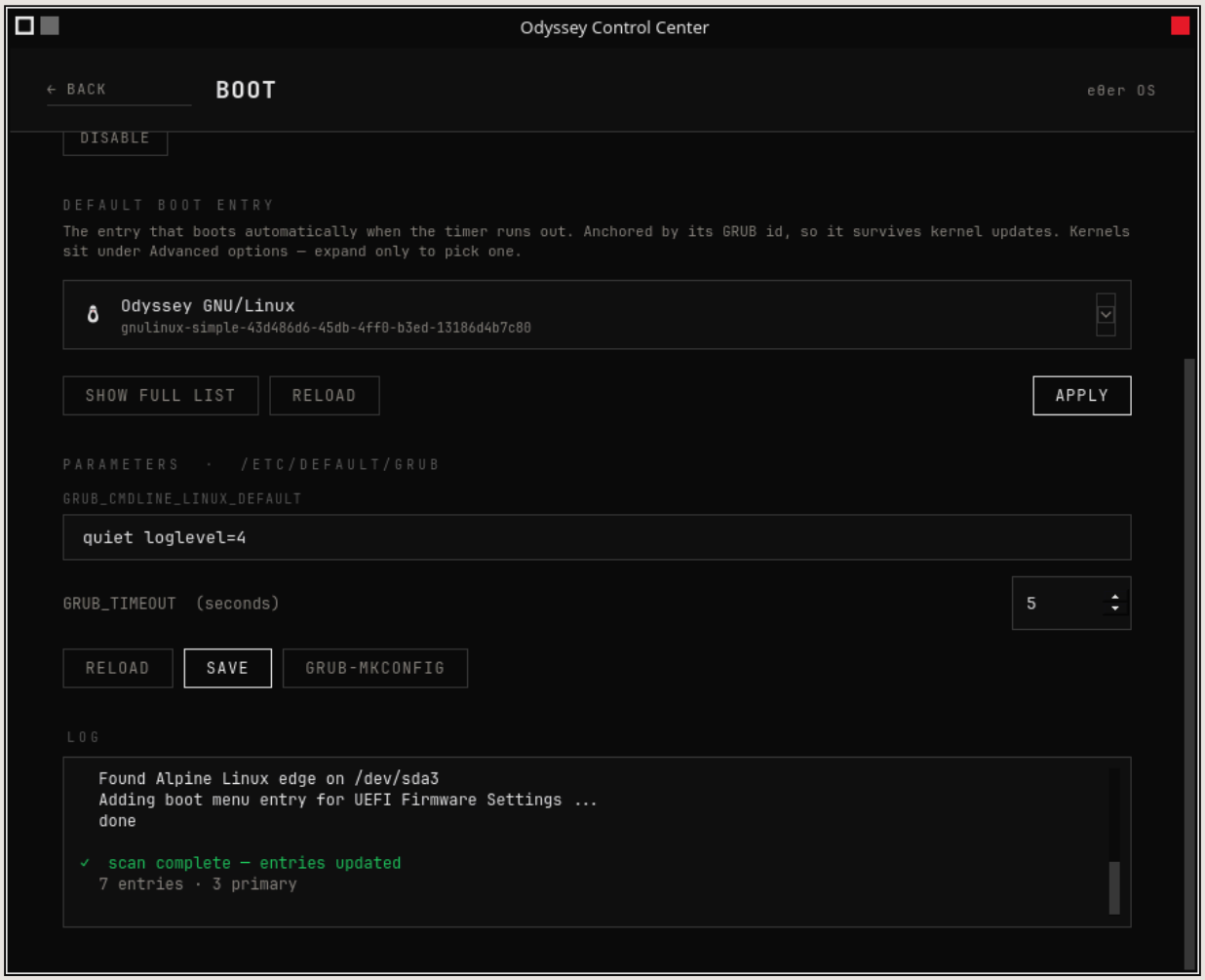


FIG · 三つのステップすべてが緑色に──os-prober はインストール済み、 スキャンは有効、準備完了。単一のアクションボタンは常に現在のステップに対応しています。

各ステップをクリックしていきます (それぞれ、必要な場合にのみ 再度認証を求められます)。そしてスキャンを実行します。パネルは、 このセクションを離れる前に、何が見つかったかを正確に示してくれ ます：

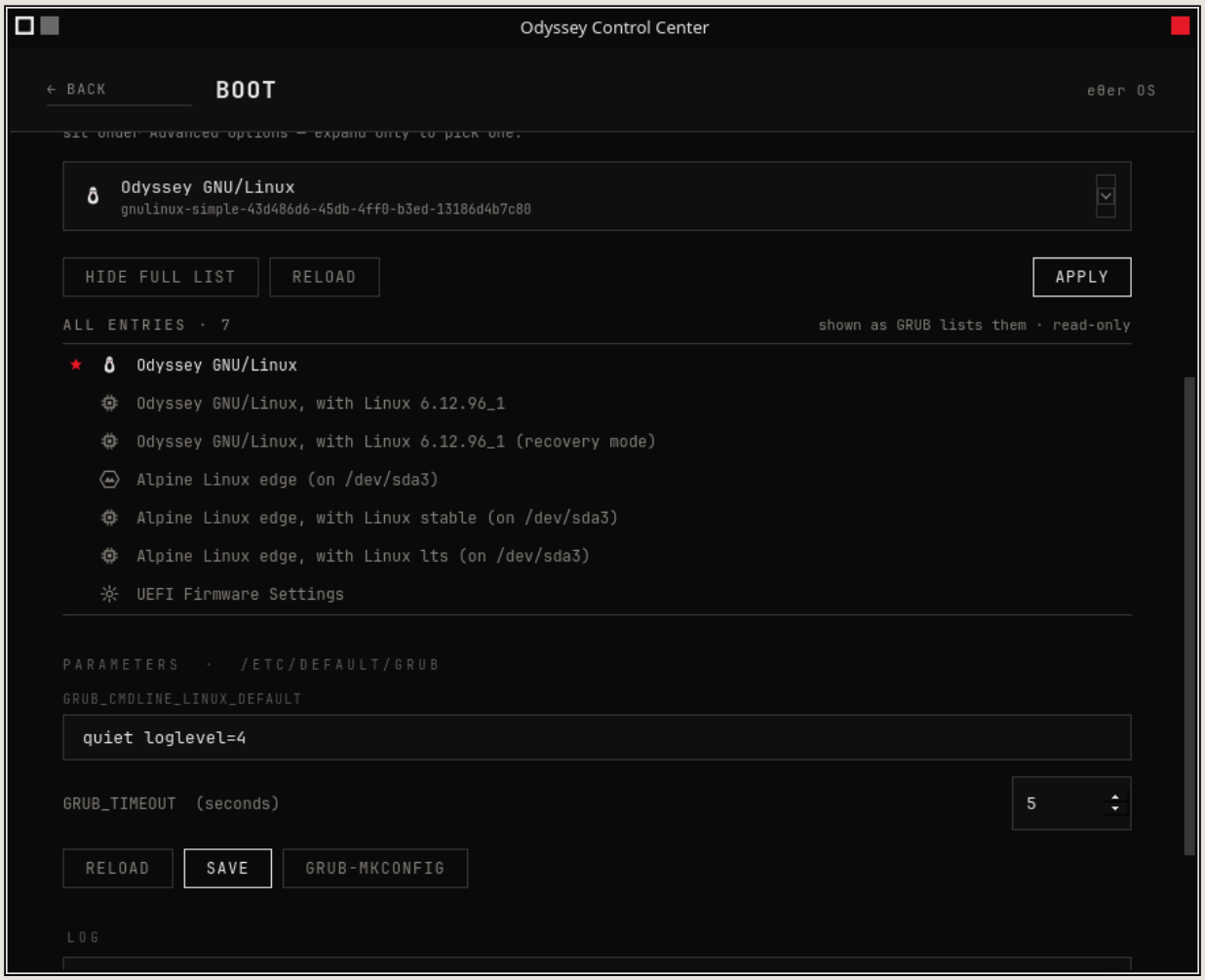


FIG · スキャン完了：/dev/sda3 上に Alpine Linux が見つかり、Odyssey 自身のエントリおよび UEFI Firmware Settings のエントリと並んで 表示されています—合計 7 エントリ、うち主要なものが 3 つ。

ログはこれをシンプルな言葉で確認します——**「scan complete — entries updated · 7 entries · 3 primary」**。os-prober が今後見つ けるものは何であれ、下のセクションに自動的に表示され、選択できるようになります。

2.2 ビジュアルガイド—デフォルトで起動するものを選ぶ

Default boot entry までスクロールします。現在のデフォルトが 一行で表示されており、それをクリックすると、代わりに選択できる すべての項目を含むドロップダウンが開きます。

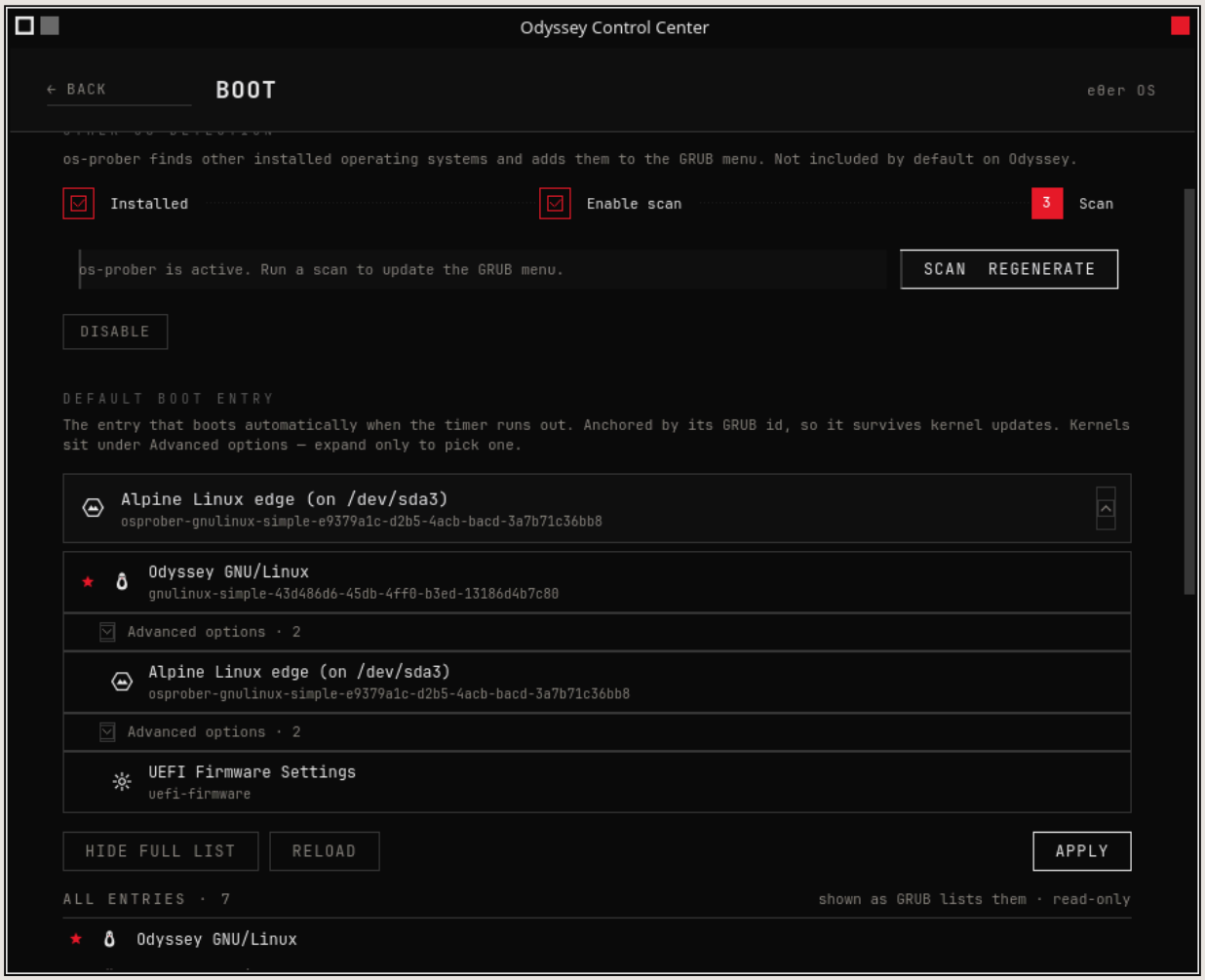


FIG · 開かれたドロップダウンで、Alpine Linux が保留中の選択として 選ばれています—まだ適用されていません。

リストからエントリを選ぶことは、その選択を**保留状態にする**だけ です——まだ効力はありません。それがまだ保留中であることは、ボックスが新しい選択を表示し続けている一方で、まだ何も書き込まれ ていないことからわかります。それを確定するボタンは、右側にある **APPLY** です：

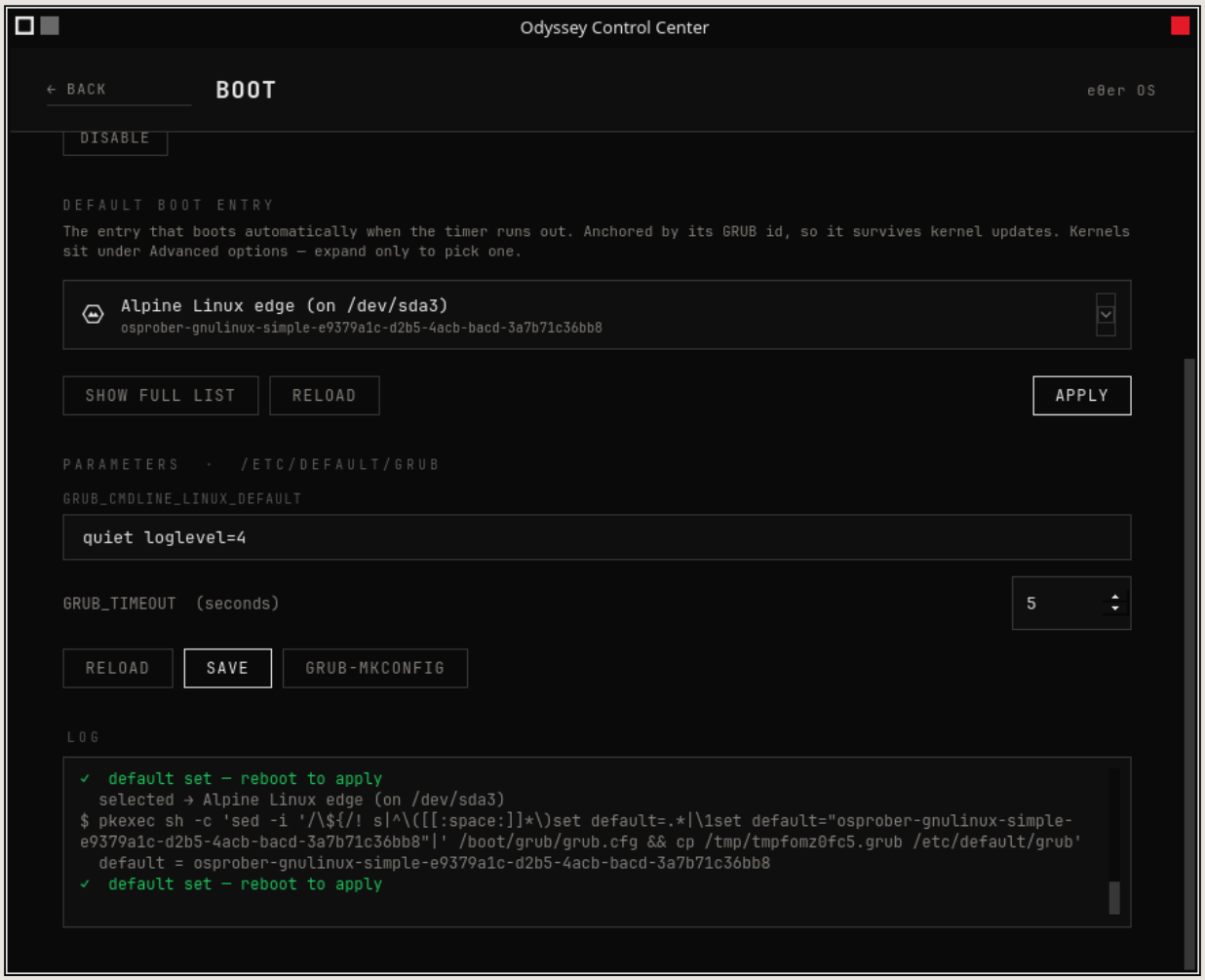


FIG · APPLY の後：ログが「default set — reboot to apply」と確認し、 セレクターは Alpine Linux を現在有効なデフォルトとして表示して います。

WHAT	Boot → GRUB タブ → 現在のデフォルトエントリをクリック → ドロップダウンから別のものを選択 → **APPLY** をクリック。
WHY	デュアルブート環境で、自動的に起動するシステムを変更したい 場合、あるいはカーネルの更新によって古いカーネルが選択された ままになっており、最新のものに戻したい場合。
RESULT	選択したエントリは、次回の再起動以降、自動的に起動する ようになります。この変更はそのエントリの安定した id に固定 されるため、将来のカーネル更新後も変わりません。
RISK	データに対するリスクはありません——最悪の場合でも、間違った エントリを選んでしまい、戻ってもう一度変更すればよいだけです。

SHOW FULL LIST は、GRUB 自身がリストするとおりに、すべての エントリを正確に表示します——読み取り専用で、個々のカーネルごとの バリアントは、メインリストを埋め尽くすのではなく、展開可能な「Advanced options」の行の下にグループ化されています。設定がパネル の外で変更されたと思われる場合は、**RELOAD** が GRUB の設定を 再読み込みします。

2.3 ビジュアルガイド—EFI タブ

上部の **EFI** タブをクリックします。これはファームウェアの層です ——オペレーティングシステムがまだ何も起動していない段階で、 あなたのマザーボード自身の起動メニューが表示する内容です。

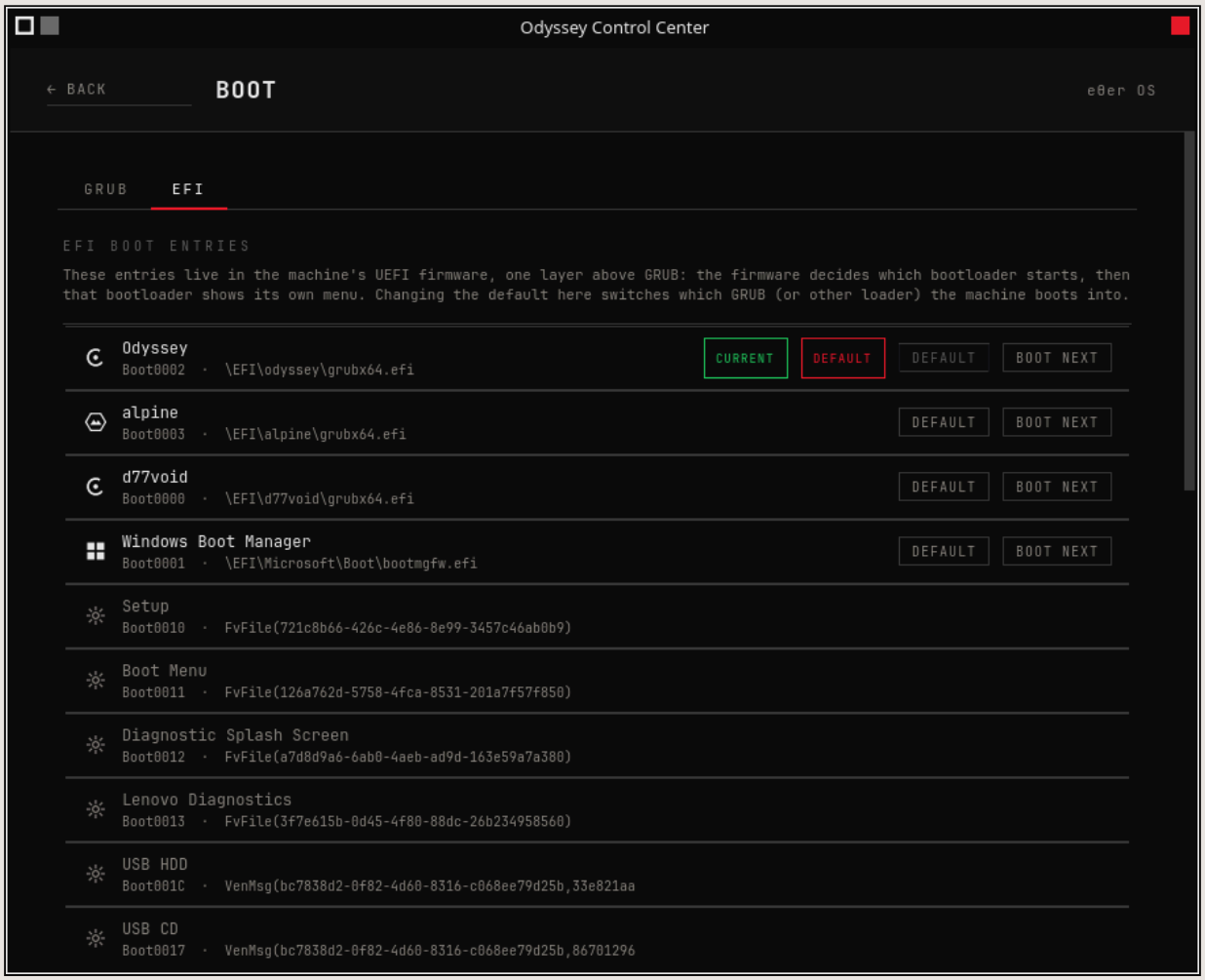


FIG · EFI タブ：Odyssey が緑と赤で CURRENT と DEFAULT としてマーク され、alpine、d77void、Windows Boot Manager がそれぞれ独自の DEFAULT / BOOT NEXT ボタンを持ち、下にはファームウェアの ユーティリティエントリが並んでいます。

各行はひとつのファームウェアエントリです——その名前、GRUB の id （`Boot0002` など）、そしてそれが指す正確な `.efi` ファイル。行 ごとに二つのボタンがあります：

ボタン	動作
DEFAULT	このエントリをファームウェアの永続的なデフォルトにします——これ以降、すべての再起動でこのエントリが起動します。
BOOT NEXT	一回限りの例外です——このエントリは**次回**の再起動でのみ起動し、その後は通常の順序が自動的に再開されます。例外が保留中の間は、上部に CANCEL ボタン付きのバナーが表示されます。

WHAT	Boot → EFI タブ → 目的のエントリを見つける（例：「Odyssey」や「Windows Boot Manager」）→ DEFAULT をクリック、 または一回限りの例外として BOOT NEXT をクリック。
WHY	マザーボード自身の起動メニューが、まったく間違ったブート ローダーを起動している場合——これは GRUB の設定の問題ではなく、 ファームウェアの設定の問題です——ここが唯一それを修正できる 場所です。
RESULT	DEFAULT：そのエントリはファームウェアの起動順序の先頭 に永続的に移動します。BOOT NEXT：一度だけ起動し、その後は 元に戻ります。
RISK	低いリスクで、何かが予期せず起動した場合でも、この同じ タブから、あるいはマザーボード自身の UEFI セットアップ画面 （通常は電源投入時に Del または F2）から、常に復旧できます。

2.4 カーネルパラメーター

GRUB タブの最後のセクションは、プレーンテキストのカーネルコマンド ライン——GRUB の構文を必要とせず、通常のテキストフィールドのように 編集できます——と、秒単位のメニュータイムアウトです。**SAVE** はこれ らを `/etc/default/grub` に書き込みます。**GRUB-MKCONFIG** はこの ファイルから実際の起動メニューを再生成します。この手順によって、 保存したパラメーターが次回の起動時に有効になります。パネルは 再生成のたびに、現在のデフォルトの選択を常に再適用するため、この 操作によってデフォルトで起動するシステムがリセットされることは ありません。

```
▸ ADVANCED

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

2.5 Advanced

実際に起きていること

Odyssey の GRUB 設定は**フラット**です——`submenu` ブロックはあり ません。各カーネルはトップレベルの `menuentry` であり、これは 安定した id（数値の位置ではなく）が、カーネル更新をまたいでも常に それを正しく選択できることを意味します。セレクターに表示される「Advanced options」のグループ化は、純粋に視覚的な便宜です—— あるオペレーティングシステムの単純なエントリと、そのカーネルごとの バリアントの間で共有される UUID を突き合わせることで構築されて います。GRUB 自身のファイルにはこのようなグループ化はありません。

デフォルトを設定すると、同じ操作の中で二箇所に書き込みが行われ ます：`/etc/default/grub` の永続的な `GRUB_DEFAULT`、そして—— `grub.cfg` がすぐには再生成されない可能性があるため—— `/boot/grub/grub.cfg` 自体の中にある、現在有効な `set default="..."` の行に対する、`sed` による直接的かつ的確な編集 です。これは `grub-reboot` が使用するワンショットの分岐 `${next_entry}` を経由しません。だからこそ、デフォルトの設定は、 最初に完全な `grub-mkconfig` を強制することなく、次回の起動時に 有効になるのです。

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

OS-PROBER を手動で

```
$ sudo xbps-install -y os-prober
# スキャンを有効化—/etc/default/grub の GRUB_DISABLE_OS_PROBER を編集
$ 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
```

EFI を手動で—EFIBOOTMGR

EFI タブは `efibootmgr` へのインターフェースです。パネルが行う ことはすべて、ひとつのコマンドに対応しています：

```
# 現在の起動順序ですべてのファームウェアエントリを一覧表示
$ sudo efibootmgr -v
# 並べ替え：エントリ 0002 を先頭に（= DEFAULT）
$ sudo efibootmgr -o 0002,0000,0001,0003
# エントリ 0003 への一回限りの起動（= BOOT NEXT）
$ sudo efibootmgr -n 0003
# 保留中の BootNext をキャンセル
$ sudo efibootmgr -N
```

孤立したローダー

インストーラーが、EFI System Partition に自身のブートローダーを 書き込みながら、それに対応するファームウェアエントリを登録し 忘れることがあります——ローダーはディスク上に存在しますが、ファームウェアはそこにあることを認識していません。Odyssey の パネルは ESP を直接スキャンし、それらを**「ON DISK, NOT IN FIRMWARE」**として別枠でリストアップし、それぞれにワンクリックの REGISTER ボタンが付いています。手動での操作は `efibootmgr -c -d /dev/sdX -p N -L "名前" -l '\EFI\名前\bootx64.efi'` に相当し ます——新しく作成されたエントリは、ファームウェア自身によって 起動順序の**先頭**に配置されることに注意してください。したがって、それが主導権を奪ってほしくない場合は、その後で普段使っている エントリに DEFAULT を再度適用してください。

このパネルが触れるファイル

ファイル	役割
`/etc/default/grub`	カーネルコマンドライン、タイムアウト、GRUB_DEFAULT、GRUB_DISABLE_OS_PROBER
`/boot/grub/grub.cfg`	実際に生成された起動メニューそのもの——手動で編集しないでください。grub-mkconfig によって上書きされます
EFI 変数 (NVRAM)	ファームウェアの起動エントリ——これはファイルではまったくなく、完全に efibootmgr を通じて読み書きされます

CHAPTER 3

Kernel

四つのタブ——**SYSCTL**、**CPU GOVERNOR**、**HUGEPAGES**、**MODULES**——に 加えて、その四つすべての上にワンクリックのプリセットの列があり、 すべてに同時に適用されます。

3.1 ビジュアルガイド—プリセット

上部には：現在稼働しているカーネルのバージョン、アクティブな スケジューラー、そして四つのプリセットボタン——**DESKTOP**、 **GAMING**、**SERVER**、**BALANCED**——が表示されます。

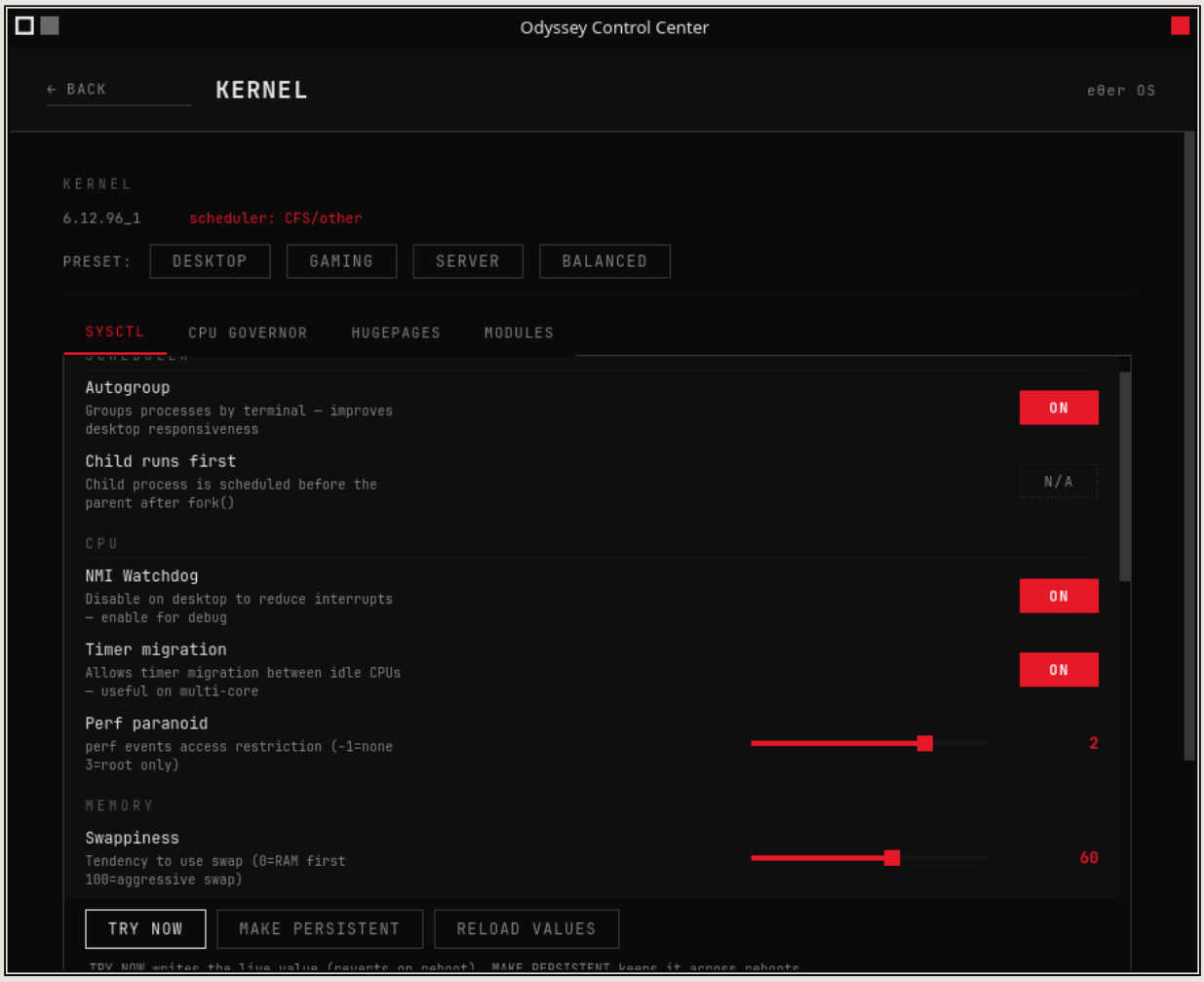


FIG · 開いたばかりの SYSCTL タブ：カーネル 6.12.96_1、スケジューラー CFS/other、四つのプリセット、そして下に Scheduler/CPU/Memory のグループ。

プリセット	使う価値があるとき
DESKTOP	日常使いのマシン——ウェブ閲覧、オフィス作業、軽いマルチタスク。より低い swappiness、適度なキャッシュ圧力、アイドル時の穏やかな消費電力のためのタイマーマイグレーションの有効化。ほとんどの人にとって理にかなったデフォルトです。
GAMING	swappiness をほぼゼロまで下げる（すべてを RAM に留める）、タイマーマイグレーションを無効化（コアが作業を移動させず専用状態を維持する）、MangoHud のようなプロファイリングツールのために perf イベントの制限を緩和。フレームタイムがアイドル時の消費電力より重要な場合に使用します。
SERVER	より高い swappiness とキャッシュ圧力への耐性、NMI watchdog の有効化、カーネルポインタ/ログに対するより厳格な制限。対話的な応答性よりも安定性とセキュリティを優先する、ディスプレイのないマシンで使用します。
BALANCED	Desktop と Gaming の中間——ワークステーションになったりゲームセッションになったりするマシンで、プリセットを頻繁に切り替えたくない場合に。

WHAT	Kernel → DESKTOP、GAMING、SERVER、BALANCED のいずれかを クリック。
WHY	単に「ゲームに適した良い設定」が欲しいだけの人にとって、 十数個の sysctl の値を手動で調整するのは要求が多すぎます—— プリセットはまさにそれ、あらかじめ決められた答えです。
RESULT	SYSCTL タブのすべての値がプリセットの数値へと一斉に 切り替わり、準備が整い可視化されます——実際に適用するには それでも TRY NOW か MAKE PERSISTENT（下記）が必要です。
RISK	なし——プリセットを選ぶことは、フォームに値を入力する だけです。

3.2 ビジュアルガイド—SYSCTL、グループごとに

下にスクロールすると、各パラメーターは見出しの下にグループ化 されており、それぞれにパネル内で直接わかりやすい言葉での説明が 付いています——`vm.compaction_proactiveness` が何を意味するのかを、 まずそこで読まずに知る必要は一切ありません。

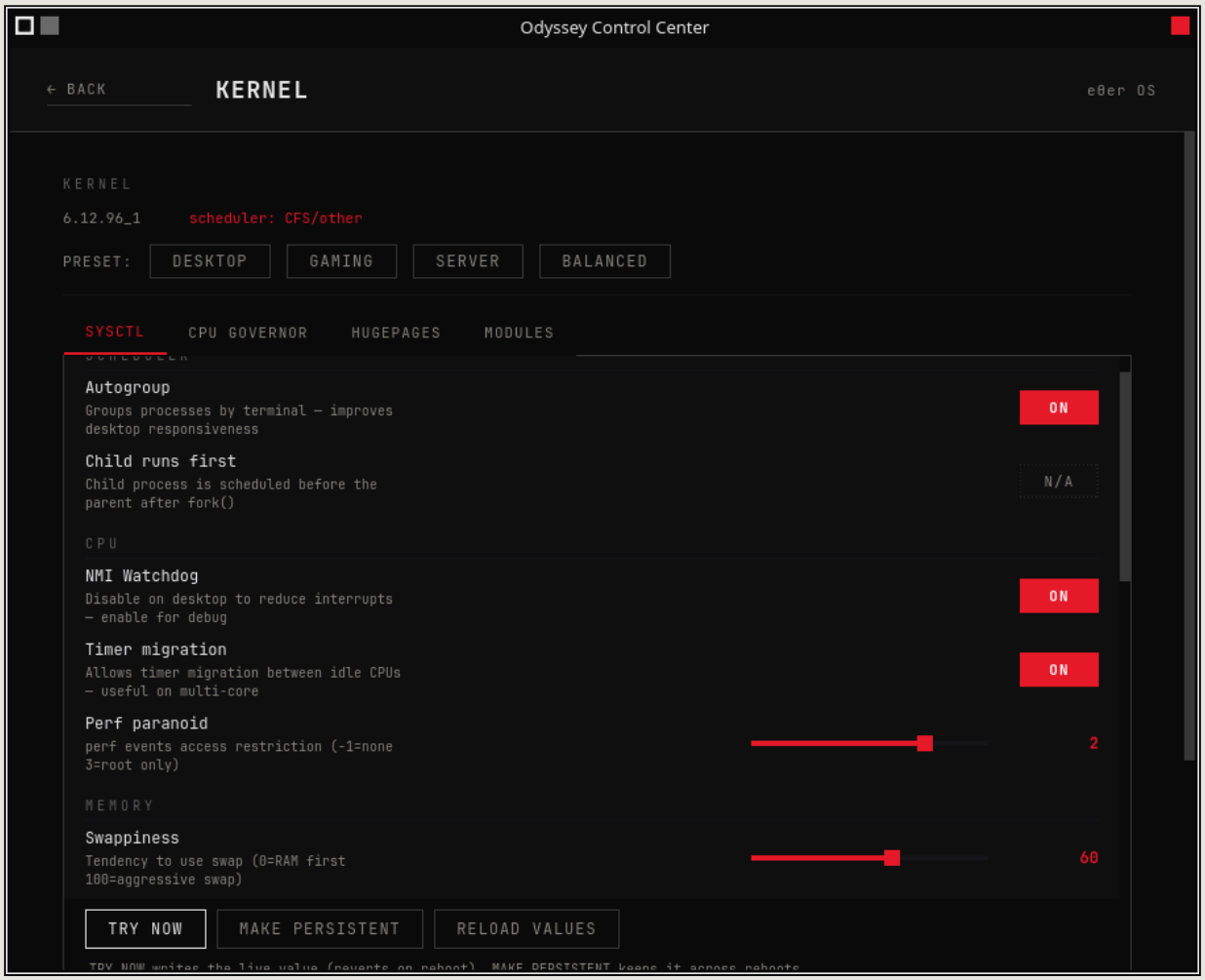


FIG · Scheduler と CPU のグループ：Autogroup、Child runs first、NMI Watchdog、Timer migration、Perf paranoid—それぞれに独自の スイッチやスライダーと、説明の一行が付いています。

グループ	含まれる内容
Scheduler	プロセスがどのようにグループ化され、優先順位付けされるか
CPU	Watchdog、タイマーマイグレーション、パフォーマンス監視に関する制限
Memory	Swappiness、キャッシュ圧力、「ダーティ」ページの書き戻し閾値
Network	TCP Fast Open、backlog キューのサイズ、ソケットバッファの上限
Security	カーネルポインタとログメッセージが、権限のないユーザーから隠されるかどうか

下部の二つのボタンが編集内容を適用します——これらは意図的に異なる ものです：

WHAT	TRY NOW
WHY	失敗した再起動を生き延びるリスクをゼロにして、値を試して みる。
RESULT	`sysctl` を通じてライブに書き込まれます——即座に有効に なり、このセッション限りです。
RISK	なし——再起動時に自動的に元に戻ります。

WHAT	MAKE PERSISTENT
WHY	値（またはプリセット全体）を試してみて、それが再起動後も 残ってほしい場合。
RESULT	`/etc/sysctl.d/99-odyssey.conf` に書き込まれ、**同時に** 同じクリックでライブにも適用されます。
RISK	低い——これはただのテキストファイルです。行を削除するか 再適用すれば元に戻せます。

RELOAD VALUES は現在実際に有効な内容を再読み込みし、入力済み だが適用されていないものはすべて破棄します。

3.3 ビジュアルガイド—CPU governor

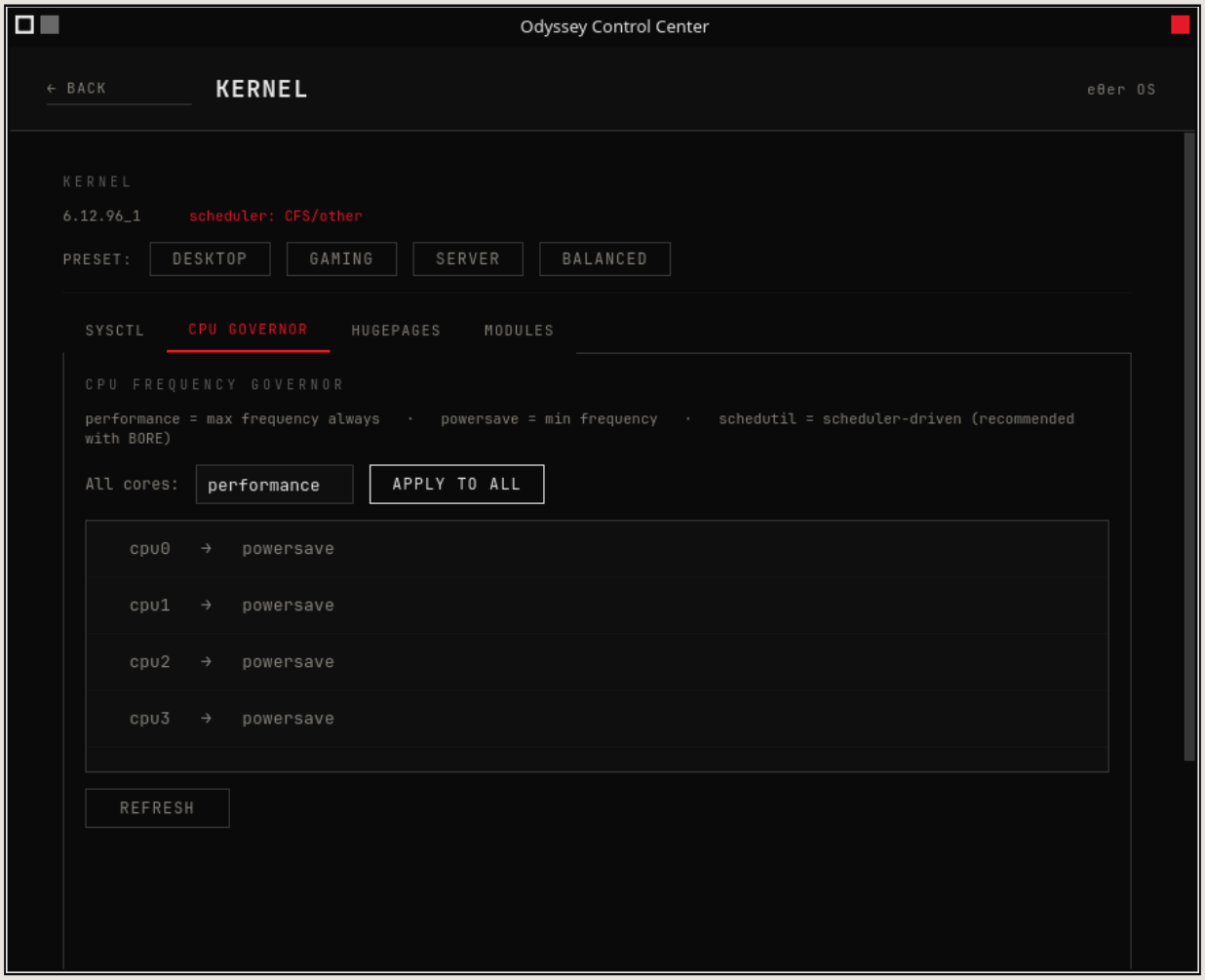


FIG · CPU GOVERNOR タブ：「All cores」が performance に設定され APPLY TO ALL が表示されている一方、下のコアごとのリストはまだすべて powersave を示しています—修正を待っている不 一致です。

Governor は、CPU がクロック周波数をどれだけ積極的に変更するかを 決定します。ドロップダウンはすべてのコアに一括で governor を 適用します。下のリストはコアひとつひとつの **ライブな** governor の状態を示します——これによって、上のスクリーンショットのように 「All cores」が performance と表示されているのに、下の各コアが まだ powersave のままになっている、つまり APPLY TO ALL がまだ クリックされていない、といった間違った設定に固定されたコアを発見 できます。

GOVERNOR	動作
performance	常に最大周波数——最も応答性が高く、消費電力も最大
powersave	常に最小周波数——最も静かで、応答性は最も低い
schedutil	スケジューラー自身の負荷判断に基づいて周波数が制御される——BORE スケジューラーとの組み合わせで推奨されます

ドロップダウンには、あなたの具体的な CPU とカーネルが実際に公開 している governor のみが表示されます。

3.4 ビジュアルガイド—hugepages

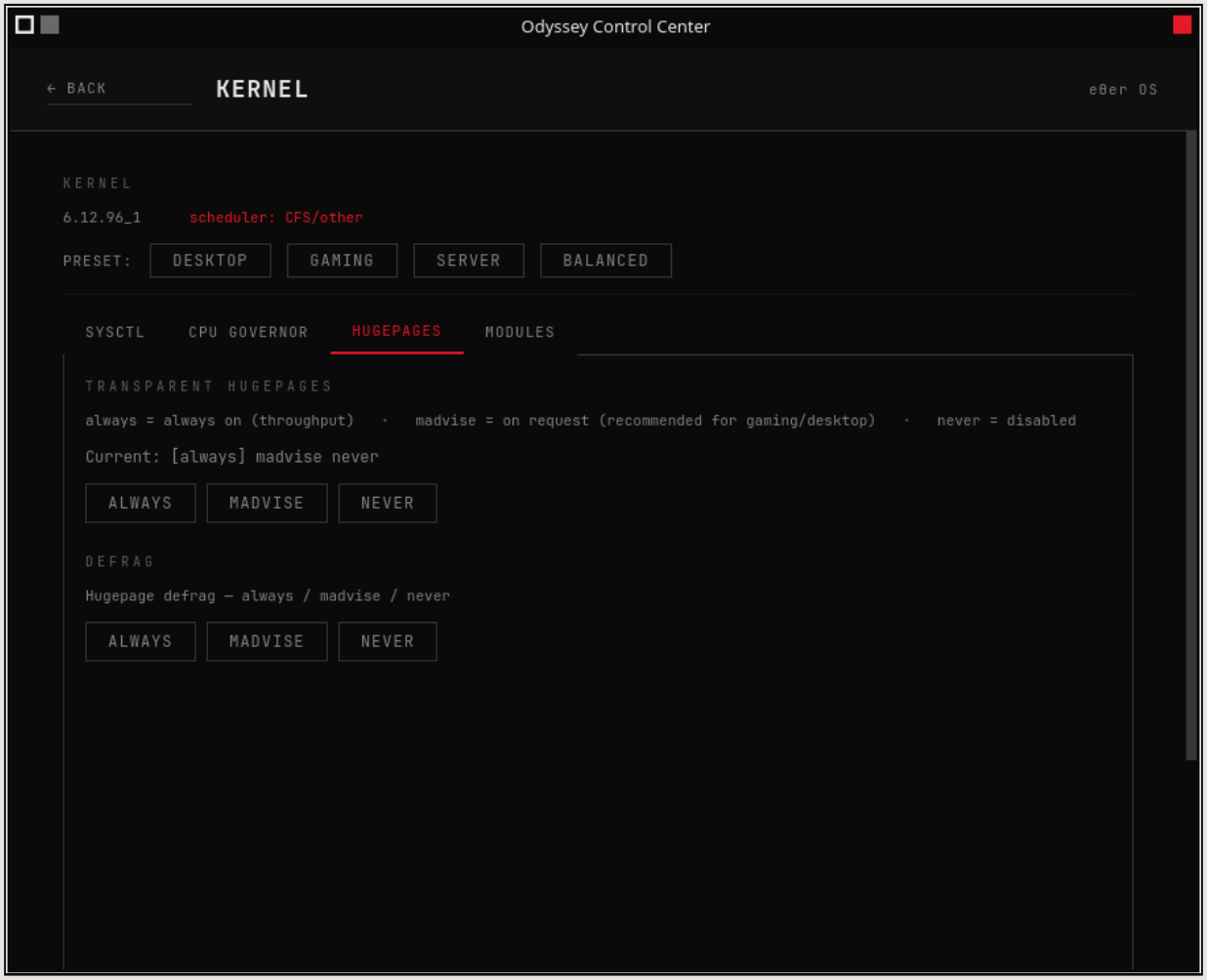


FIG · HUGEPAGES タブ：Transparent Hugepages が「Current: [always] madvise never」を表示し ALWAYS が強調されており、下の Defrag にも同じ三つのボタンの並びが再度表示されています。

二つの独立した設定があり、それぞれクリックした瞬間に適用される 三つのボタンの並びです——現在のライブな値は常にボタンの上に角 括弧で表示されます。スクリーンショットの `[always] madvise never` のように。

設定	制御内容
Transparent hugepages	ALWAYS = 常にすべての場所で有効（最大のスループット）；MADVISE = アプリが明示的に要求したときのみ（ゲーム/デスクトップ用途で推奨）；NEVER = 完全に無効

設定	制御内容
Defrag	要求に応じて hugepages を**形成**するために、カーネルがどれだけ積極的にメモリを圧縮するか——上記の設定とは独立した、同じ三つの選択肢

3.5 ビジュアルガイド—modules

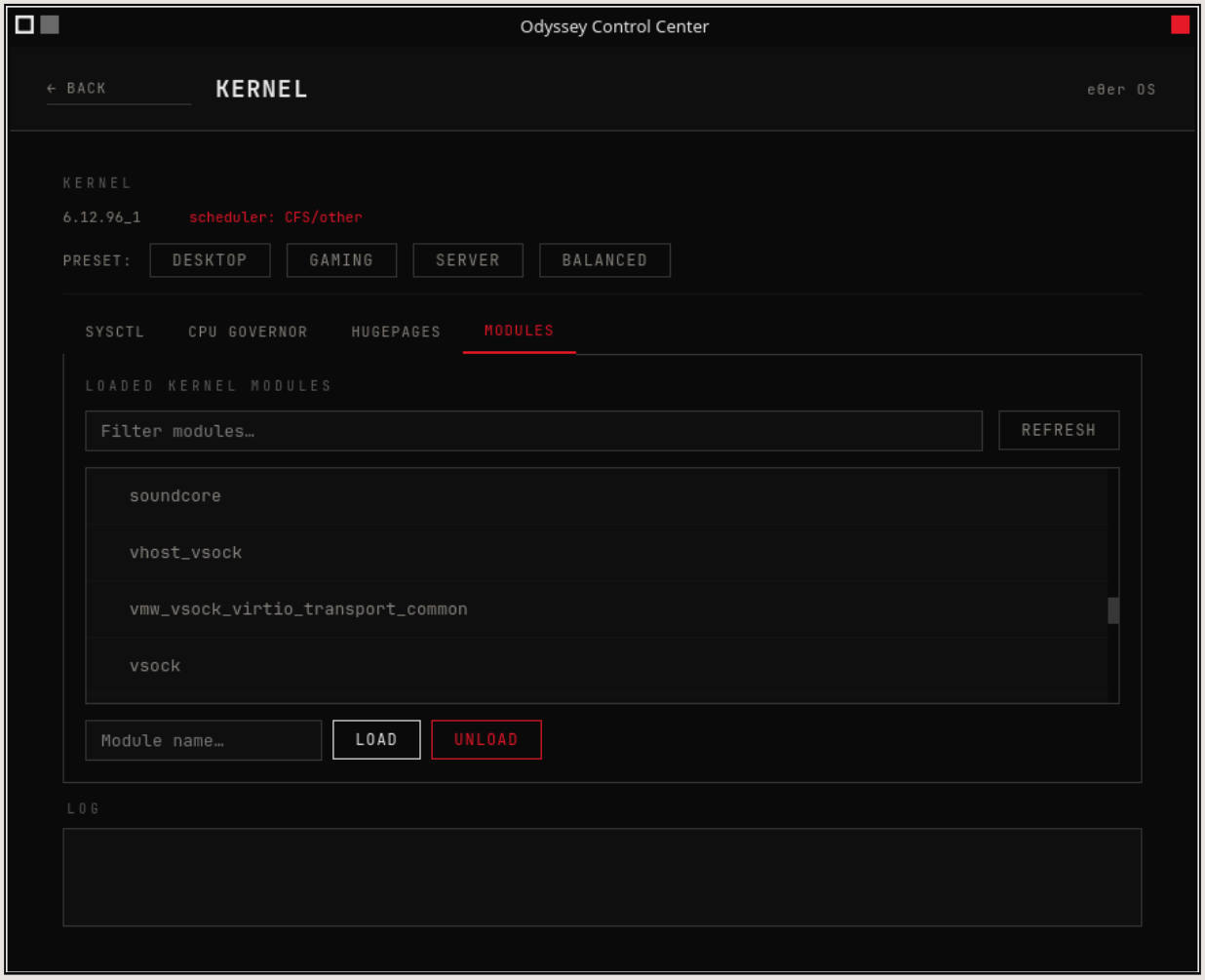


FIG · MODULES タブ：フィルターフィールド、ライブなリスト （soundcore、vhost_vsock、vsock……）、そして下部に、名前で 入力するモジュール用の LOAD/UNLOAD の並び。

現在ロードされているすべてのカーネルモジュールの、ライブでフィル タリング可能なリストです——フィルターに入力して絞り込み、 REFRESH で再読み込みします。他の三つのタブとは異なり、これは 直接操作することができます：モジュール名を入力し、**LOAD** または **UNLOAD** をクリックします。

WHAT	Kernel → MODULES タブ → モジュール名を入力 → LOAD または UNLOAD。
WHY	再起動せずに、特定のドライバーや機能モジュールを有効（または削除）にしたい場合があります——ハードウェアのテスト、 別のモジュールが保持しているデバイスの解放など。
RESULT	そのモジュールは即座に稼働中のカーネルに挿入されるか、 そこから削除されます。
RISK	UNLOAD については特に中程度——他の何かが依存している モジュールを削除すると、それを使用していたもの（表示ドライ バー、ファイルシステムなど）を不安定にする可能性があります。LOAD は概して安全です。カーネルは互換性のないモジュールを 半端に適用するのではなく、その場でロードを拒否します。

▸ ADVANCED

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

3.6 Advanced

正確な SYSCTL キー

キー	パネル内のラベル
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

プリセットの値、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
# プリセットの完全な表は panel_kernel.py で直接確認できます——
# すべてのフィールドを確認したい場合は Control Center のソースコードを開いてください
```

パネルを使わず手動で

```
# ライブ、一時的
$ sudo sysctl vm.swappiness=10
# 永続的—パネルが書き込むのと同じファイル
$ echo “vm.swappiness=10” | sudo tee -a /etc/sysctl.d/99-odyssey.conf
$ sudo sysctl -system
# CPU governor、コアごと
$ 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
# モジュール
$ sudo modprobe MODULE_NAME
$ sudo modprobe -r MODULE_NAME
```

なぜ TRY NOW と MAKE PERSISTENT は別々の操作なのか

`sysctl` への書き込みは、その性質上、常にライブなものにすぎ ません——カーネルには組み込みの永続化機能がありません。MAKE PERSISTENT は、Odyssey がひとつの正規のファイル `/etc/sysctl.d/99-odyssey.conf` を選び、ライブへの書き込みと ファイルへの書き込みの両方を同時に行うことで、二つが決してずれ ないようにしたものです。このファイルを直接編集し、その後 `sysctl --system` を実行することは、機能的にはボタンをクリック するのとまったく同じです。

CHAPTER 4

Memory & Swap

一つの画面、三つのセクション：読み取り専用のサマリー、 swappiness、そして ZRAM です。

4.1 ビジュアルガイド

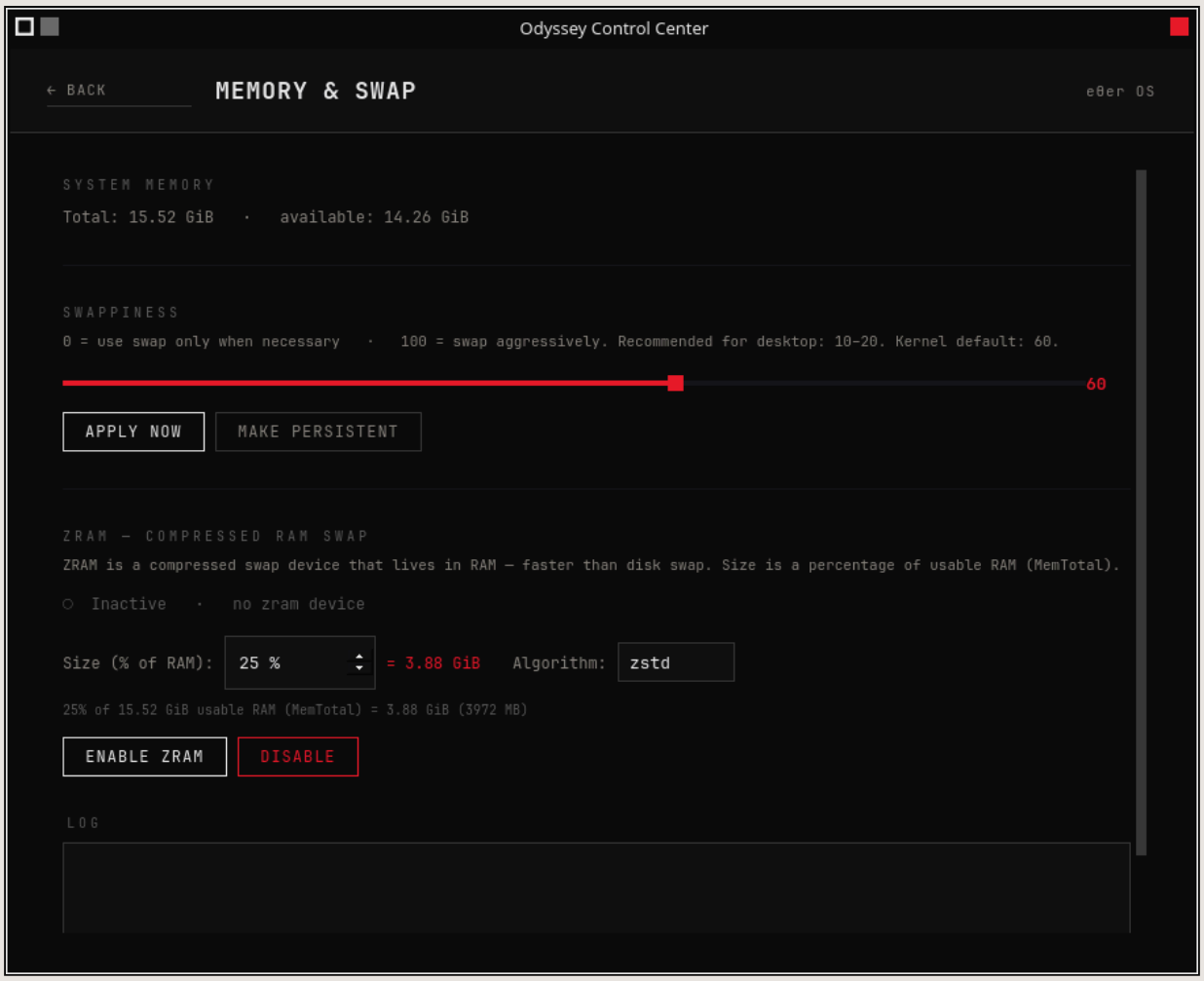


FIG · 完全なパネル：System memory（合計 15.52 GiB、利用可能 14.26 GiB）、APPLY NOW / MAKE PERSISTENT 付きの 60 に設定された Swappiness スライダー、そして下に「Inactive · no zram device」 と表示された、サイズ・アルゴリズム・ENABLE/DISABLE 付きの ZRAM。

SYSTEM MEMORY

上部には単なるサマリーがあります——合計 RAM と現在利用可能な量。 操作項目はありません。これは下の二つのセクションのための背景 情報です。

SWAPPINESS

0 から 100 までの単一のスライダーです。0 は「本当に他に選択肢が ない場合にのみ swap を使う」ことを意味し、100 は「RAM が満杯に なるずっと前から積極的に swap を使う」ことを意味します。カーネル のデフォルトは 60 です（スクリーンショットに表示されている値）。パネル自身のヒントはデスクトップ用に 10〜20 を推奨しています—— これは Kernel パネルの DESKTOP プリセットが設定するのと同じ数値 です。

WHAT	スライダーをドラッグ → APPLY NOW で試す、満足したら MAKE PERSISTENT。
WHY	より低い値は、あなたが積極的に使っているものを、より 遅い swap に移動させるのではなく、高速な RAM に留めます—— 十分なメモリが搭載されている場合、通常これがあなたの望む 動作です。
RESULT	APPLY NOW はそれを即座にライブで変更し、このセッション 限り有効です。MAKE PERSISTENT はそれを設定ファイルに書き込み、 再起動後も残るようにします。
RISK	非常に低い。本当に RAM が不足しているマシンで極端に低い 値を設定すると、穏やかに swap するのではなく、高負荷時に 突然メモリを使い果たしやすくなる可能性があります——そうなった 場合は、値を再び上げてください。

ZRAM—RAM 内の圧縮スワップ

ZRAM は、RAM 自体の**内部**に存在する、圧縮された swap デバイス です——ディスクがまったく関与しないため、ディスク上の swap より 著しく高速です。サイズは固定の数値ではなく、利用可能な合計 RAM に対する割合であるため、この設定は異なるマシンに対して合理的に スケールします。サイズフィールドの横にあるライブな表示が、あなた のためにその割合を GiB に換算してくれます（スクリーンショット では：25% = 15.52 GiB のシステムで 3.88 GiB）。

WHAT	サイズの割合を設定し、アルゴリズム（zstd、lz4、lzo、deflate のいずれか）を選択 → ENABLE ZRAM。
WHY	RAM 内の圧縮 swap は、ディスク上の swap よりもはるかに 小さなパフォーマンスへの影響でメモリの圧力を吸収します—— どのマシンでも役立ちますが、特に RAM が限られている場合に有用です。
RESULT	選択したサイズとアルゴリズムで `zram0` デバイスが作成 され、有効化されます。ステータス行は「Inactive」からアクティブ な状態へと変わります。
RISK	低い。zstd（デフォルト）はほとんどのマシンにおいて圧縮率 と CPU コストのバランスが良好です。DISABLE はいつでもきれいに それを無効化します。

▸ ADVANCED

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

4.2 Advanced

ここでの「ハードウェアに依存する」の実際の意味

どちらの設定も、単なる「RAM が多いほど良い」ではなく、あなたの 具体的なハードウェアと相互作用します：

- **高速な NVMe SSD**：10〜20 の範囲の swappiness は、たとえ swap が 発生してもほとんどコストがかかりません——ディスクはほとんど 気付かないほど十分に高速です。ZRAM はそれでもレイテンシと SSD の摩耗の点で優位ですが、機械式ディスクの場合よりもその差は 小さくなります。
- **swap として使う機械式 HDD や eMMC**：swappiness を低く（0〜10） 保ってください——遅い媒体でのディスク swap は「thrashing」と 呼ばれるカクつきを引き起こします。まさにここで ZRAM ははるかに 大きな効果を発揮します。
- **CPU の余裕**：ZRAM は、ディスク I/O を、圧縮と解凍に費やされる CPU サイクルと引き換えにします。CPU に制約のあるマシン（古い ハードウェア、あるいはゲーム中にすでに限界に近いマシン）では、`lz4` は `zstd` より圧縮率は劣りますがより高速に圧縮します—— CPU 時間がより逼迫したリソースであれば、これは合理的な トレードオフです。現代的なハードウェアであれば、`zstd` のより 良い圧縮率は、そのわずかに高いコストに見合う価値があります。
- **RAM の余裕**：合計わずか 8 GiB のマシンで大きな ZRAM の割合 （50% など）を使うと、ZRAM 自体が圧縮されたページを保存する ために実メモリを消費するため、アプリケーションに残される実 RAM は少なくなります。16 GiB 以上であれば、25% は余裕を持った 快適なデフォルトです。

手動で

```
# swappiness、ライブ
$ sudo sysctl vm.swappiness=15
# swappiness、永続的—パネルが書き込むのと同じファイル
$ echo “vm.swappiness=15” | sudo tee /etc/sysctl.d/99-odyssey.conf
# zram、手動設定
$ 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
# 有効になっていることを確認
$ swapon -show
```

なぜパネルはライブでリサイズせず、ZRAM0 をリセットするのか

ZRAM デバイスのサイズは、それがアクティブで swap として使用中の 間は変更できません——カーネルは、まず無効化してリセットすること を要求します。パネルの ENABLE ZRAM ボタンは、この一連の手順（swapoff、reset、再設定、swapon）を自動的に実行します。だから こそ、すでにアクティブなデバイスでサイズやアルゴリズムを変更 しても、それはエラーではなく、依然としてひとつのきれいな操作 として表示されるのです。

CHAPTER 5

Firewall

Firewall パネルには、**Security and Hardening Guide** の中に、 その全深度を扱う専用の章があります——ルールセットを一行ずつ解説 し、21 個すべてのプリセットにそれぞれのリスクを付し、生の設定を 編集するエディター、そしてコマンドラインでの等価な操作方法まで。 この章は、そのビジュアルガイド版にあたります。

5.1 ビジュアルガイド

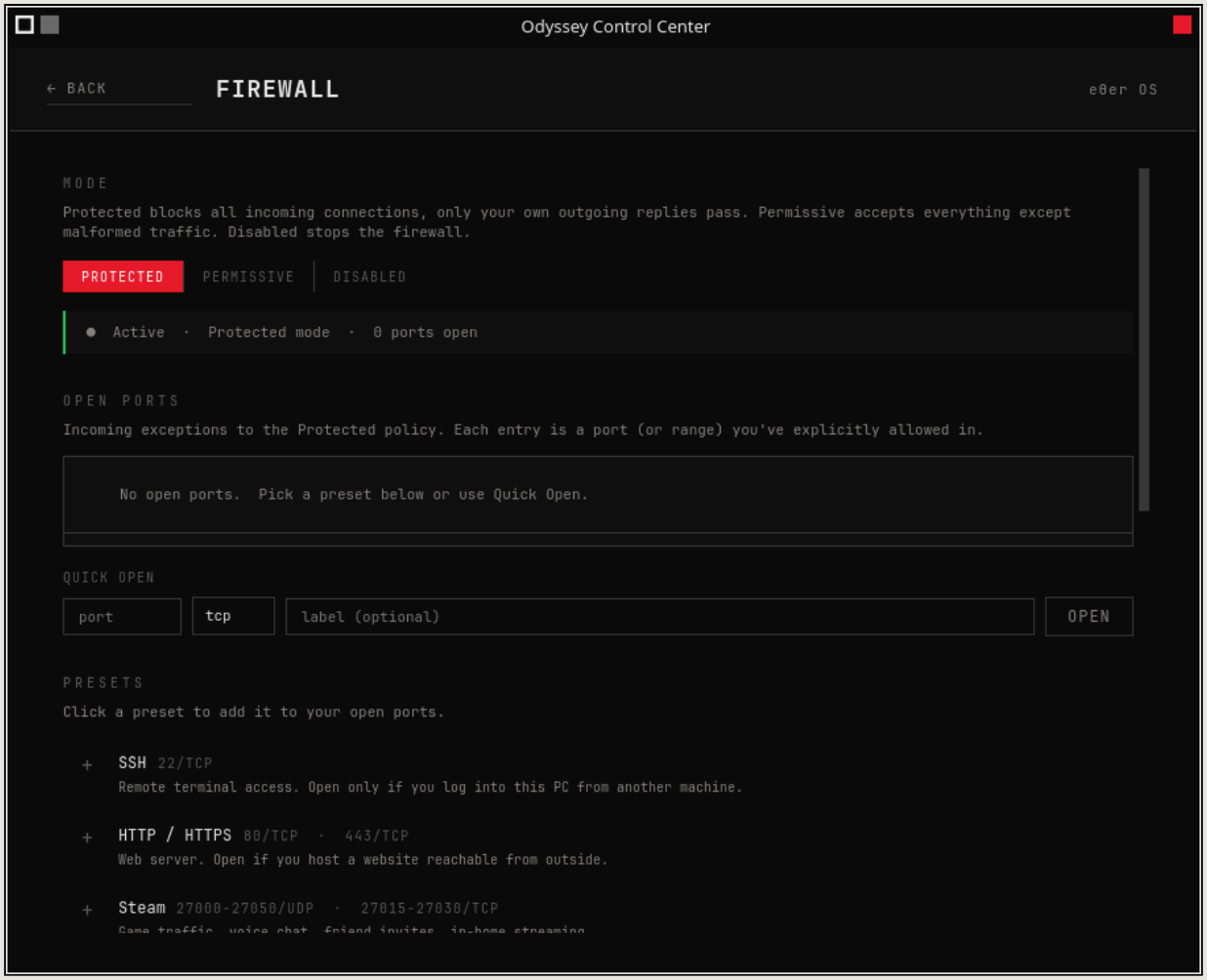


FIG · PROTECTED モードが有効、「0 ports open」、Quick Open フィール ド、そして SSH、HTTP/HTTPS、Steam から始まるプリセットの タイル。

MODE

上部に三つのボタンがあり、パネルはそのすぐ上で、それぞれを一行で 説明しています：

モード	動作
PROTECTED	デフォルト。すべての着信接続をブロックします。通過するのは自分自身の発信への応答だけです。
PERMISSIVE	不正な形式のトラフィックを除くすべてを許可します——「ファイアウォールがこれを壊しているのか？」を診断するための設定であり、常用する場所ではありません。
DISABLED	ファイアウォールを完全に停止します。

下のステータス行は、実際に何が有効になっているかを確認します—— スクリーンショットでは **「Active · Protected mode · 0 ports open」** です。

OPEN PORTS とプリセット

OPEN PORTS は、明示的な例外を一覧表示します——表示されている とおり、デフォルトでは空です。**QUICK OPEN** は、プリセットリスト にない任意のものについて、ポート、プロトコル、任意のラベルを直接受け付けます。下の **PRESETS** は、八つのよく使われるサービスを それぞれワンクリックで提供し、それぞれに、本当にそれが必要になる のはどんなときかを説明する一行が付いています（SSH：「リモート ターミナルアクセス。別のマシンからこの PC にログインする場合の み開いてください」）。さらに展開可能な「MORE PRESETS」の下に、 あと 13 個あります。

WHAT	Firewall → プリセットのタイルをクリック、あるいは QUICK OPEN に入力 → SAVE & APPLY。
WHY	別のマシンから到達する必要がある何か——ゲームサーバー、 メディアサーバー、開発用サーバー——を動かしている場合。
RESULT	そのポートが開きます。それ以外はすべて PROTECTED ポリシーによって閉じたままです。
RISK	完全にプリセット次第です——インターネットから到達可能な マシンで SSH や HTTP/HTTPS 以外の何かを開く前に、Security and Hardening Guide のプリセットごとのリスクに関するメモを 参照してください。

SAVE & APPLY を押すまで、何も反映されません——変更はまず保留 状態になるため、確定する前に、これから何を開こうとしているのかを 正確に確認できます。

```
▸ ADVANCED

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

5.2 Advanced

完全な技術リファレンス——ルールごとに解説された `nftables.conf` のデフォルトルールセット、それぞれの具体的な露出範囲と緩和策を 含む 21 個すべてのプリセット、LAN のみにアクセスを制限するための レシピ、`nft` コマンドラインでの等価な操作方法、そして締め出さ れた SSH セッションのトラブルシューティング——はすべて、**Security and Hardening Guide の第 2 章**（インストール済みシス テム上では `/usr/share/doc/odyssey/`）にあります。本マニュアルは それを意図的に重複させていません。あの章を、このセクションの 続きとして考えてください。

```
# 今すぐ知っておく価値のあるコマンド：
$ sudo nft list ruleset
# パネルとは無関係に、カーネルに実際にロードされている内容を正確に表示します
```

CHAPTER 6

AppArmor

五つのタブ：**STATUS**、**PROFILES**、**COMMUNITY**、**VIOLATIONS**、 **EDITOR**。Firewall と同じ相互参照です：強制アクセス制御が実際に 何であるか、閉じ込め（confinement）の完全なライフサイクル、実践 的な例といった詳しいリファレンスは、Security and Hardening Guide にあります。ここではビジュアルガイドを示します。

6.1 ビジュアルガイド—STATUS

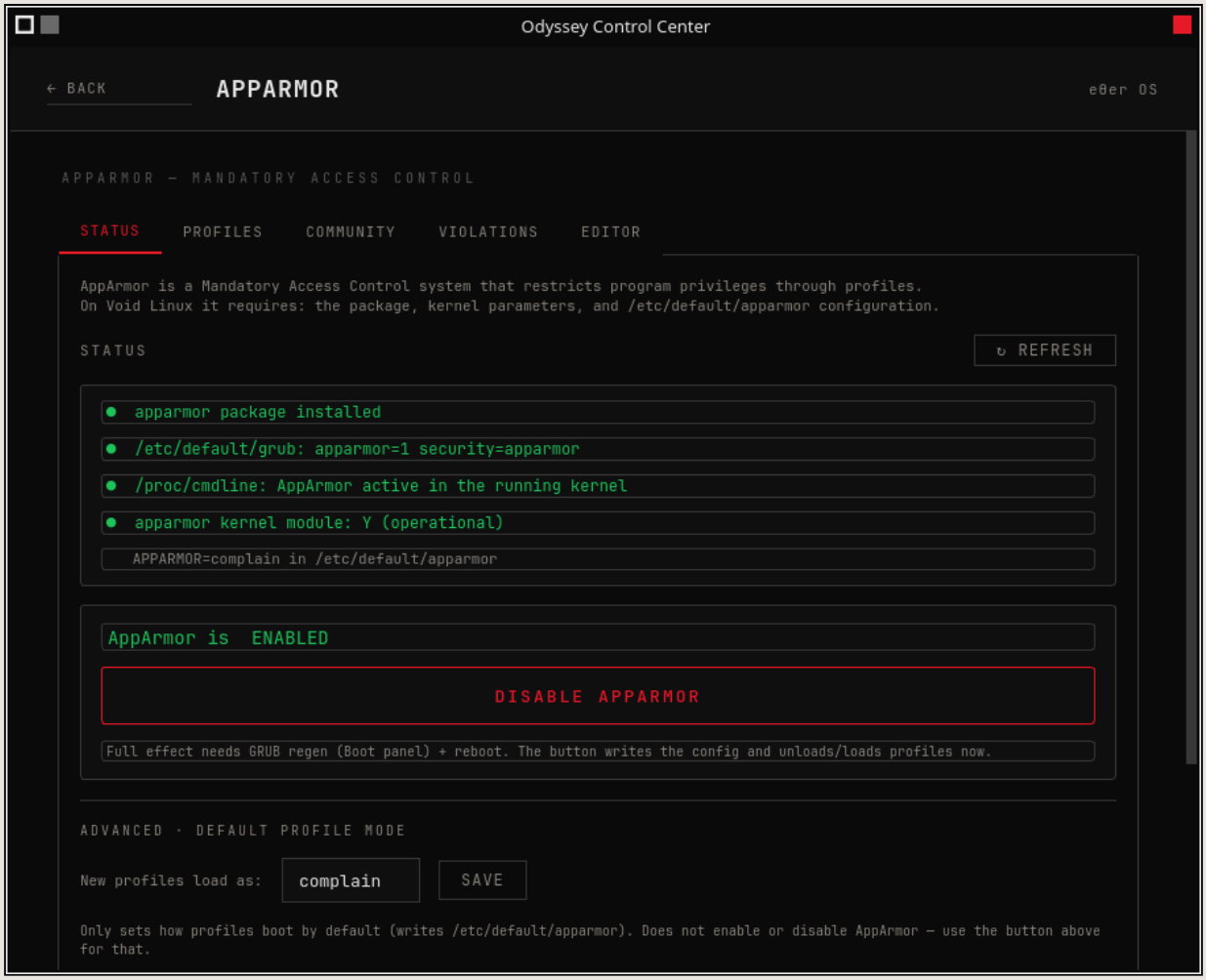


FIG · STATUS タブ：四つの緑色のチェック（パッケージのインストール、 GRUB パラメーター、稼働中のカーネル、カーネルモジュール）、 赤い DISABLE APPARMOR ボタン付きの「AppArmor is ENABLED」、 そして complain に設定されたデフォルトのプロファイルモード。

四つのチェックが、上から下に、このフレームワークが実際に機能して いることを確認します：apparmor パッケージがインストールされて いること、GRUB のカーネルパラメーターが設定されていること、 **稼働中の**カーネルが実際にそれらを反映して起動していること（これは、それらを有効にした後に再起動していない場合に遅れが 生じうる項目です）、そしてカーネルモジュール自体が正常に動作して いること。その下には、

大きな有効化/無効化ボタンと、**ADVANCED · DEFAULT PROFILE MODE**——新しく読み込まれたプロファイルが complain と enforce のどちらで起動するか——があります。

WHAT	AppArmor → STATUS → 四つのチェックすべてが緑色である ことを確認。
WHY	いずれかのチェックが赤や黄色になっている場合、このパネル の他の部分は正しく機能できません——AppArmor に関して何か おかしいと感じたときに、常に最初に確認すべき箇所です。
RESULT	このフレームワークが単にインストールされているだけで なく、本当にアクティブであるという確信が得られます。
RISK	なし——読み取り専用の検証です。

6.2 ビジュアルガイド—PROFILES

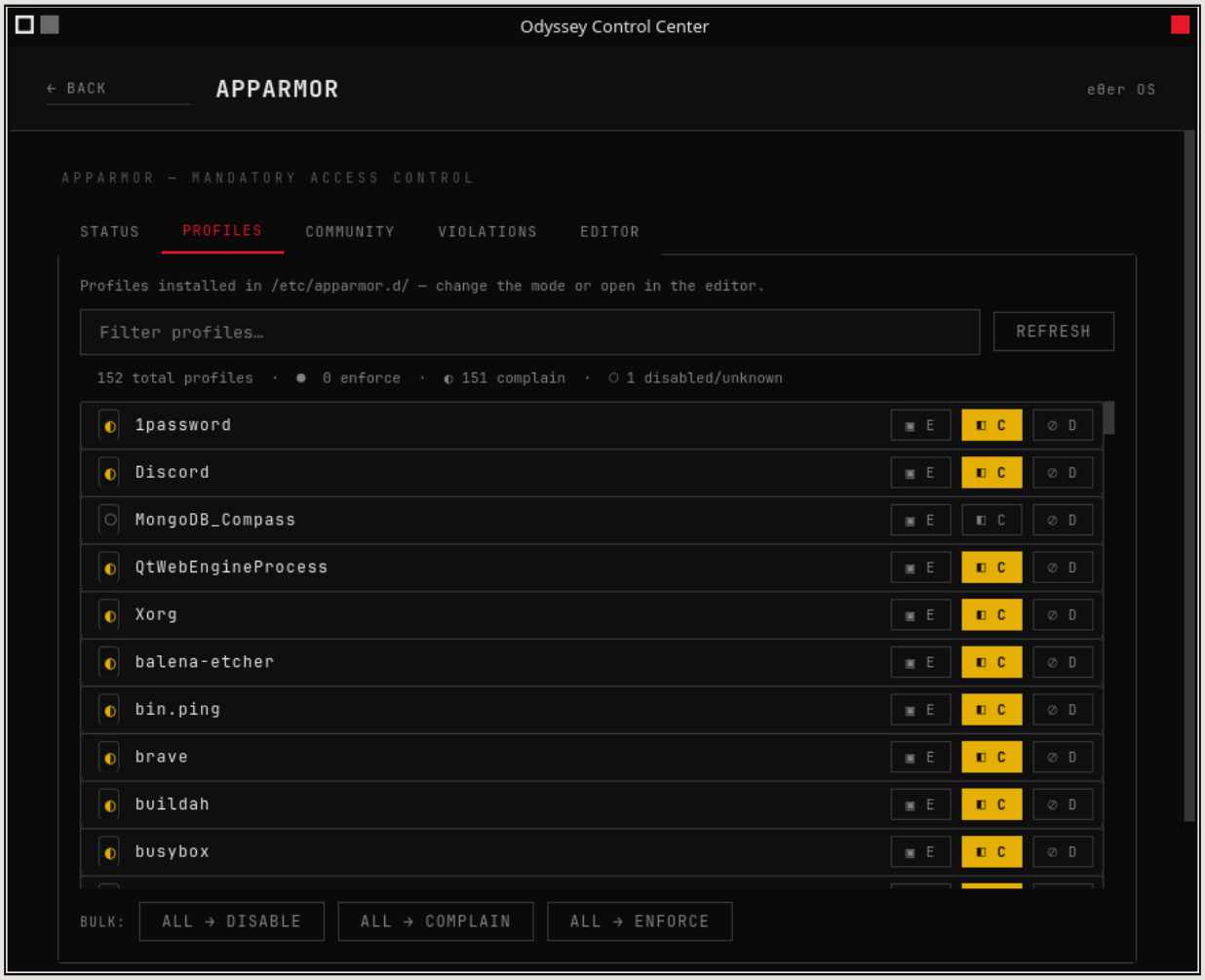


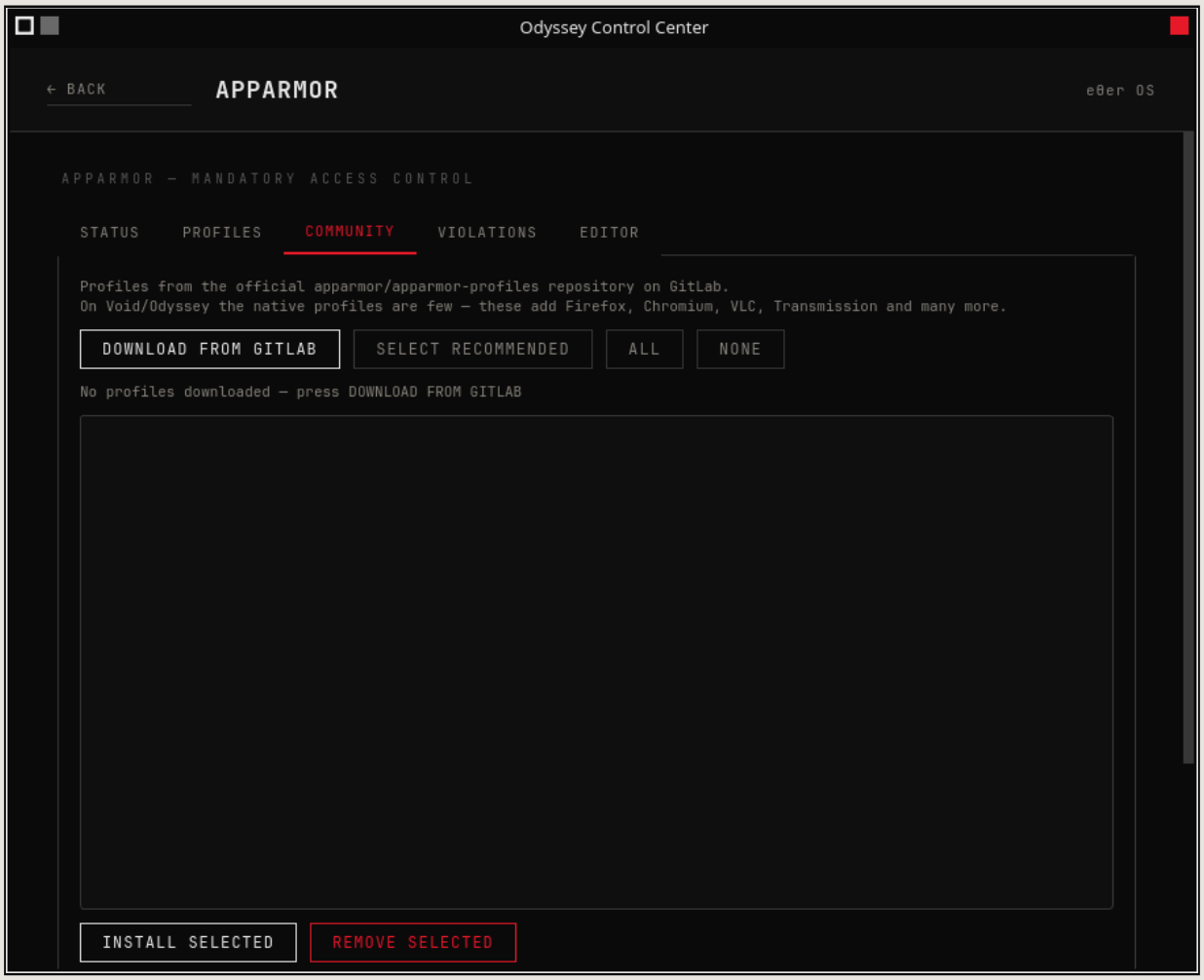
FIG · PROFILES タブ：合計 152 個のプロファイル、0 が enforce、151 が complain、1 が disabled/unknown—1password、Discord、 MongoDB_Compass などを含むフィルタリング可能なリストで、それ ぞれに E / C / D ボタンがあり、下部には ALL→DISABLE / ALL→COMPLAIN / ALL→ENFORCE。

読み込まれた各プロファイルにひとつずつ行があり、上部にはフィル ターフィールドとカウントがあります（スクリーンショットでは： 152 個のプロファイルのうち一つを除くすべてが complain モード—— これが Odyssey のデフォルトです）。行ごとに三つのボタンがあり ます：

ボタン	効果
E	このプロファイルを enforce に切り替えます——違反はブロックされ、単に記録されるだけではなくなります
C	complain に切り替えます——違反を記録しますが、何もブロックしません
D	プロファイルを完全に無効化します——プログラムは制限なしで実行されます

下部の三つの **BULK** ボタンは、リスト内のすべてのプロファイルに 一度に同じ切り替えを適用します——COMMUNITY からまとめてインス トールした直後に便利ですが、ALL→ENFORCE については特に注意して 扱ってください。次のセクションのアクション解説を参照してください。

6.3 ビジュアルガイド—COMMUNITY



FIG・COMMUNITY タブ：「No profiles downloaded — press DOWNLOAD FROM GITLAB」、DOWNLOAD FROM GITLAB、SELECT RECOMMENDED、ALL、NONE のボタン、そして下部に INSTALL SELECTED / REMOVE SELECTED。

標準の AppArmor が主に制限するのはシステムのコンポーネントです ——あなたが本当に気にかけているデスクトップアプリケーション （ブラウザ、チャットアプリ、メディアプレーヤー）には、コミュニ ティ提供のプロファイルが必要です。このタブは、GitLab 上の公式 リポジトリ `apparmor/apparmor-profiles` から直接それらを取得 します。

1 **DOWNLOAD FROM GITLAB**

現在利用可能なプロファイルの リストを取得します——まだ何もインストールされません。これは リストを埋めるだけです。

2 **SELECT RECOMMENDED（または ALL / NONE）**

デスクトップ 向けの妥当なデフォルトセットにチェックを入れます——Firefox 系の ブラウザ、Chromium、VLC、Transmission など。

3 **INSTALL SELECTED**

チェックしたプロファイルをインス トールします——他のすべてと同様、complain モードで読み込まれ、下記の「観察してから適用する」というサイクルの準備が整います。

WHAT	AppArmor → COMMUNITY → DOWNLOAD FROM GITLAB → SELECT RECOMMENDED（または個別に選択）→ INSTALL SELECTED。
WHY	信頼できない入力に最もさらされるアプリケーション——悪意 のあるページをレンダリングするブラウザ、細工されたファイル を開くメディアプレーヤー——こそ、プロファイルが最もよく保護 してくれる対象です。
RESULT	新しいプロファイルが PROFILES に complain モードで 表示され、まだ何もブロックせずに、ブロックしていたはずの 内容を記録します。
RISK	インストール時にはリスクはありません——PROFILES で あなたが決定するまで、何も適用されません。

6.4 ビジュアルガイド—VIOLATIONS

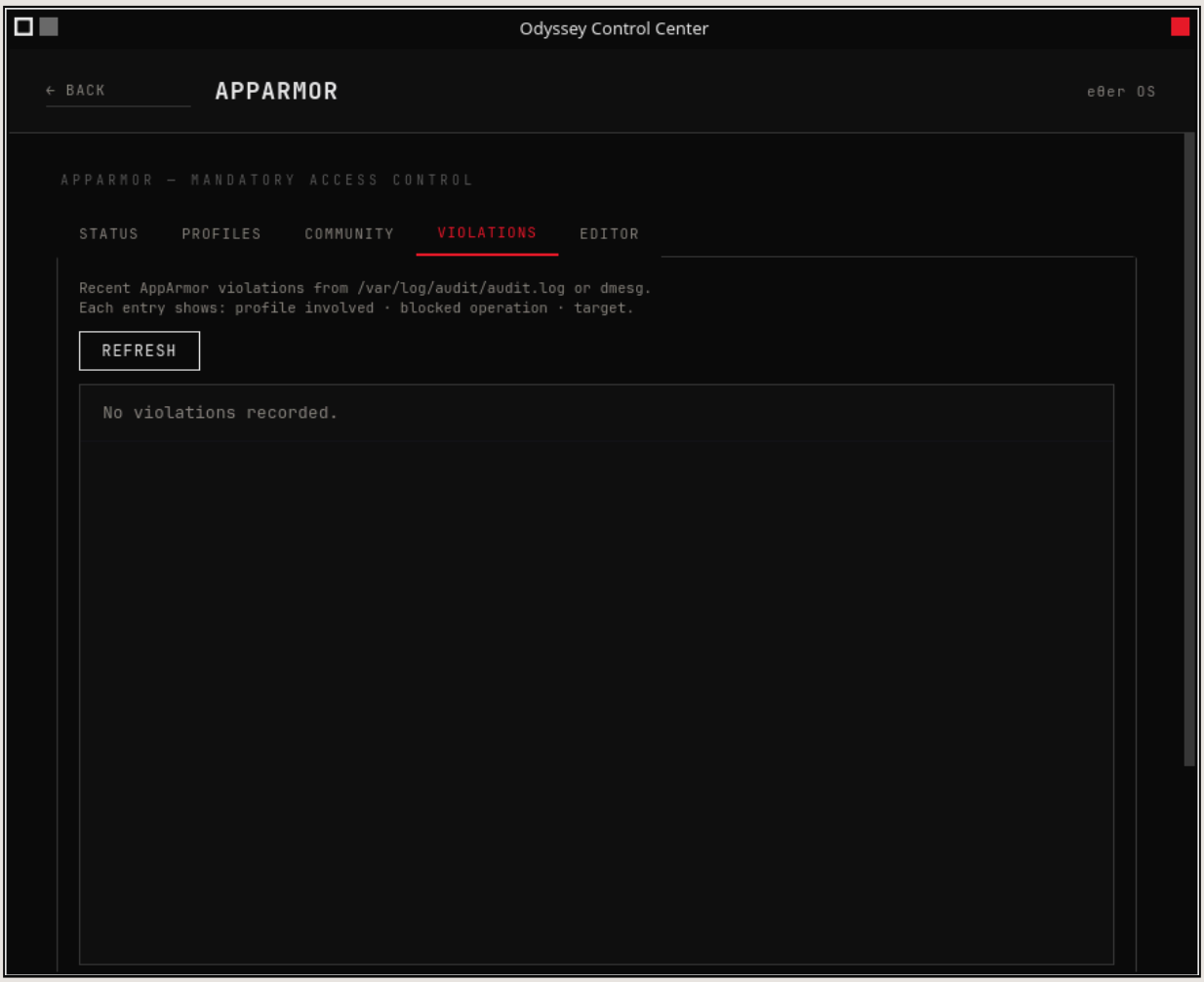


FIG · VIOLATIONS タブ：「No violations recorded」、REFRESH ボタン、 そしてエントリが /var/log/audit/audit.log または dmesg から 得られるというメモ。

このタブは、システムの AppArmor 監査ログを読み取り、プロファイル が（complain モードで）**ブロックしていたはずの**内容、あるいは（enforce モードで）**実際にブロックした**内容を表示します——各 エントリには、プロファイル、ブロックされた操作、対象となった ファイルまたはリソースが示されています。

> **健全なシステムでは、空のリストが想定どおりの表示です**
>
> スクリーンショットは、通常あなたが目にすることになる、まさに その様子を示しています：違反なし。これはバグでも、パネルが 正しく動作していない兆候でもありません——前回のログ確認以降、プロファイルの適用されたすべてのプログラムが許可された範囲 内に留まってきたことを意味します。プロファイル適用済みの何かが、そのプロファイルが許可していないことをしようとした瞬間に、違反 はここに現れます。該当するプログラムを実際に使ってから戻り、 REFRESH を押してください。

エントリが表示されるようになったら、そのうちのひとつをダブル クリックすることが、その正確なプロファイルがすでに開かれた状態 で EDITOR に入るための近道です——詳しくは下記を参照してください。

6.5 ビジュアルガイド—EDITOR

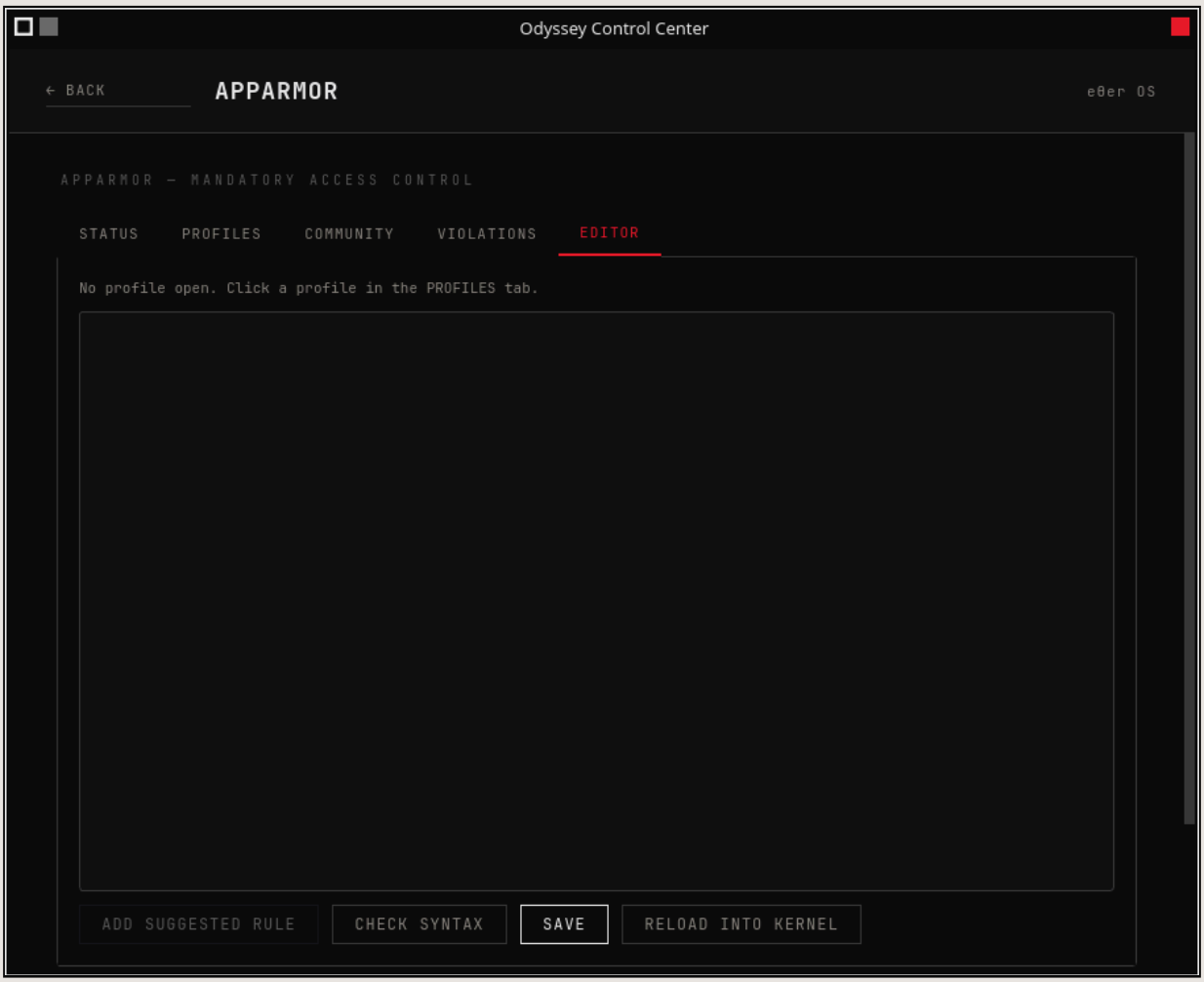


FIG · EDITOR タブ：「No profile open. Click a profile in the PROFILES tab」、空のテキストエリア、そして四つのボタン— ADD SUGGESTED RULE、CHECK SYNTAX、SAVE、RELOAD INTO KERNEL—SAVE 以外はすべて無効化されています。

エディターは意図的に**空の状態**で開きます。表示されているとおり です——一度にひとつのプロファイルしか編集せず、どれを編集するか は PROFILES（行をクリック）または VIOLATIONS（エントリをダブル クリック）から選択します。これにより、誤って間違ったファイルを 編集してしまうという単純なミスを防ぎます。

ボタン	動作
ADD SUGGESTED RULE	VIOLATIONS のエントリから来た場合にのみ有効——記録されたアクションを許可していたはずの、正確なルールの行を挿入します
CHECK SYNTAX	プロファイルをロードせずに検証します——問題になる前にタイプミスを捕捉します
SAVE	編集内容をディスク上のプロファイルファイルに書き込みます
RELOAD INTO KERNEL	保存済みの編集内容を、次回の起動やサービスの再起動を待たずに即座に有効にします

WHAT	PROFILES → プロファイル名をクリック → EDITOR がそれを 開く → 編集を行う → CHECK SYNTAX → SAVE → RELOAD INTO KERNEL。
WHY	これが、閉じ込め（confinement）の完全なサイクルを一行で 表したものです：complain モードで観察し（VIOLATIONS）、本当に 不足している部分を修正し（EDITOR）、それからようやくそのプロ ファイルを enforce に切り替える（PROFILES）。
RESULT	プロファイルは、再起動なしで、編集したとおりに正確に ライブで動作します。
RISK	プロファイルを誤って編集すると、後で enforce の下で 正当な何かをブロックしてしまう可能性があります——CHECK SYNTAX は SAVE の前に不正な形式のルールを捕捉しますが、論理的な誤りは捕捉できません。編集したプロファイルが enforce の下で 問題を引き起こし始めた場合は、PROFILES からそれを complain に 戻し、VIOLATIONS を再度確認してください。

▸ ADVANCED

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

6.6 Advanced

コマンドラインでの等価な操作

```
$ sudo aa-status
# PROFILES タブのヘッダーと同じ集計数値を、ターミナルから
$ sudo aa-enforce /etc/apparmor.d/PROFILE_FILE
$ sudo aa-complain /etc/apparmor.d/PROFILE_FILE
$ sudo aa-disable /etc/apparmor.d/PROFILE_FILE
# 手動編集の後の RELOAD INTO KERNEL に相当
$ sudo apparmor_parser -r /etc/apparmor.d/PROFILE_FILE
```

生の監査ログを読む

```
$ sudo grep apparmor /var/log/audit/audit.log | tail -20
# または、専用の監査デーモンが稼働していない場合：
$ sudo dmesg | grep -i apparmor
```

一行を手動で読む際に知っておくと役立つフィールド： `apparmor="DENIED"`（enforce）または `apparmor="ALLOWED"`（complain、記録のみ）、`operation=`（試みられた操作）、`profile=`（判断を下したプロファイル——EDITOR が開くものに対応）、`name=`（パスまたはリソース）、`comm=`（プロセス）。

一括の ALL→ENFORCE がリスクを伴う理由

VIOLATIONS のログがしばらく静かな状態を保った後に、プロファイル を個別に段階的に引き上げていくのが安全なパターンです——通常の 使用ではそれが発動しないことを、プロファイルごとに確認できるから です。ALL→ENFORCE は、すべてのプロファイルについてこの観察期間 を一気に飛ばしてしまいます。COMMUNITY からインストールしたばかり で、実際にはまだ一度も使っていないプロファイルも含めてです。実際に使われる前に強制されたプロファイルは、あなたがそれを初めて 使ったまさにその瞬間に、正当な機能をブロックしてしまう可能性が あります。ひとつずつ進めるサイクルを優先し、一括操作はALL→COMPLAIN（常に安全——緩めるだけです）や、トラブルシュー ティング時の ALL→DISABLE のために取っておいてください。

このパネルが触れるファイル

パス	役割
`/etc/apparmor.d/`	制限された各プログラムに対応するプロファイルファイル——PROFILES が一覧表示し、EDITOR が開くもの
`/etc/default/apparmor`	起動時のデフォルトモード（`APPARMOR=complain`）と enable/disable の状態
`/etc/default/grub`	STATUS がチェックするカーネルパラメーター `apparmor=1 security=apparmor`

CHAPTER 7

Privacy

二つの独立したツールがあり、それぞれ install → enable → running という独自の三段階の流れを持ち、両方が稼働し始めた時点でそれらを 橋渡しする仕組みがあります。

7.1 ビジュアルガイド—SearXNG を段階的に

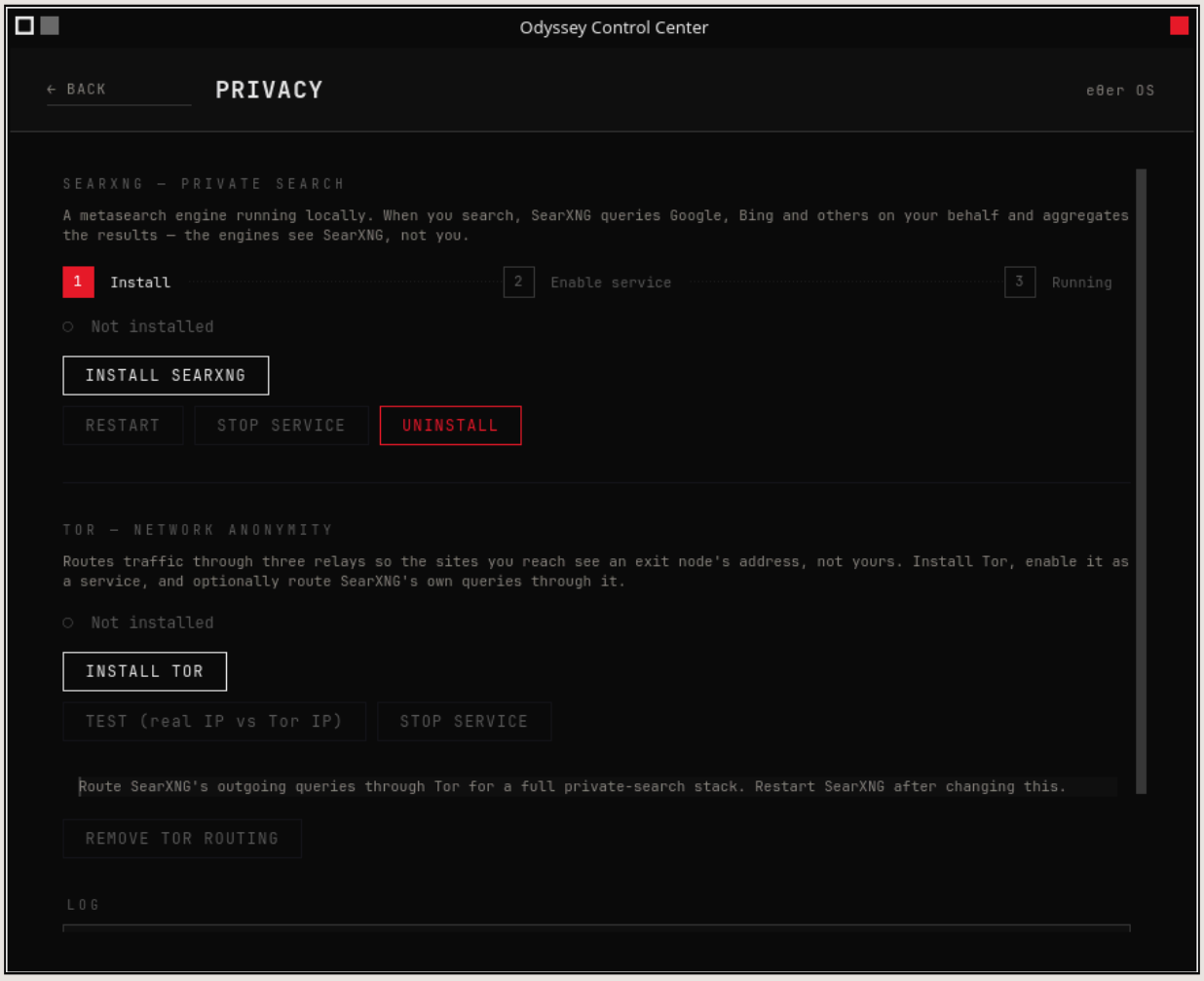


FIG · 静止状態のパネル：SearXNG のトラッカーはステップ 1「Install」、 ステータスは「Not installed」、ボタンは INSTALL SEARXNG ひとつ だけ。Tor のトラッカーも同様に「Not installed」で、その下に INSTALL TOR。

SearXNG は、あなた自身のマシン上でローカルに動作するメタ検索 エンジンです——あなたが検索するとき、Google や Bing などにあなた の代わりに問い合わせるのは SearXNG です。そのため、これらの検索 エンジンから見えるのは、SearXNG がリクエストを行っている姿だけ で、あなた自身が直接それを行っているようには決して見えません。

1 **INSTALL SEARXNG**

クリックします。ログには実際の インストール作業の進行が表示されます——ソースコードのクロー ン、システムユーザーの作成、Python 仮想環境のセットアップなど。

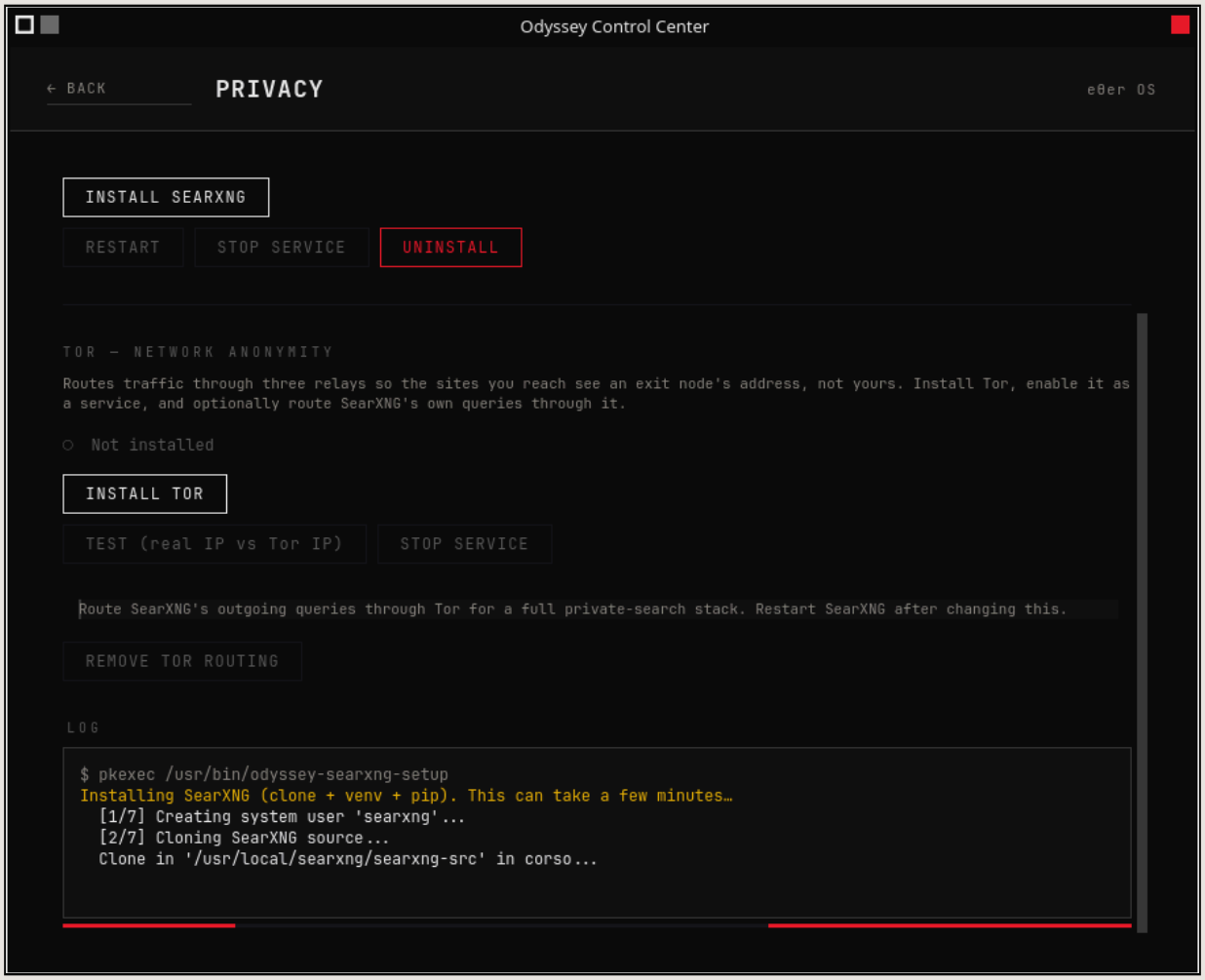


FIG · インストールの途中：ログが実際の出力を送信中—「Installing SearXNG (clone + venv + pip). This can take a few minutes…」 で、ステップ 1/7 と 2/7 が見えています。

2 ENABLE SERVICE

インストールが完了すると、トラッカー はステップ 2 に進み、ステータス行が黄色になります：「Installed — service not enabled」。ここではボタンはひとつだけです： ENABLE SERVICE。

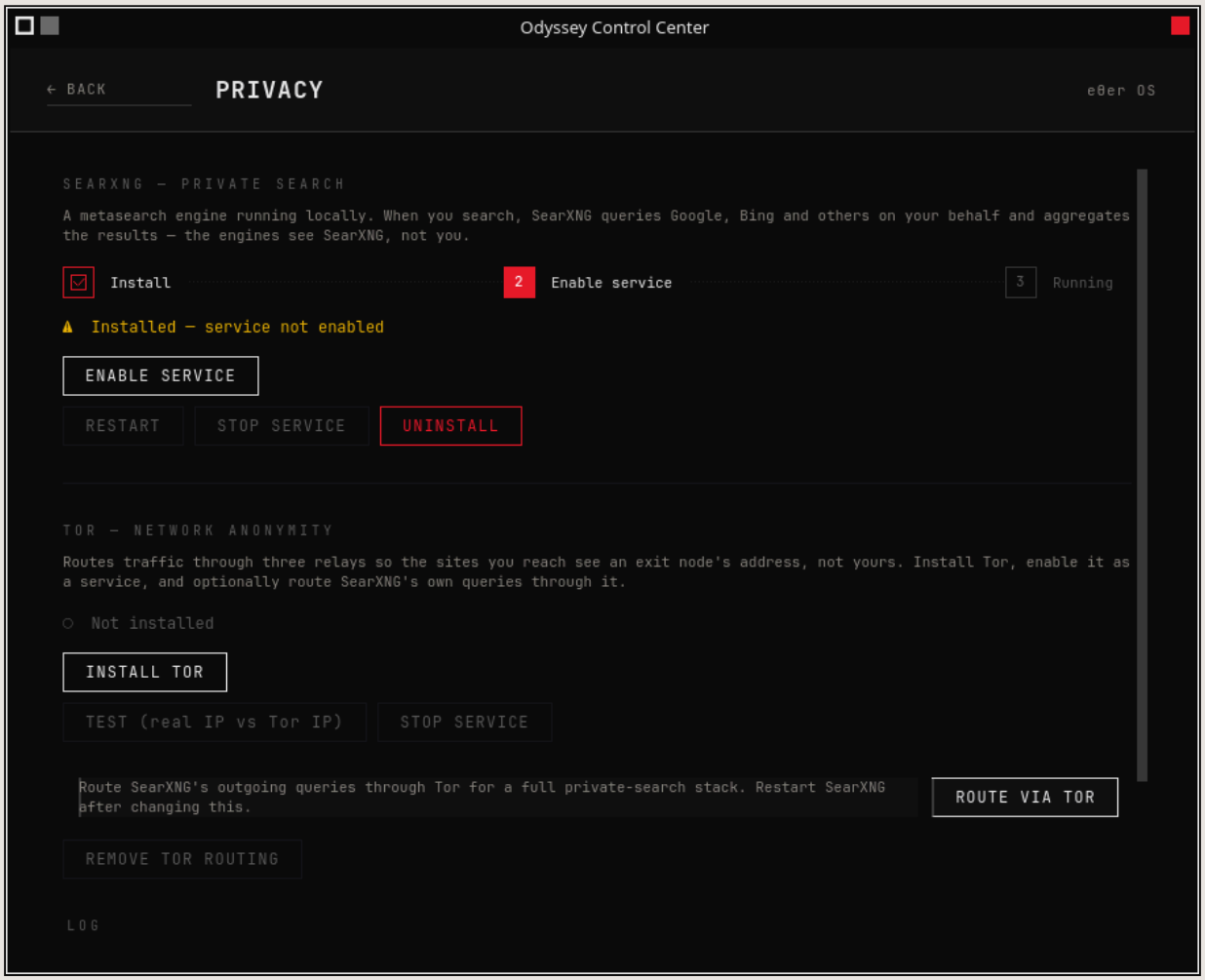


FIG · ステップ 2 がアクティブ：「Installed — service not enabled」、 唯一有効なボタンとして ENABLE SERVICE、その下では RESTART/STOP SERVICE/UNINSTALL がまだ無効化されています。

3 Running

有効化した後、トラッカーはステップ 3 に到達 し、操作の行全体がアクティブになります。稼働し始めると、**OPEN IN BROWSER** が直接そこへ連れて行ってくれ ます——SearXNG はデフォルトで `http://localhost:8888` を待ち受け ています。

WHAT	Privacy → INSTALL SEARXNG → ENABLE SERVICE → OPEN IN BROWSER。
WHY	あらゆる商用の検索エンジンは、時間の経過とともにあなたの クエリからプロファイルを構築していきます。ローカルなメタ 検索のインスタンスは、集約されていないクエリを見ることが できるのがあなただけであることを意味します。
RESULT	あなた自身のマシン上で動作する、ブラウザからアクセス 可能なプライベートな検索エンジンが、あなたの代わりに大手検索 エンジンに問い合わせを行いつつ、あなた自身をそれらに直接さら すことはありません。
RISK	低い——これは他のものと同様の、ローカルな web サービス です。挙動がおかしければ RESTART、削除せずに一時停止したい 場合は STOP SERVICE、完全に削除したい場合は UNINSTALL。

7.2 ビジュアルガイド—Tor を段階的に

同じ三段階の形式で、独立したトラッカーが、SearXNG のセクションの すぐ下にあります。

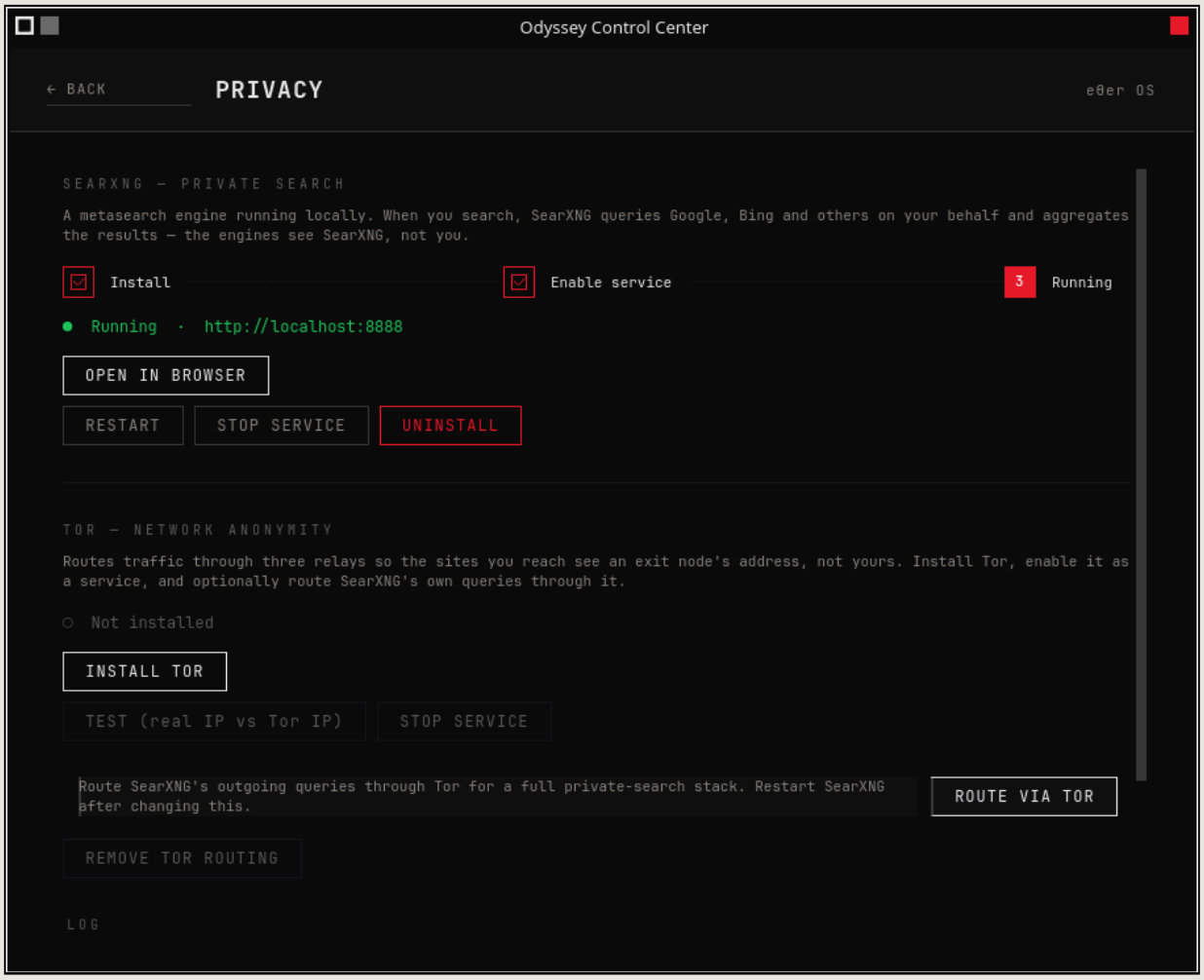


FIG · 上では SearXNG が完全に稼働中。下の Tor セクションはまだ 「Not installed」で、唯一有効なボタンとして INSTALL TOR。

Tor は、あなたのトラフィックが目的地に到達する前に、三つの中継 ノードを経由させます。そのため、あなたが訪れるサイトが見るのは 出口の中継ノードのアドレスであり、あなた自身のものではありません。

1 **INSTALL TOR**

ログはパッケージのインストールを確認し、 サービスが runit にリンクされたことを示します。

2 **ENABLE SERVICE**

ステータスが「Installed — service not enabled」に変わります——SearXNG と同じパターンです。

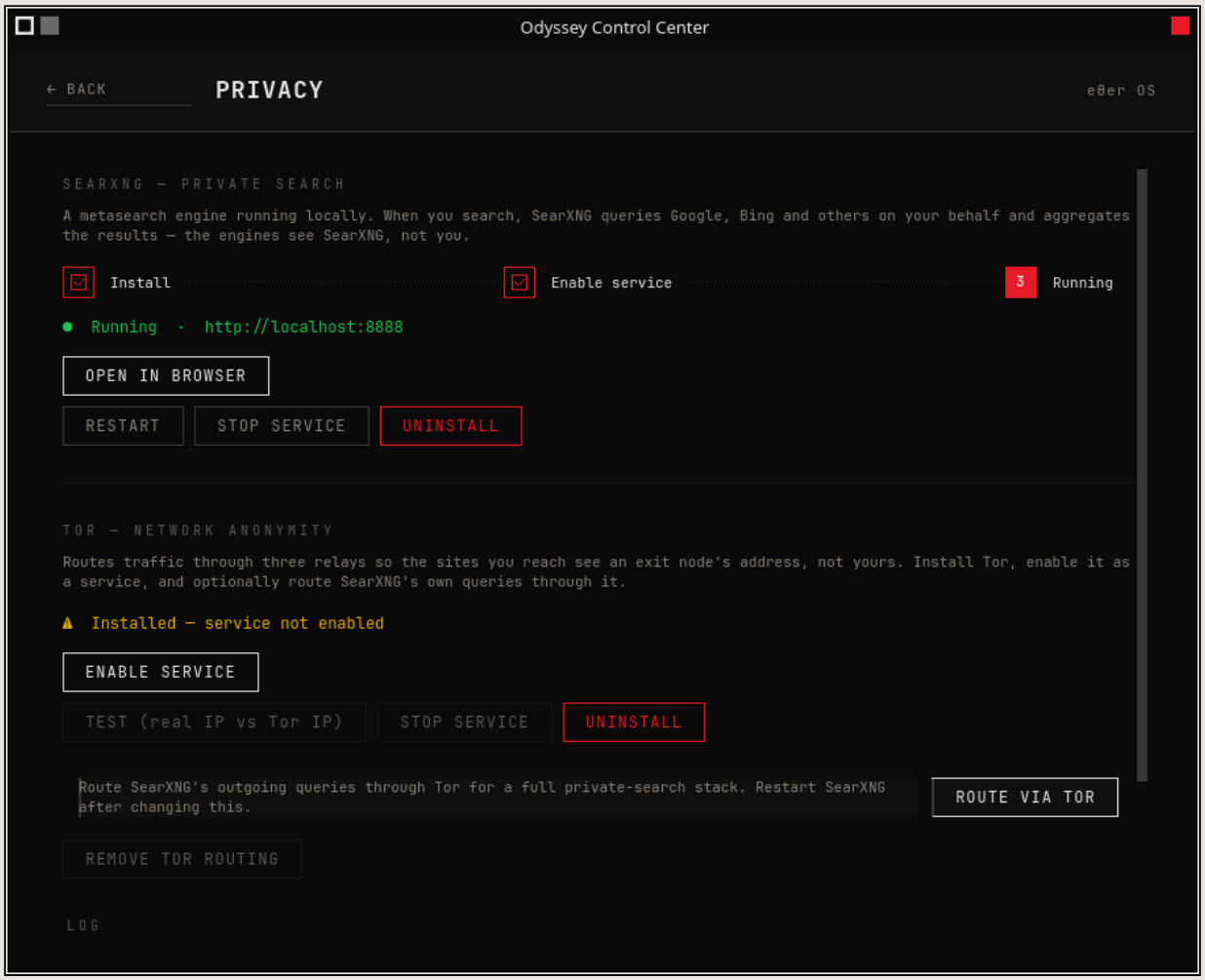


FIG · ステップ 2 の Tor：「Installed — service not enabled」、 ENABLE SERVICE が有効、TEST と STOP SERVICE はまだ無効化され ています。

3 Running、そしてそれを検証する

有効化すると、専用の **TEST (real IP vs Tor IP)** ボタンが表示されます——それを使って ください。あなたの実際のアドレスと、Tor が提示しているアドレス を並べて比較することで、ただ信じるしかないステータス表示では なく、それが実際に機能しているという実際の証拠が得られます。

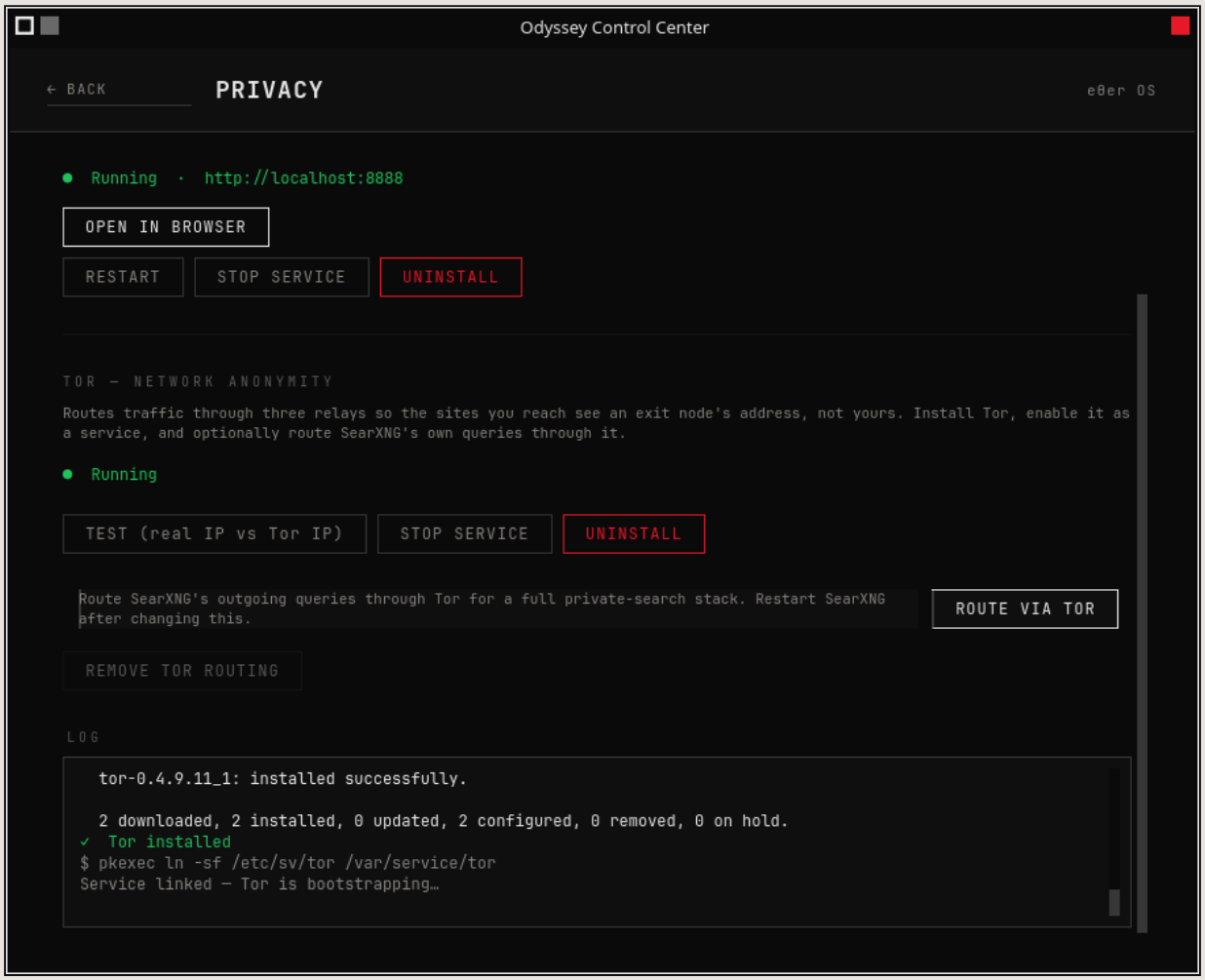


FIG · 両方のセクションが緑色で稼働中：SearXNG は localhost:8888 で OPEN IN BROWSER 付き、Tor は稼働中で TEST / STOP SERVICE / UNINSTALL がすべて有効。

WHAT	Privacy → INSTALL TOR → ENABLE SERVICE → 確認のために TEST (real IP vs Tor IP)。
WHY	機密性の高い閲覧を Tor 経由でルーティングすることで、 訪れるサイトからあなたのアドレスを隠すことができます——TEST は単なる主張ではなく、証拠を与えてくれます。
RESULT	他のアプリケーションが経由できる、稼働中の Tor サービス。うまく機能していれば、TEST は本当に異なる二つの アドレスを表示します。
RISK	Tor 経由のトラフィックは明らかに遅くなります——これが 匿名性の代償です。STOP SERVICE はそれを一時停止し、 UNINSTALL はそれを削除します。

7.3 ビジュアルガイド—両者を組み合わせる

両方が稼働し始めると、SearXNG **自身**のクエリも Tor 経由でルー ティングすることを提案する、インラインの通知が現れます——あなた が SearXNG に対して行うリクエストだけでなく、SearXNG があなたの 代わりに Google や Bing に対して行うリクエストもです。

WHAT	Privacy → 「Route SearXNG's outgoing queries through Tor」の通知 → ROUTE VIA TOR → 有効にするために SearXNG セクションで RESTART。
WHY	これを行わないと、SearXNG 自体は依然として大手検索 エンジンに直接問い合わせを行います——あなたの側からは プライベートですが、相手側からは見えています。SearXNG 自身の トラフィックを Tor 経由でルーティングすることで、完全に プライベートな検索スタックを実現するために、この隙間を 埋めます。
RESULT	あなたから SearXNG へのクエリだけでなく、SearXNG の 発信するクエリそのものが Tor を経由するようになります。
RISK	Tor 経由のトラフィックすべてに共通する理由により、結果の 読み込みが明らかに遅くなります。REMOVE TOR ROUTING はこれを 元に戻します。ここでも再起動が必要になります。

▸ ADVANCED

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

7.4 Advanced

INSTALL SEARXNG が実際に実行する内容

```
$ pkexec /usr/bin/odyssey-searxng-setup
# 専用のシステムユーザー 'searxng' を作成し、
# ソースコードを /usr/local/searxng/searxng-src にクローンし、
# Python の venv を構築して pip で依存関係をインストールします
```

サービスとポート、手動で

```
# SearXNG 自身の runit サービス
$ sudo sv status searxng
$ sudo sv restart searxng
# Tor の runit サービス—パネルの有効化ステップは文字どおりこれです
$ sudo ln -sf /etc/sv/tor /var/service/tor
$ sudo sv status tor
# Tor が実際にルーティングしていることを確認
$ curl -socks5 localhost:9050 https://check.torproject.org/api/ip
```

SEARXNG を手動で TOR 経由にルーティングする

Tor の SOCKS プロキシは、デフォルトで `127.0.0.1:9050` を待ち 受けています。SearXNG をこれ経由にルーティングするということは、 発信 HTTP クライアントをこのプロキシに向けることを意味します—— この設定は、システム全体のプロキシ変数ではなく、SearXNG 自身の `settings.yml` ファイルの中の、発信リクエストの設定に存在します。パネルの ROUTE VIA TOR ボタンは、このファイルを直接編集し、サービスを再起動します。手動で行う場合は、 `/usr/local/searxng/searxng-src/searx/settings.yml` の中の同じ `outgoing:` ブロックを見つけて編集することになります。

ルーティングの変更後に SEARXNG が再起動を必要とする理由

SearXNG は、起動時に一度だけプロキシ設定を読み込みます——実行中 にホットリロードできる設定ではありません。これは、パネル経由で 変更しても手動で変更しても変わらない事実であり、それゆえどちら の経路も同じ RESTART のステップで終わるのです。

このパネルが触れるファイル

パス	役割
`/usr/local/searxng/searxng-src/`	SearXNG 自身のインストール内容——ソースコード、venv、設定
`/etc/sv/searxng`、`/etc/sv/tor`	runit のサービス定義
`/var/service/`	サービスを有効化すると、稼働中の監視ツリーにリンクされる場所

CHAPTER 8

Containers

最初から最後までのコンテナ管理：エンジンのセットアップ、稼働中の コンテナのリスト、始めるための厳選されたカタログ、そこにない すべてのもののための生のコマンド、そしてディスク使用量——すべて がひとつのスクロール可能なパネルの中にあります。

8.1 ビジュアルガイド—エンジン

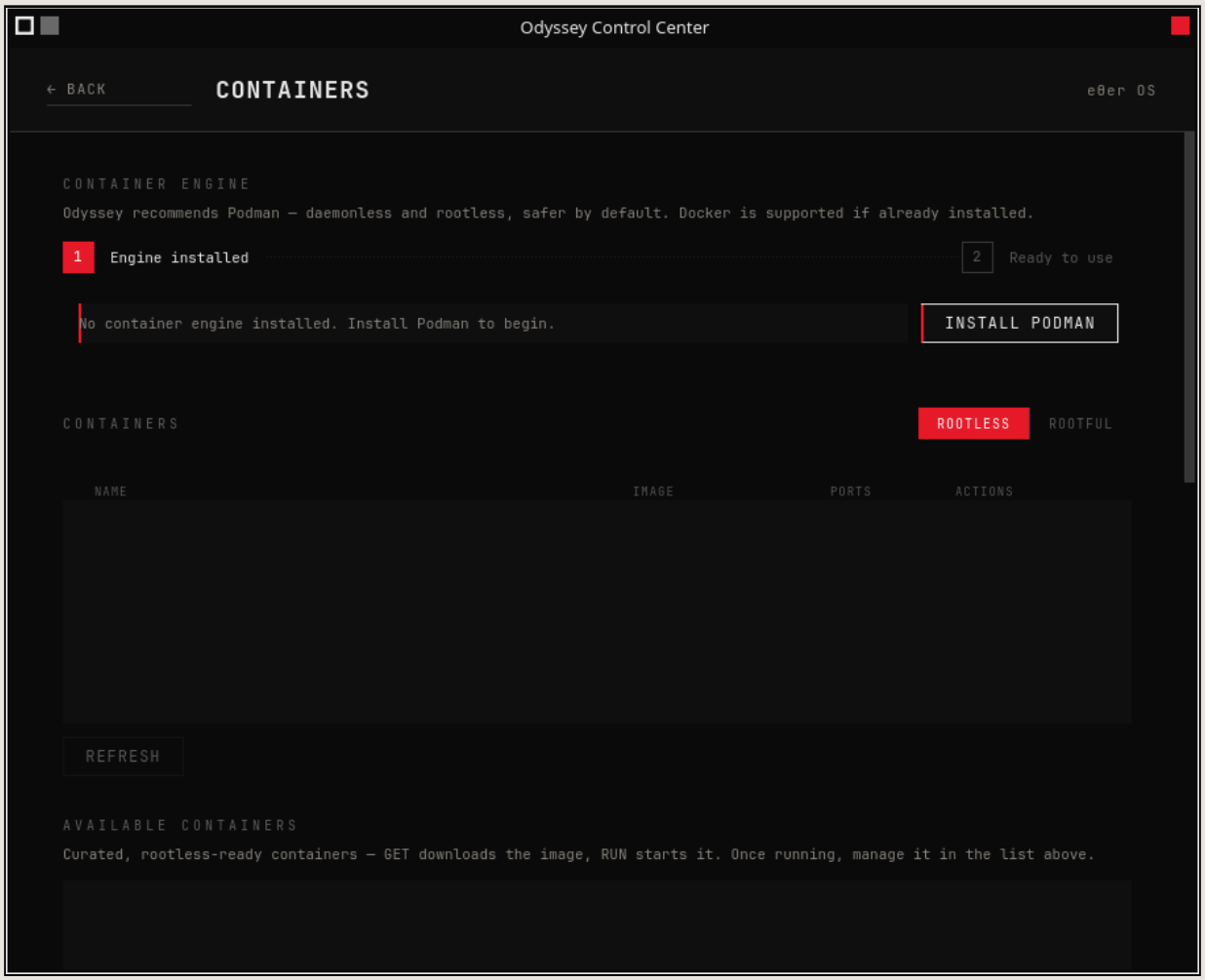


FIG · 2 ステップ中の 1 ステップ目：「No container engine installed. Install Podman to begin」、唯一有効なアクションとして INSTALL PODMAN。下の Containers list と Available containers のセクションは見えていますが、空の状態です。

Odyssey は **Podman** を推奨しています——デーモンレスで rootless、つまりコンテナは常駐するバックグラウンドサービスなしで動作し、 デフォルトでは root 権限も持ちません。すでに使っているものが それであれば、**Docker** もサポートされています。

1 **INSTALL PODMAN**

ログには実際のパッケージインストール が表示されます。

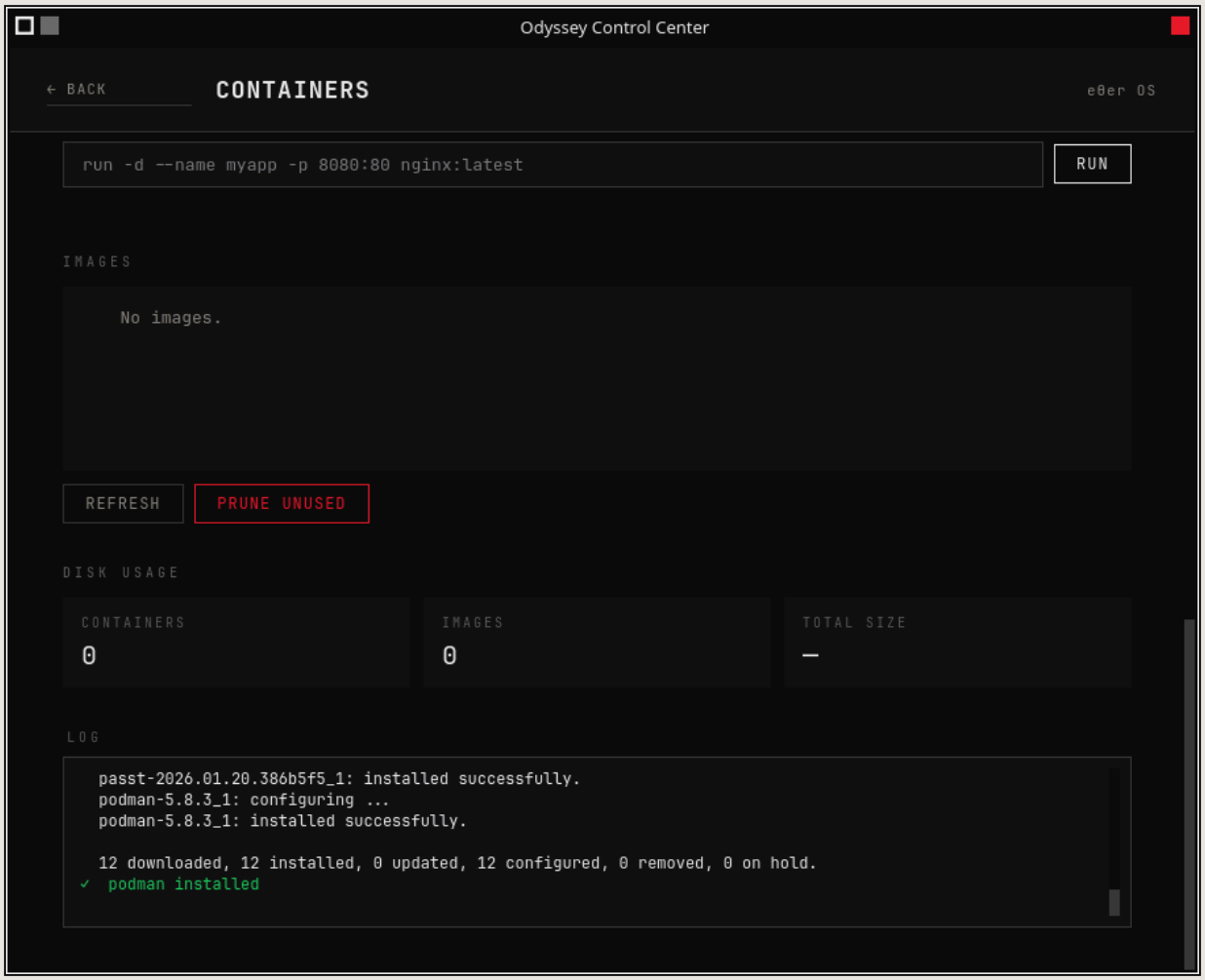


FIG · インストール後：「Engine installed」が赤くチェックされ、ステップ 2 の「Ready to use」、情報行には「podman version 5.8.3 · Podman installed · Docker not installed」と表示され、並行して Docker も欲しい人のための INSTALL DOCKER TOO ボタンがあります。

2 Ready to use

エンジンはアクティブです。両方のエンジン を並べて使いたい場合、**INSTALL DOCKER TOO** は引き続き利用 可能です——それらは競合することなく共存します。

8.2 ビジュアルガイド—rootless vs. rootful、そして空の状態

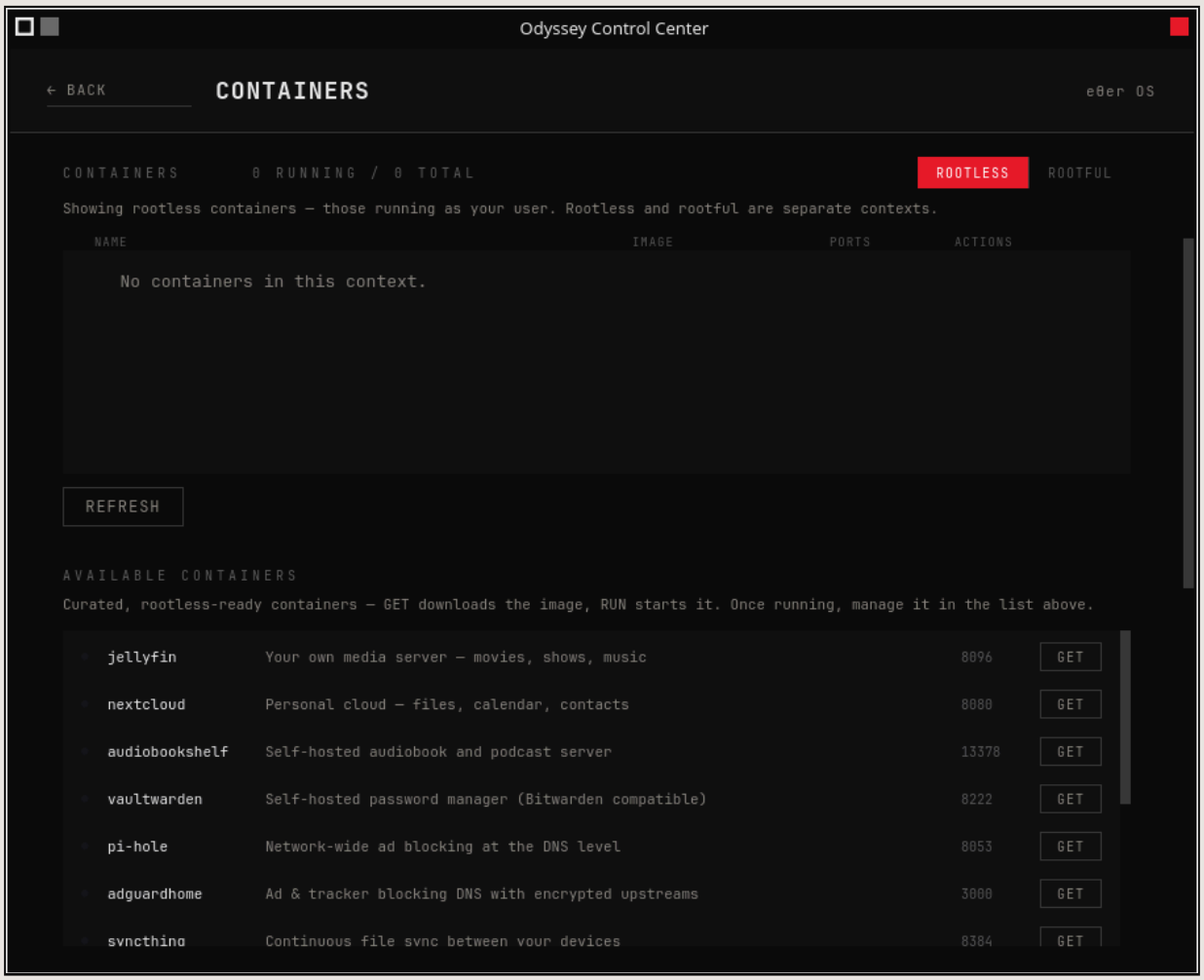


FIG · ROOTFUL の隣に ROOTLESS が選択されており（赤くハイライト）、「0 RUNNING / 0 TOTAL」、NAME / IMAGE / PORTS / ACTIONS の 列見出しの下に「No containers in this context.」。

上部の **ROOTLESS / ROOTFUL** スイッチは、あなたが表示・管理して いるコンテキストを切り替えます——この二つは完全に分離されたコン テナの名前空間であり、統合されたリストに対するフィルターではあり ません。Odyssey はデフォルトで ROOTLESS を表示します。これは 意図的なデフォルトです：

> **どちらを使うべきか**
>
> Rootless はより安全な選択であり、パネルがあなたを導く方向でも あります：rootless モードで閉じ込め（confinement）を脱出した コンテナであっても、あなた自身のユーザーアカウントがすでに できることを超えて何かができるわけではありません。Rootful は、 それを必要とする特定のケース——番号の小さい特定のネットワーク ポート、いくつかのハードウェアアクセスなど——のために存在して おり、常用ではなく例外であるべきです。

8.3 ビジュアルガイド—厳選されたカタログ、最初から最後まで

Available containers までスクロールします——すでに用意された リストがあるおかげで、パネルが、あなたがイメージ名をすでに暗記 しているのを待つ空の箱になることは決してありません。各行には、コンテナ、一行の説明、それが使用するポートが表示されます。

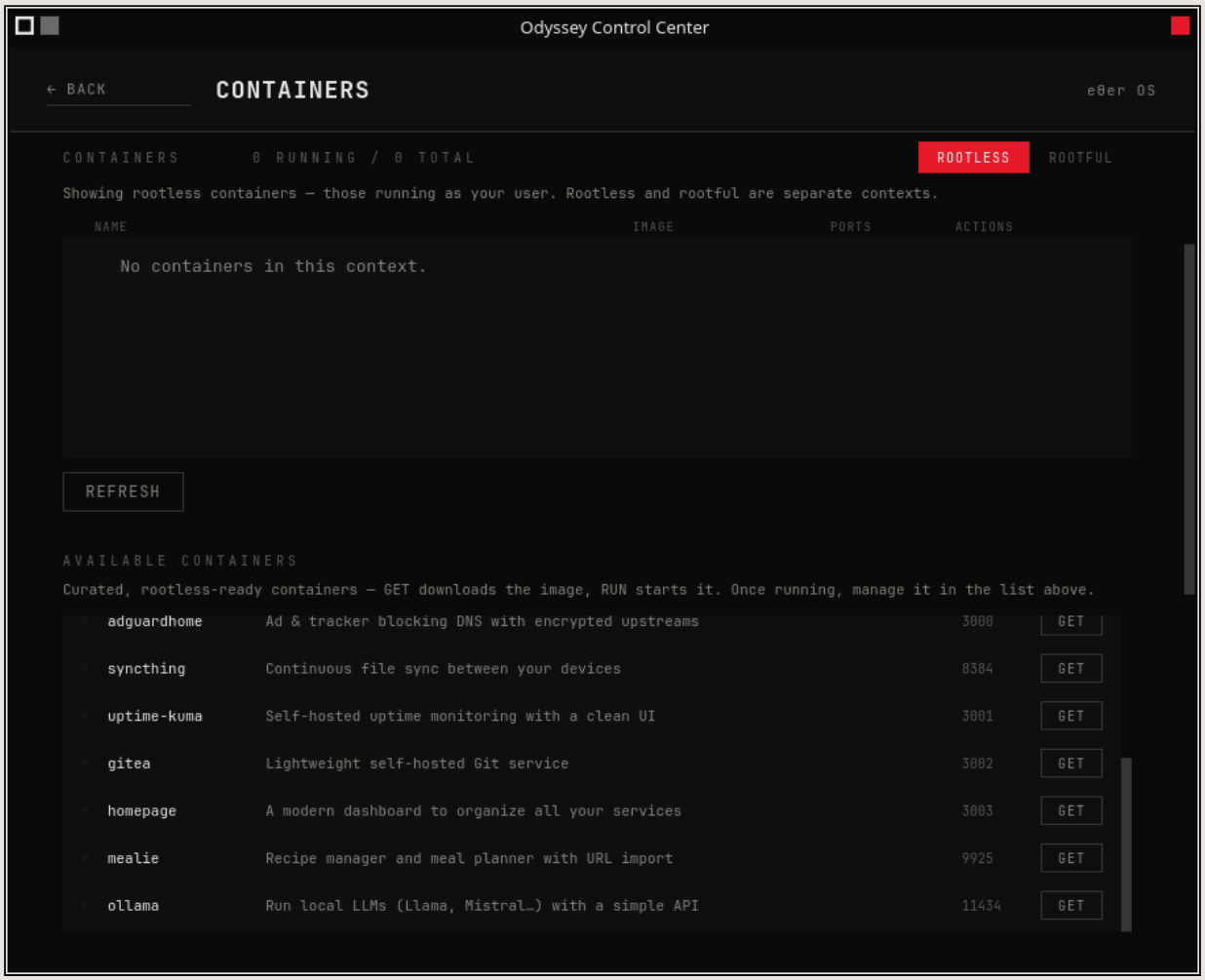


FIG · カタログのリスト：adguardhome（ポート 3000）、syncthing （8384）、uptime-kuma（3001）、gitea（3002）、homepage （3003）—各行に GET ボタン。

このポート番号は、稼働し始めた後にそのサービスにアクセスすること になるものです——たとえば `homepage` と `3003` の組み合わせは、 有効になれば `http://localhost:3003` を意味します。

1 GET

`homepage` の GET をクリックします。これは コンテナのイメージをダウンロードするだけです——まだ何も稼働 していません。

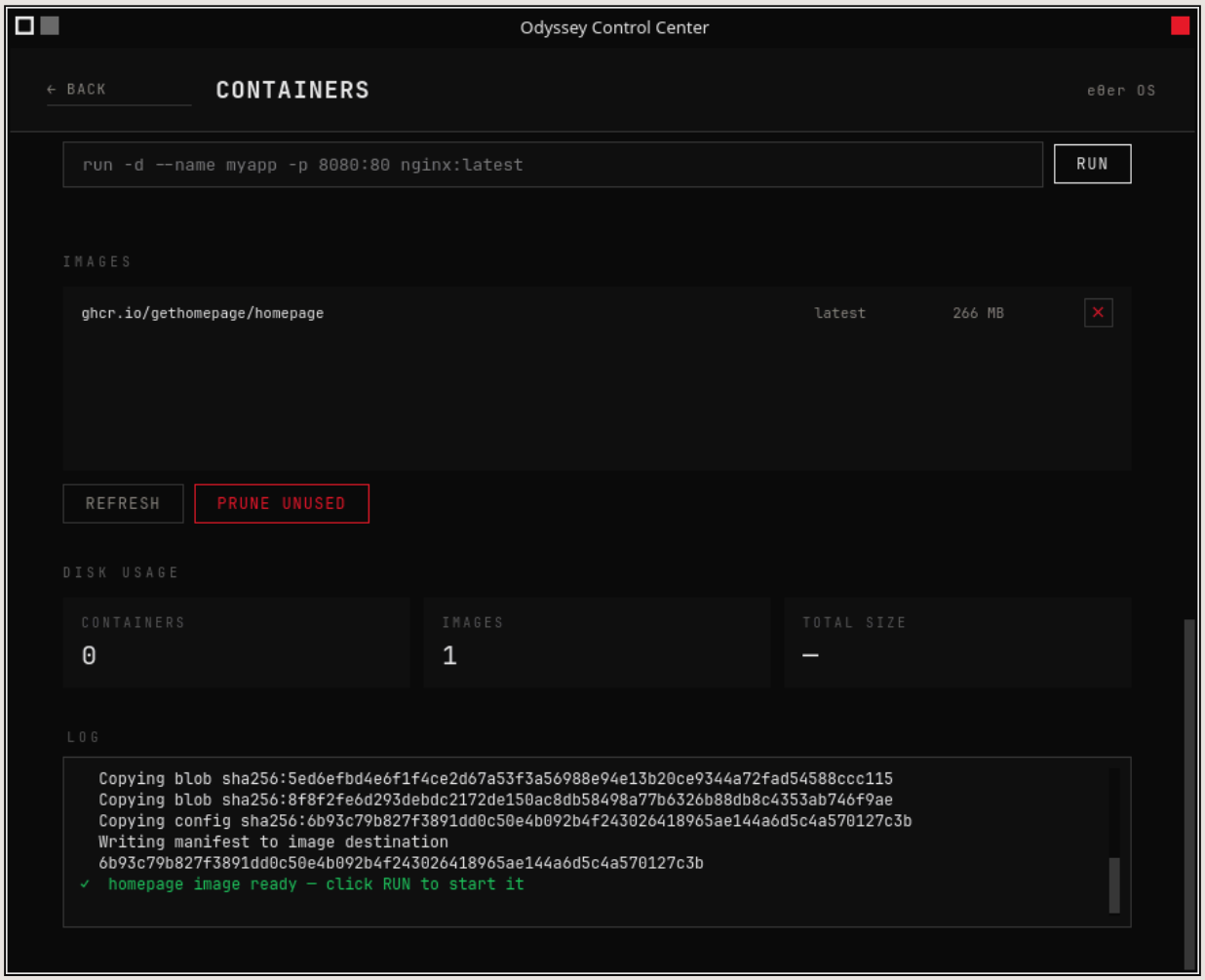


FIG · ログが実際のイメージダウンロードを送信中：「Copying blob sha256:5ed6…」、「Writing manifest to image destination」、緑色で終わります—「homepage image ready — click RUN to start it」。

2 RUN

GET ボタンがあった行には、今度は代わりに RUN が 表示されます——それをクリックして、実際にコンテナを起動します。

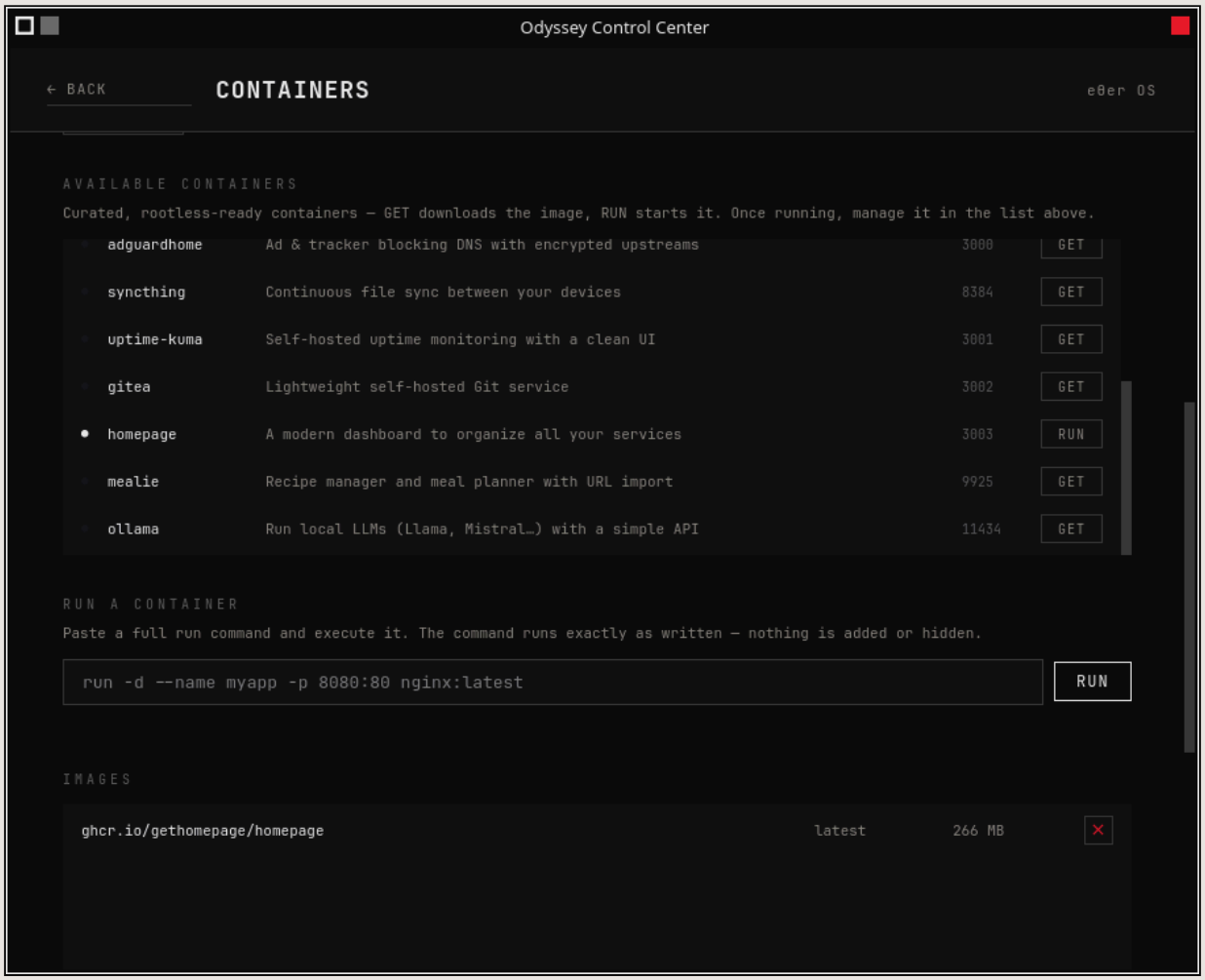


FIG · homepage が上の CONTAINERS セクションに表示されるようになり、緑色のドット、イメージパス ghcr.io/gethomepage/homepage、ポート 「0.0.0.0:3003→…」、独自のアクションアイコンの行があります。 下の Available Containers には、別のインスタンスを起動するための新しい RUN ボタン付きで、依然として表示されています。

稼働し始めると、`homepage` は同時に二つの場所に現れます：上の **Containers** リスト内のアクティブなエントリとして（そのポートと アクションアイコン付きで）、そして後で二つ目のインスタンスを 起動したい場合に備えて、依然としてカタログにも残っています。

WHAT	Containers → Available containers → カタログの中の何か に対して GET → ダウンロードが完了したら RUN。
WHY	二段階（ダウンロード、その後起動）に分かれていることに よって、すぐに実行することにコミットせずに、事前にイメージを 取得しておくことができます——回線が遅い場合や、何が利用可能かをただ眺めている場合に便利です。
RESULT	稼働中のコンテナが、そのポートマッピングとともにメイン のリストに表示され、それ以降そこから管理できます。
RISK	低い——厳選されたカタログのセルフホスト型サービスは、 デフォルトで rootless に適しており、リスクが低くなるように 選ばれています。露出面は、他の web サービスを実行する場合と同じです：どのポートで待ち受けていようと、あなたの Firewall の設定に従ってアクセス可能になります。

8.4 ビジュアルガイド—コンテナリストとその操作

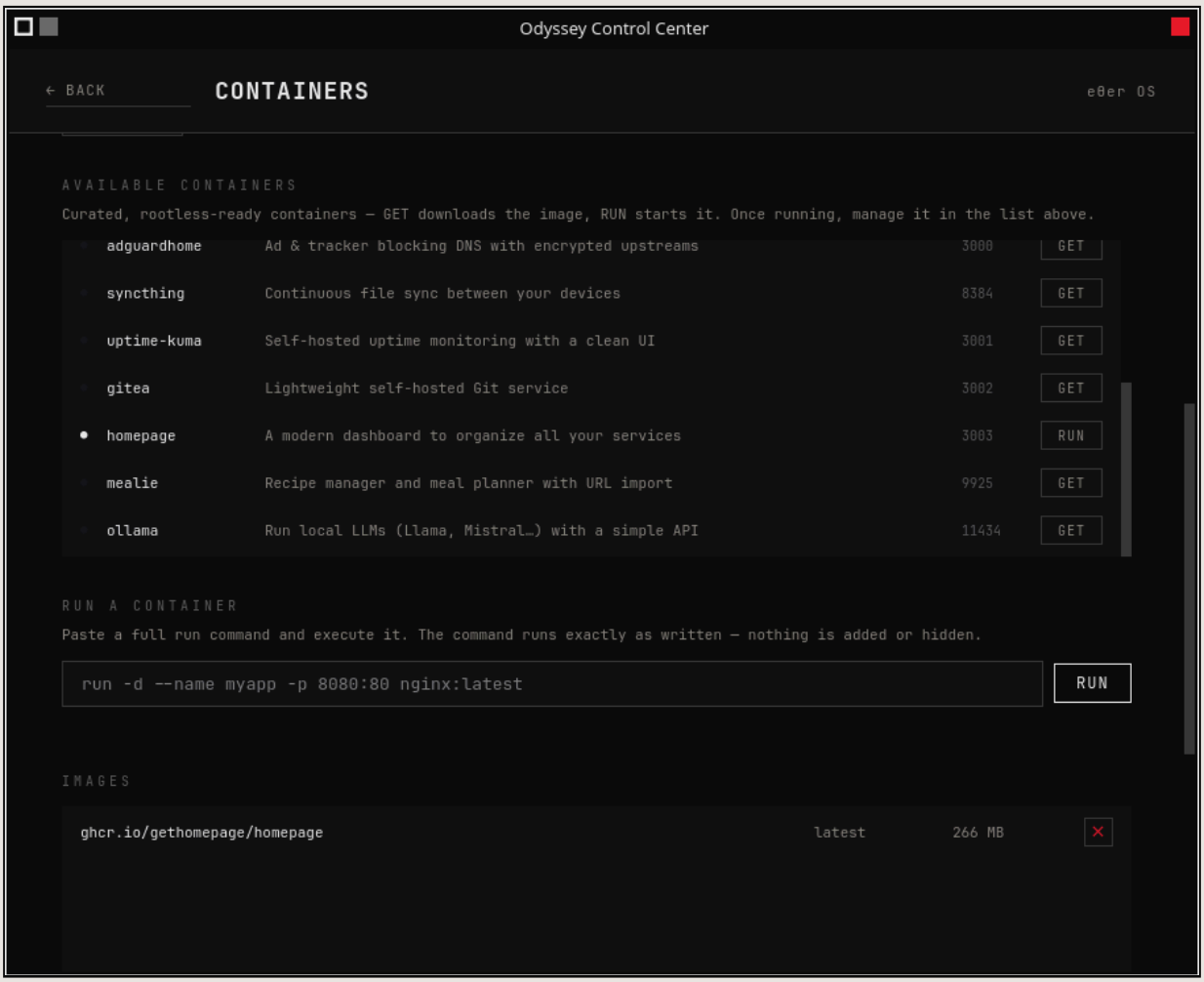


FIG · Containers リスト内の homepage の行。右側に見える五つの アクションアイコン—それぞれ以下で説明します。

稼働中のコンテナの各行は、一連のアイコンで終わります——どれかに マウスを重ねればラベルが見えますが、それぞれの機能は次のとおり です：

アイコン	操作
開く（矢印）	そのサービスを直接ブラウザで開きます——何かが稼働している状態からの近道です
停止 / 起動（四角 / 三角）	コンテナを削除せずに停止する、あるいは再び起動します
再起動（円形の矢印）	停止と起動をひとつの操作で行います——「稼働はしているが応答しない」場合の解決策です
logs（線状のアイコン）	そのコンテナ自身のログ出力を開きます——何かがうまくいっていない場合に最初に確認すべき場所です
削除（×）	コンテナを完全に削除します——イメージ自体はダウンロード済みのまま残り、Images に別途表示されます

8.5 ビジュアルガイド—コンテナを実行し、それを開く

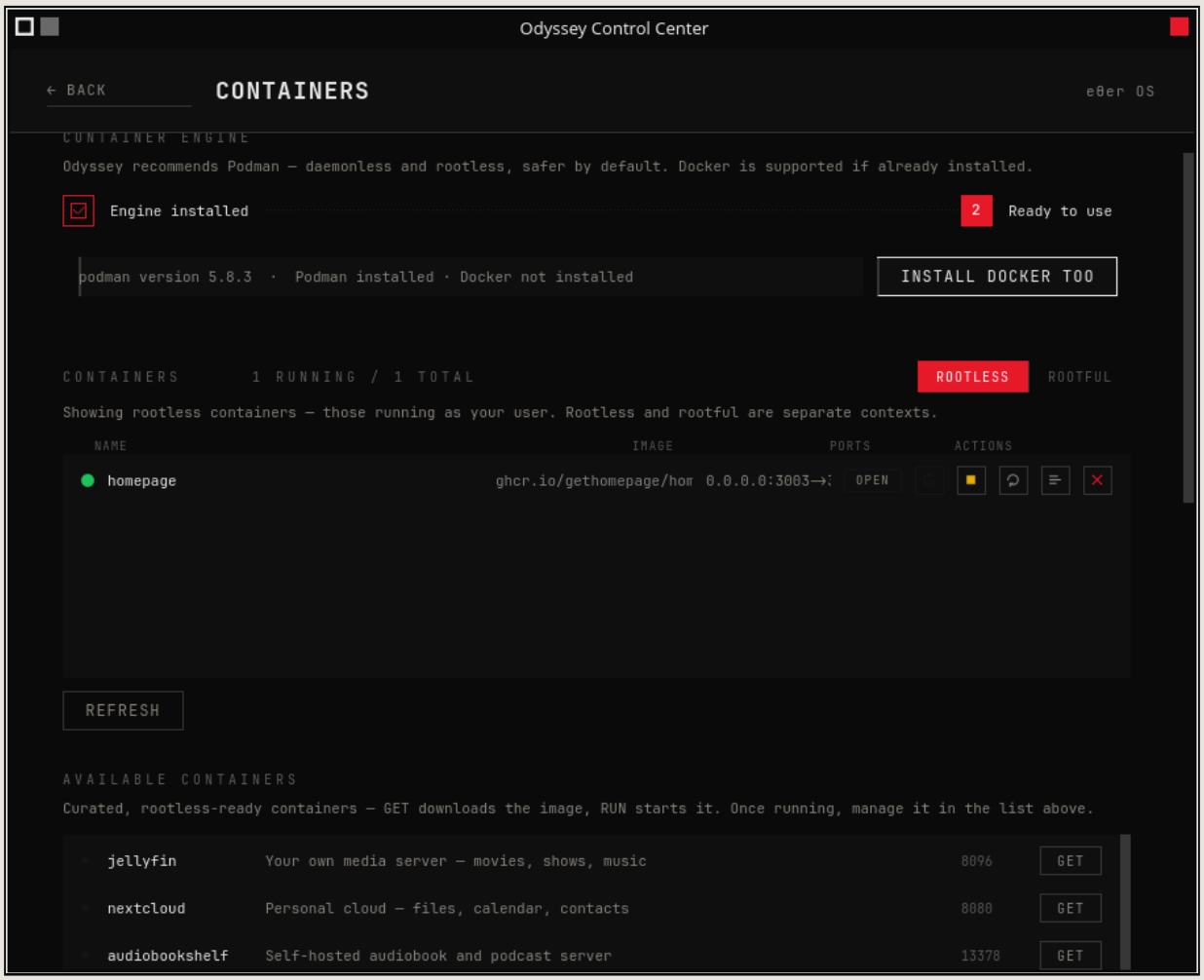


FIG · プレースホルダーテキストとしてコマンドの例が入った「Run a container」フィールド、RUN ボタン、そして下に ghcr.io/gethomepage/homepage · latest · 266 MB を削除ボタン 付きで一覧表示する Images セクション。

カタログにない任意のものに対して、**Run a container** は完全な `podman run`（または `docker run`）コマンドを受け取り、書かれた とおりに正確に実行します——何も追加されず、何も隠されません。

WHAT	Containers → 「Run a container」→ 完全な run コマンド （例：`run -d --name myapp -p 8080:80 nginx:latest`）を 貼り付ける → RUN。
WHY	厳選されたカタログは一般的なケースをカバーしています。これは、カタログが提供していない特定のイメージ、特定のフラ グ、特定のポートのための逃げ道です。
RESULT	コンテナはコマンドが指定するとおりに正確に起動し、 稼働し始めると、メインの Containers リストに加わります。
RISK	コマンド自体が何をするかに依存します——これは忠実な 受け渡しであり、サンドボックスではありません。ターミナルで 同じコマンドを入力するのと同じ注意深さで扱ってください。

下には、**Images** が、現在コンテナがそこから稼働しているかどうか に関わらず、ダウンロード済みのすべてのイメージを一覧表示し、 **REFRESH** と **PRUNE UNUSED** によって、もう誰も参照していない イメージから容量を取り戻すことができます。

稼働中のコンテナを開く

稼働中のコンテナの行で OPEN をクリックすると——上のアイコン行に ある同じ「開く」操作です——デフォルトのブラウザが直接その アドレスとポートに移動します。たとえば `homepage` であれば `http://localhost:3003` です。稼働している限り、自分でポート 番号を覚えておく必要はありません。

8.6 ディスク使用量

パネルの一番下には、三つのリアルタイムの合計——コンテナ数、イメージ数、合計サイズ——があり、ターミナルを開かなくても、 コンテナがディスク上でどれだけの容量を占めているかを常に把握 できます。

```
▸ ADVANCED

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

8.7 Advanced

コマンドラインでの等価な操作

```
# パネルが「GET」と呼んでいるもの
$ podman pull ghcr.io/gethomepage/homepage
# カタログのエントリに対してパネルが「RUN」と呼んでいるもの
$ podman run -d -name homepage -p 3003:3000 ghcr.io/gethomepage/homepage
# 稼働中のコンテナを一覧表示—rootless コンテキスト
$ podman ps
# 同じもの、rootful
$ sudo podman ps
# ログ、停止、再起動、削除—アイコン行、手動で
$ podman logs -f homepage
$ podman stop homepage
$ podman restart homepage
$ podman rm homepage
# イメージとディスク使用量
$ podman images
$ podman system df
$ podman image prune
```

ROOTLESS VS. ROOTFUL、その裏側

Rootless の Podman は、あなた自身のユーザーとして、ユーザーの 名前空間の内側でコンテナを実行します——コンテナの「root」は、ホスト上の権限のない UID の範囲にマッピングされるため、コンテナ のプロセスが完全に侵害されたとしても、あなた自身のアカウントの 権限に依然として縛られたままです。Rootful の Podman（`sudo podman ...`）は従来どおりの方式で動作し、コンテナの root はホスト の実際の root にマッピングされます。パネルのスイッチは、実際には 単に普通の `podman` を実行するか `sudo podman` を実行するかを 切り替えているだけです——それぞれ本当に独立した状態ディレクトリ、独立したコンテナリストを持ち、デフォルトでは二つのコンテキスト 間で何も共有されません。

```
# 各コンテキストが自身の状態をどこに保持しているか
$ podman info -format '{{.Store.GraphRoot}}'
# rootless:  /.local/share/containers/storage
# rootful: /var/lib/containers/storage
```

一部のカタログエントリが特定のポートの開放を必要とする理由

コンテナのポートマッピング（homepage の例では `3003:3000`）は、 デフォルトでは、そのサービスを**このマシンからのみ**アクセス可能に します——特に指定しない限り、Podman はそれを `localhost` にバイン ドします。ネットワーク上の別のデバイスからそこにアクセスするに は、さらに Firewall パネル（第 4 章）でそのポートが開かれている 必要があります——この二つのシステムは独立した層であり、ポートが マッピングされた稼働中のコンテナは、Podman がそこで待ち受けて いるという理由だけで、自動的にインターネットや LAN からアクセス 可能になるわけではありません。

CHAPTER 9

Gaming Mode

上部にひとつのメインスイッチ、リアルタイムのステータス行、そして 四つのタブ：**PACKAGES**、**GAMEMODE**、**MANGOHUD**、**PROTON**。

9.1 ビジュアルガイド—メインスイッチ

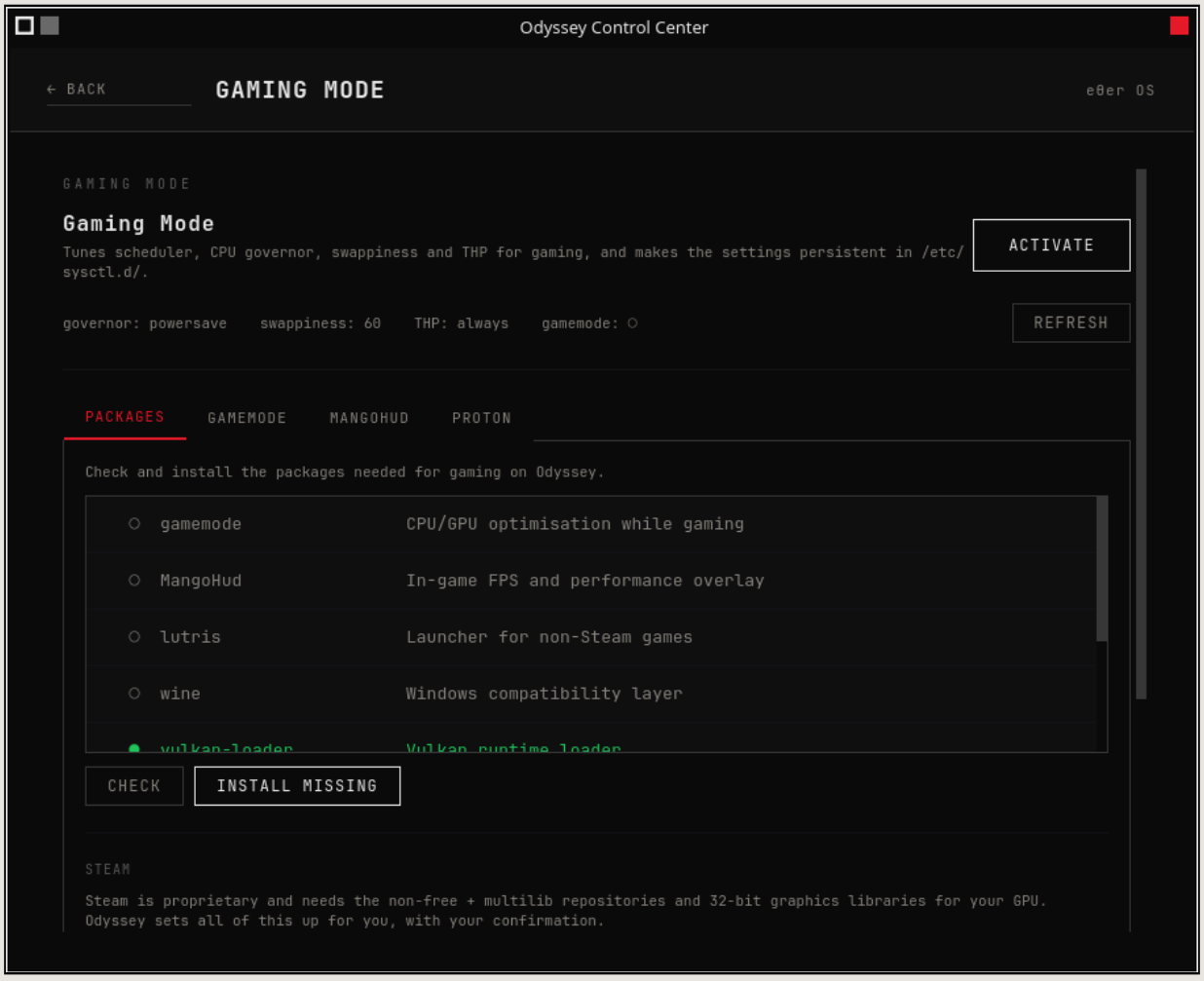


FIG · パネルの上部セクション：「Gaming Mode — Tunes scheduler, CPU governor, swappiness and THP for gaming, and makes the settings persistent in /etc/sysctl.d/」、ACTIVATE ボタン、 REFRESH ボタン、そしてライブなステータス行：governor powersave、swappiness 60、THP always、gamemode off。

このステータス行は、Kernel と Memory パネルが表示するのと同じ ライブな値——governor、swappiness、THP の状態、そして `gamemode` サービスが稼働しているかどうか——を示します。そのため、どちらか のパネルを別々に確認しに行かなくても、有効化が実際に効いたのかを 一目で確認できます。

WHAT	Gaming Mode → ACTIVATE。
WHY	ゲームに適した governor、swappiness、THP の値を手動で 設定するために、わざわざ Kernel と Memory を訪れるのは、専用 のモードが取り除くべき、まさにその種の多段階の作業です。
RESULT	Scheduler、governor、swappiness、THP のすべてが ゲームに適した値に切り替わり、即座に適用されて永続化され ます——ステータス行はその変更を反映して更新されます。
RISK	低く、可逆的です——他の方法で Kernel/Memory で手動で 設定するのと同じ値を、ワンクリックにまとめただけです。

9.2 ビジュアルガイド—Packages と Steam

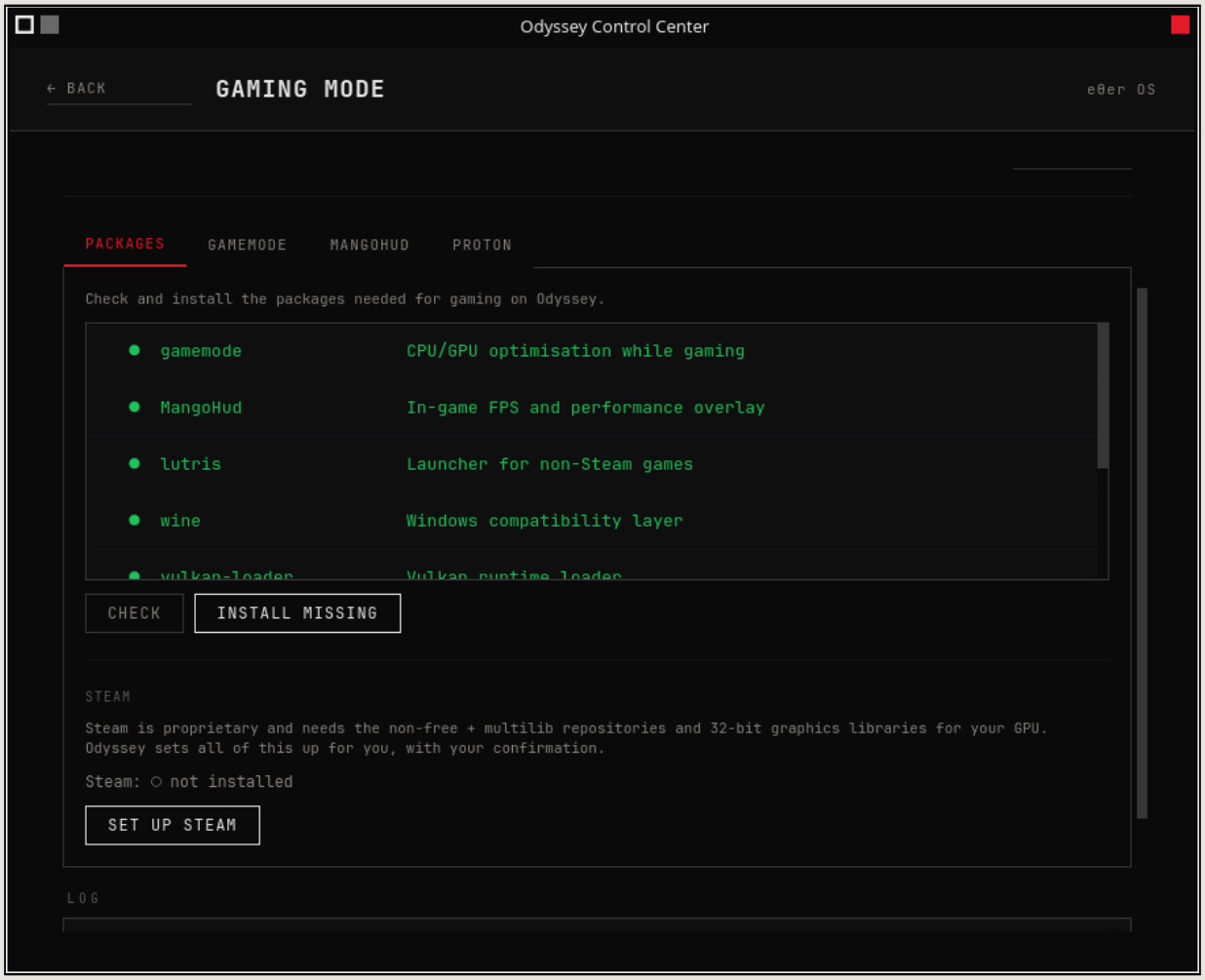


FIG · PACKAGES タブ：gamemode、MangoHud、lutris、wine、 vulkan-loader、すべて塗りつぶされた緑色のドット（インストール 済み）；上部に CHECK と INSTALL MISSING；下の Steam セクション には「Steam: ● installed」と REINSTALL STEAM ボタン。

ゲームに一般的に必要なパッケージのチェックリストです——CHECK は 不足しているものを確認し、INSTALL MISSING はその隙間をひとつの 操作で埋めます。すべてがスクリーンショットのように塗りつぶされた 緑色のドットになれば、完全にセットアップが完了しています。

Steam は、単純なパッケージのインストールだけでは足りないため、 チェックリストの下に独自の専用フローを持っています。

1 **SET UP STEAM**

クリックします——これはまだ何もインス トールしません。最初に説明を伴う確認ダイアログが開きます。

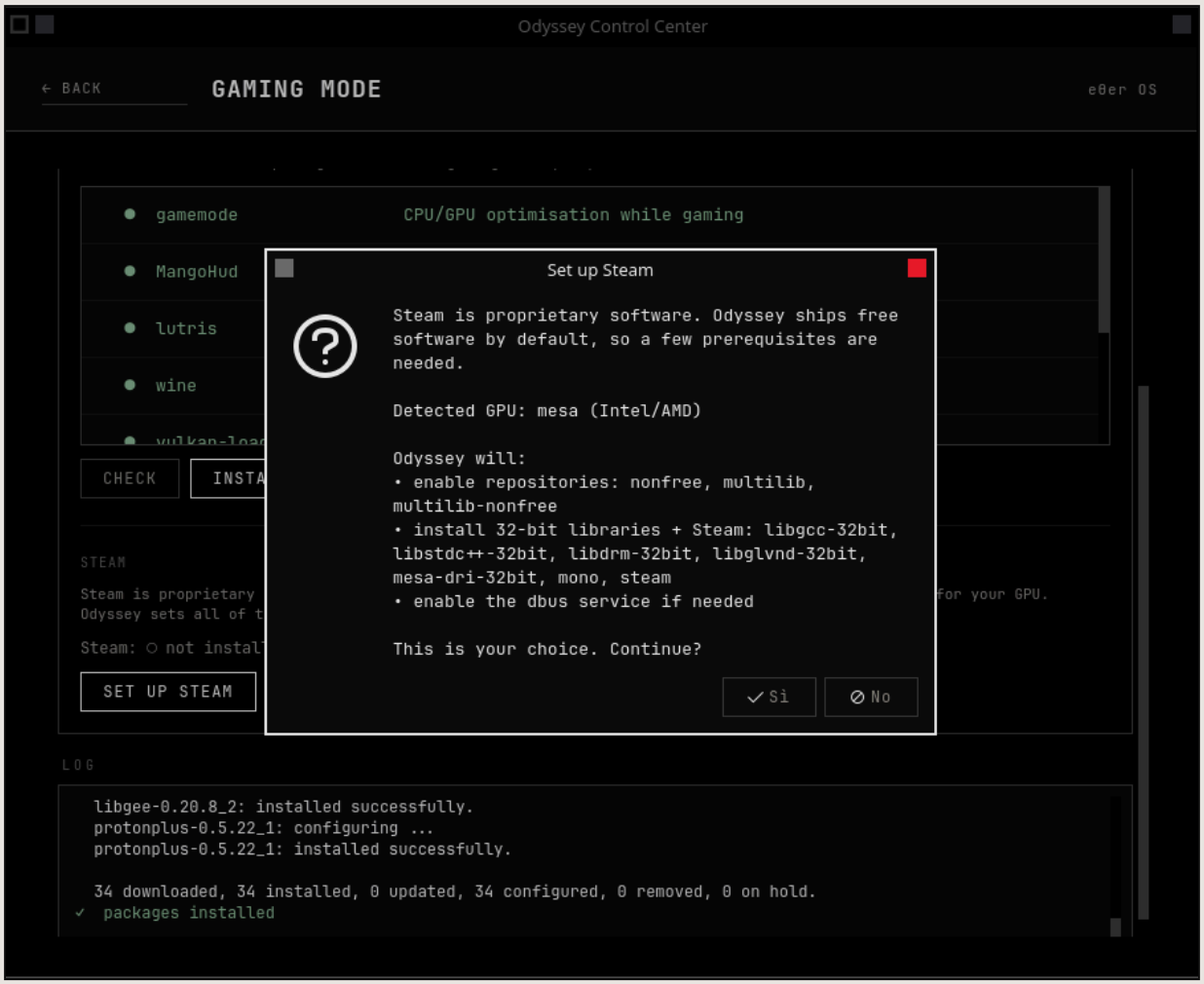


FIG · 確認ダイアログ：「Steam is proprietary software. Odyssey ships free software by default, so a few prerequisites are needed. Detected GPU: mesa (Intel/AMD).」具体的に何が有効化 され、何がインストールされるかの箇条書きリストがあり、最後は 「This is your choice. Continue?」で終わり、Sí / No ボタンが あります。

このダイアログは、あなたが同意する前に、これから起きることを 正確に伝えます：どのリポジトリが有効化されるか（nonfree、multilib、multilib-nonfree）、あなたの**検出された** GPU に 合わせてどの 32 ビットライブラリがインストールされるか、そして まだ有効になっていなければ dbus サービスが有効化されること。確認の背後に隠されているものは何もありません。

2 **Sí**

それを確認します。ログには実際のインストール 作業——パッケージのダウンロード、設定、完了——が送信されます。

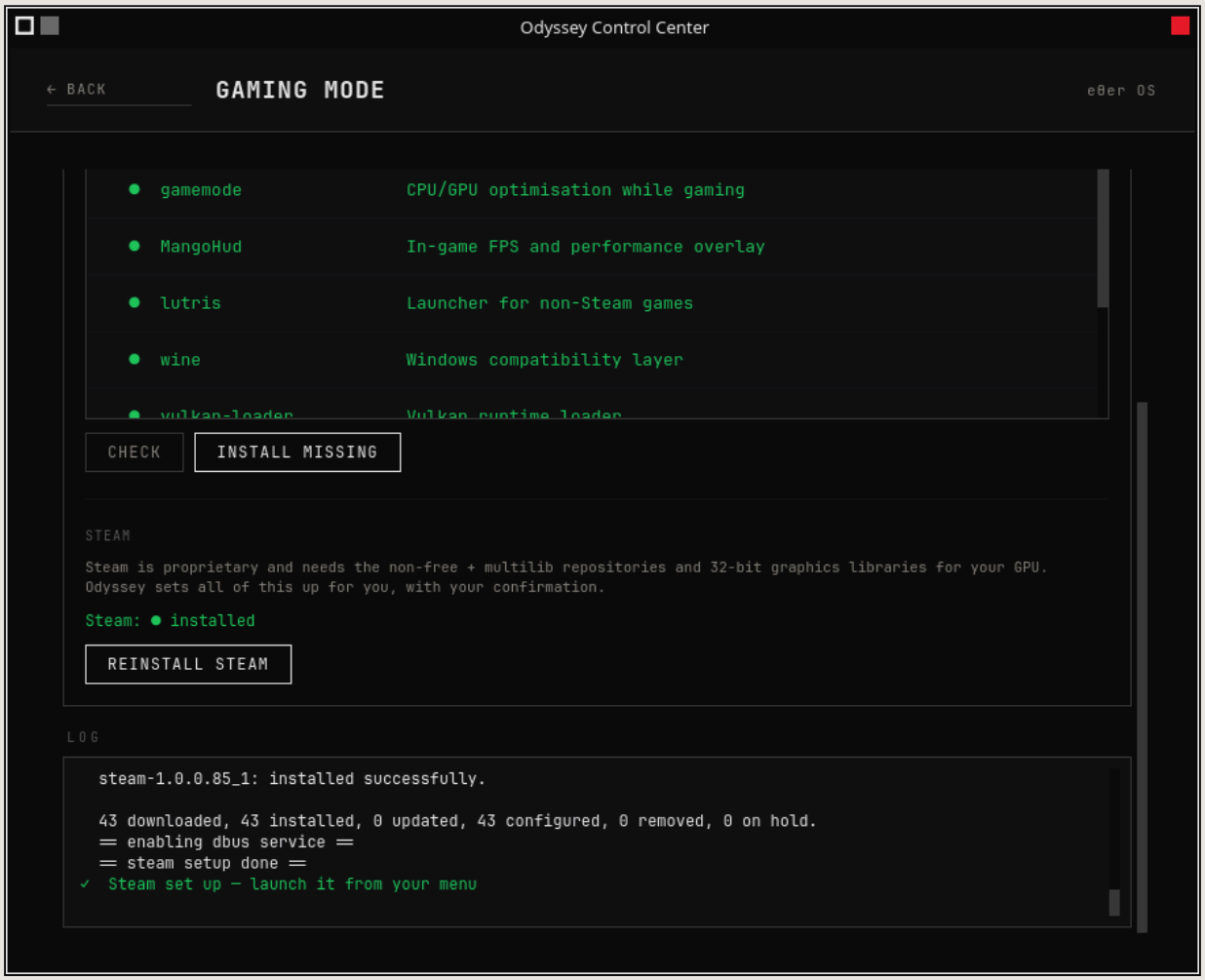


FIG · セットアップ後の PACKAGES タブ：すべてのパッケージが緑色に なり、Steam は REINSTALL STEAM 付きで「● installed」と表示 され、ログの最後は「Steam set up — launch it from your menu.」で終わります。

WHAT	Gaming Mode → PACKAGES タブ → SET UP STEAM → ダイアログ を確認 → Sí。
WHY	Steam はプロプライエタリであり、Void がデフォルトでは 有効化していない non-free/multilib のリポジトリと、GPU に 合わせた 32 ビットライブラリを必要とします——これを手動で 正しく行うことは、Void ベースのシステムを初めて使う人にとって よくあるつまずきのポイントです。
RESULT	必要なリポジトリ、Steam 自体、そしてあなたの具体的な GPU に合った 32 ビットグラフィックライブラリが、すべてインス トールされます——その後、アプリケーションメニューから Steam を起動してください。
RISK	低い——これはリポジトリを有効化してパッケージを追加する だけです。既存のものは何も削除も再設定もされません。ダイア ログは、あなたが確認する前に、何が起こるかの完全なリストを示しています。

9.3 GameMode と MangoHud

GameMode（独自のタブ）は、Feral Interactive によるゲームごとの パフォーマンスデーモンです——それを通じて起動されたゲームは、自分自身のプロセスのためだけに一時的なパフォーマンスの引き上げ（CPU governor、I/O 優先度）を要求し、ゲームが終了すると自動的に 元に戻されます。**MangoHud**（独自のタブ）は、ゲーム中に FPS、 フレームタイム、CPU/GPU 負荷、温度を表示する画面上のオーバー レイです。どち

らのタブも、インストール/確認のコントロールを 提供し、MangoHud についてはさらに、オーバーレイの見た目の クイック設定も提供します。

9.4 Proton

Steam を通じて Windows ゲームを実行するための互換レイヤーの 管理です——何がインストール済みで利用可能かを確認し、Valve が デフォルトでは配布していないバージョンが特定のゲームに必要な 場合、Proton-GE のようなコミュニティ製ビルドを含む、追加の Proton バージョンをインストールします。

```
▸ ADVANCED

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

9.5 Advanced

ACTIVATE が書き込む内容

Kernel と Memory パネルの MAKE PERSISTENT の操作と同じ、根底に あるファイルです——Gaming Mode はひとまとめのパッケージであり、 独立した仕組みではありません：

```
# 永続化された sysctl の値
$ cat /etc/sysctl.d/99-odyssey.conf
# CPU governor——これは sysctl ではなく sysfs ベースであり、この仕組みによって
# 永続化されることはありません。それに依存している場合は、起動後に再適用してください
$ cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_governor
```

SET UP STEAM が実際に実行する内容

確認ダイアログ自体が、ほぼ完全な技術的な開示になっていますが、 具体的には、Mesa（Intel/AMD）システム上では：

```
$ 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 システムでは、mesa-dri-32bit の代わりに追加で nvidia-libs-32bit がダウンロードされます——
# パネルはあなたの GPU ベンダーを検出し、それに応じてパッケージリストを調整します
```

GAMEMODE と MANGOHUD を手動で

```
# GameMode を有効にして任意のゲームを起動する
$ gamemoderun ./game_binary
# または、Steam ゲームの場合、起動オプション：
$ gamemoderun %command%
# MangoHud オーバーレイ、起動オプション：
$ mangohud %command%
# オーバーレイが表示する内容を調整するための、MangoHud 自身の設定ファイル
$ $EDITOR  /.config/MangoHud/MangoHud.conf
```

特定のバージョンの PROTON-GE をインストールする

PROTON タブは `protonplus` または同等のバージョンマネージャーを ラップしています——手動で行う場合の等価な操作は、GitHub の `GloriousEggroll/proton-ge-custom` からリリース版をダウンロード し、Steam の互換ツールディレクトリに展開することです：

```
$ mkdir -p  /.steam/root/compatibilitytools.d
$ tar -xf GE-Proton-VERSION.tar.gz -C  /.steam/root/compatibilitytools.d
# その後、Steam 内でゲームごとに選択します：Properties → Compatibility
```

CHAPTER 10

Telemetry

その名前にもかかわらず、このパネルはあなたが設定する項目ではあり ません——リアルタイムのモニターです。Odyssey はあなたに関する 何も収集せず、何も送信しません。だからここで無効化するべきものは 何もありません。代わりに、このパネルが行うのは、あなた**自身**の システムが行っているすべての発信接続をリアルタイムで見せること です——それによって、「私のマシンは今、誰と話しているのか」は、 何かの主張を信じることではなく、実際に見ることで答えられる問い になります。まさにこの理由から、これは自らを **Reverse Telemetry** と呼んでいます：観察という行為をひっくり返し、あなたに 代わって、それをシステム自身へと向けているのです。

10.1 ビジュアルガイド

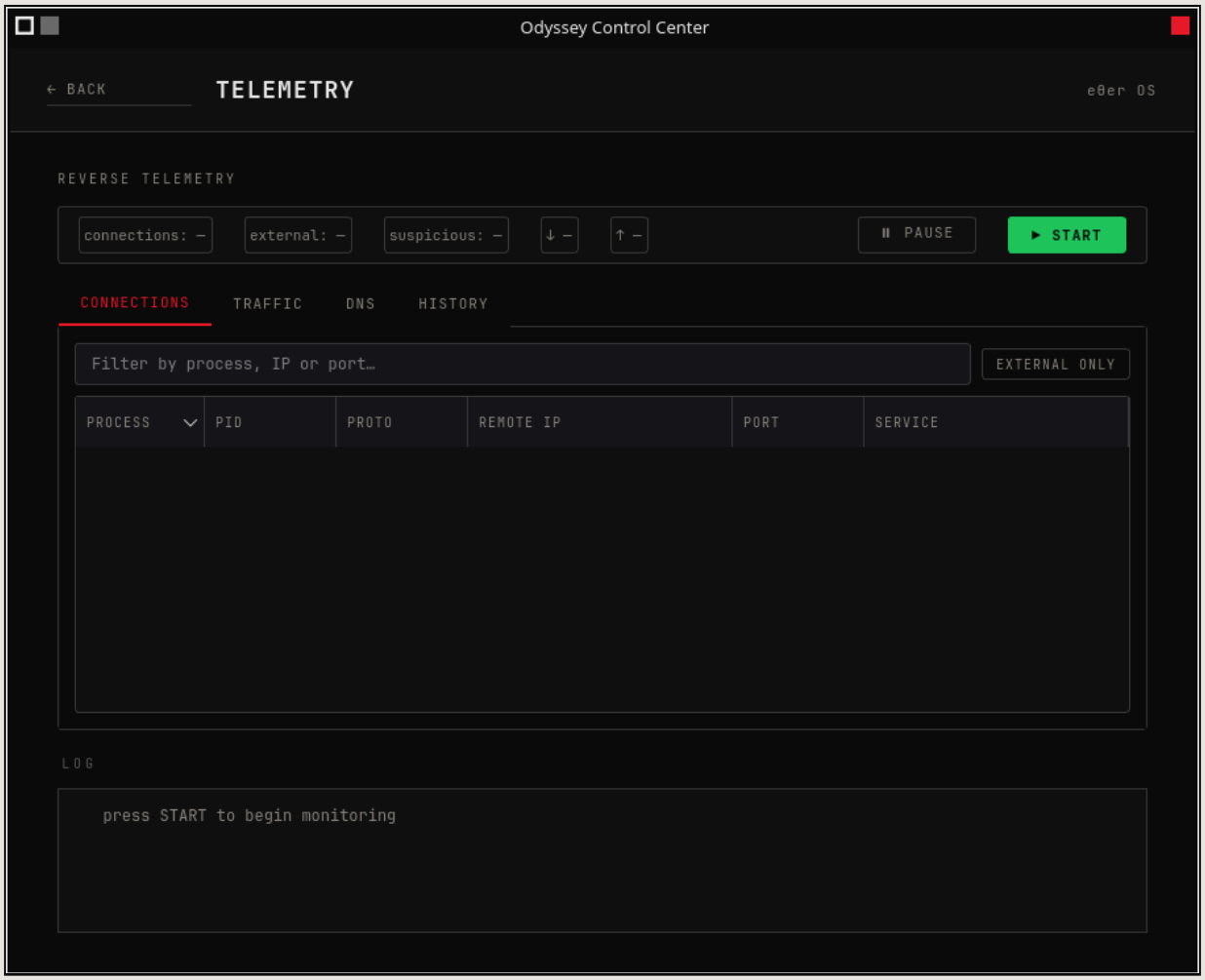


FIG · 開始前：connections/external/suspicious はすべて「–」を 表示し、緑色の START ボタン、そしてログには「press START to begin monitoring」と表示されています。四つのタブ— CONNECTIONS、TRAFFIC、DNS、HISTORY—は、まだ埋まっていない テーブルの上に空のまま残っています。

1 START

リアルタイムのモニタリングを開始します—— ステータスバーとアクティブなタブが即座に埋まり始めます。

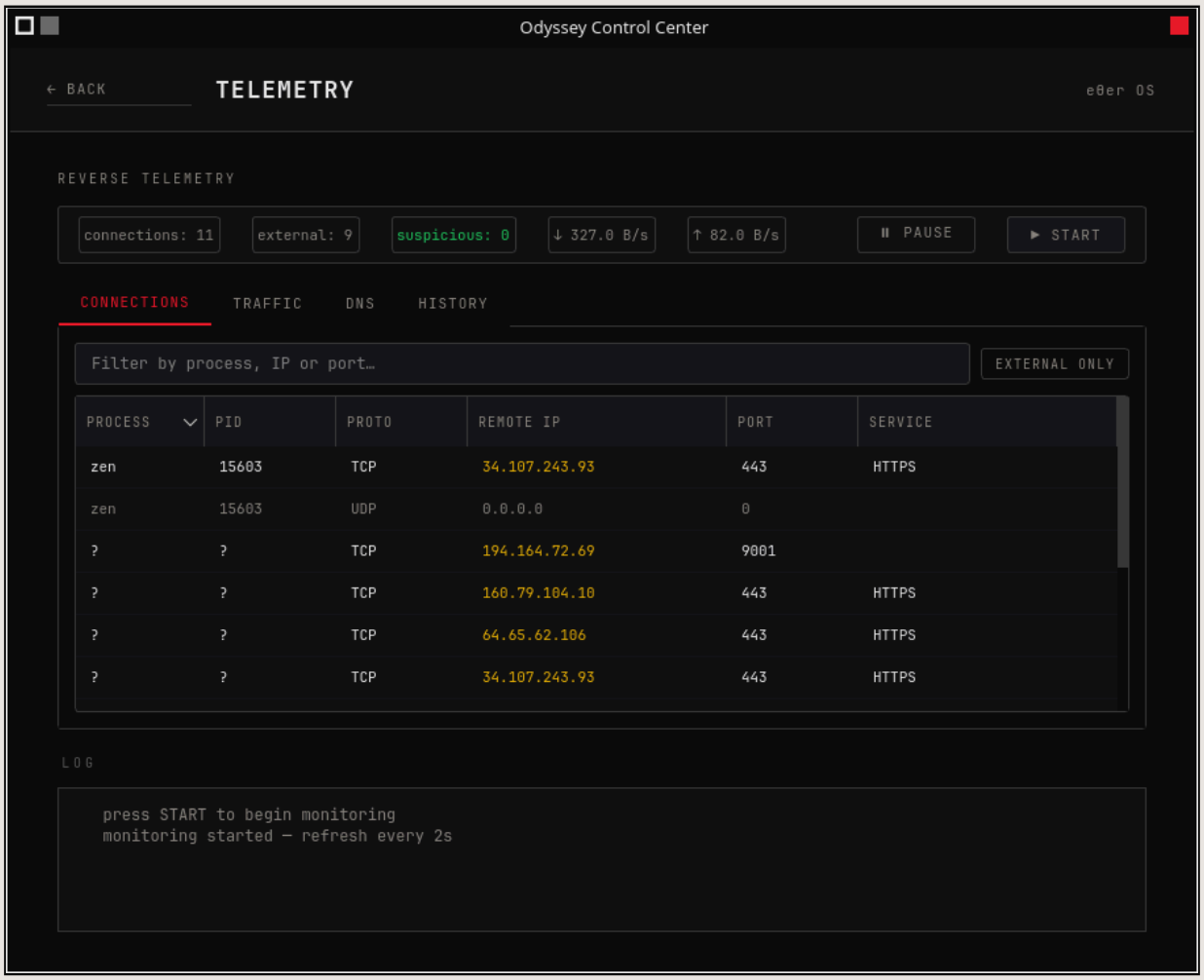


FIG · 開始後：「connections: 11 · external: 9 · suspicious: 0 · ↓327.0 B/s · ↑82.0 B/s」、PAUSE が利用可能になり、CONNECTIONS のテーブルには実際の行が埋まっています：プロセス 「zen」が TCP で 34.107.243.93:443（HTTPS）へ、いくつかの 行はプロセス/PID が「?」とマークされ、他のアドレスへの接続が あります。

上部のステータスバーはリアルタイムの合計値を提供します——接続数、 外部（ローカルとの対比で）、疑わしいもの、現在のダウンロード/ アップロード速度——そして **PAUSE** は取得済みのデータを失うことなく表示を凍結し、**START** はそれを再開します。

四つのタブ

タブ	表示内容
CONNECTIONS	アクティブな接続ごとに一行：プロセス、PID、プロトコル、リモートアドレス、ポート、そのポートに対して知られているサービス（443 → HTTPS、22 → SSH）。任意の列で並べ替え可能です。
TRAFFIC	インターフェースごとのスループット——現在のダウンロード/アップロード速度に加え、モニタリング開始以降の累計。
DNS	ドメイン名解決の、発生したそのままのリアルタイムストリーム——単なる IP よりも多くの場合わかりやすいです。ホスト名は**どこ**だけでなく、**何**であるかを教えてくれるからです。

タブ	表示内容
HISTORY	そのセッションが取得したすべての内容を、スクロール可能なログとして表示します——あなたが見た時点ではもうアクティブな接続ではなくなっている、短命なものを捕まえるのに役立ちます。

プロセス欄が ? を表示している行（スクリーンショットで確認 できます）は、その接続は取得されたものの、その瞬間には PID から 発信元のプロセスを解決できなかったことを意味します——非常に 短命な接続でよく見られます。テーブル上部の**フィルター**フィールド は、プロセス、IP、ポートで絞り込み、**EXTERNAL ONLY** は純粋に ローカルなトラフィックを隠します。

WHAT	Telemetry → START → CONNECTIONS を観察する、あるいは PROCESS や REMOTE IP で並べ替えて何か具体的なものを調査する。
WHY	これは「このマシン上に、ネットワークに接続すべきでない 何かがアクセスしていないか」という問いへの、直接的で具体的な 答えです——ある供給元が自社のソフトウェアについて主張する 内容ではなく、カーネル自身の接続テーブルから直接読み取られた ものです。
RESULT	稼働中で一時停止していない限り継続的に更新される、 すべての発信接続のライブで正確な状況。
RISK	なし——完全に読み取り専用です。

自らネットワークにアクセスする通常の理由を持たないプロセス—— むき出しのシェル、単独で実行されているスクリプトインタープリ タ——からの接続は、ステータスバーの中で自動的に「suspicious」と マークされます。それによって、ブラウザやシステムサービスの通常の トラフィックの中に埋もれてしまうのではなく、異常が目立つように なります。

```
▸ ADVANCED

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

10.2 Advanced

データはどこから来るのか

このパネルは、どんな標準的なツールでも読み取るであろう、同じ カーネルレベルの接続テーブルを読み取ります。モニタリングが アクティブな間、短い間隔で更新されますが、そのための専用の特別な ドライバーやカーネルモジュールは一切ありません。コマンドライン での等価な操作：

```
# 開いているすべてのソケットと、その所有プロセス—CONNECTIONS と同じフィールド
$ sudo ss -tupn
# インターフェースごとのスループット—TRAFFIC と同じ数値
$ ip -s link
# パネルの外でそれを見たい場合の、一度限りのライブ表示
$ sudo nethogs
```

DNS 解決を手動で確認する

```
# 発信 DNS クエリのライブキャプチャ（パケットキャプチャツールが必要です）
$ sudo tcpdump -i any -n port 53
```

一部の行で PROCESS が「?」を示す理由

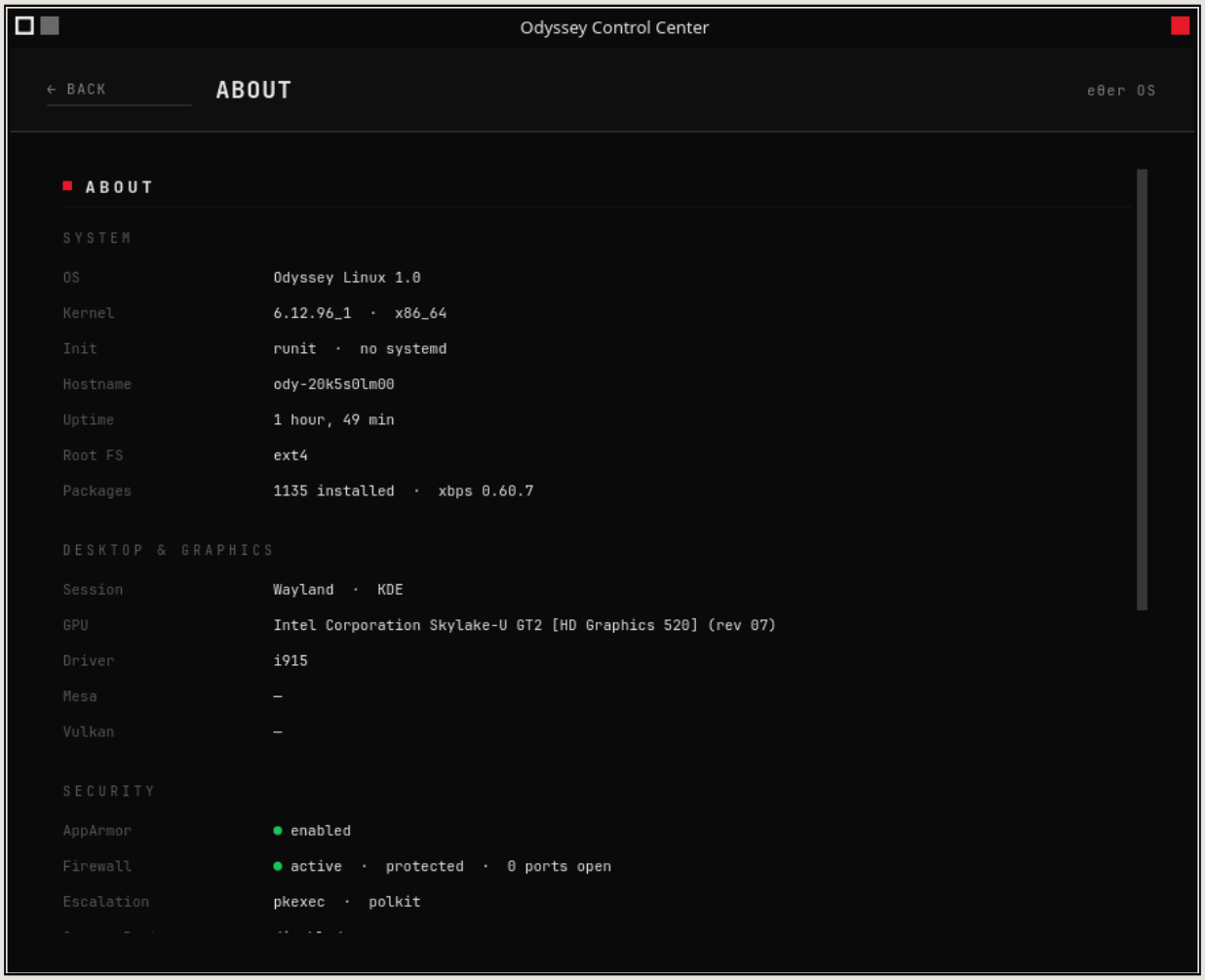
ライブな接続をその所有プロセスに一致させるには、取得の瞬間に 候補となるすべてのプロセスの `/proc/PID/fd` を読み取る必要が あります——接続がパネルの更新間隔よりも速く開いて閉じてしまう 場合、あるいは所有プロセスがすでに終了している場合、パネルには その一致を行うための時間の窓がなく、推測する代わりに `?` を表示 します。これはタイミング上の制約であり、何かが隠されている兆候 ではありません。

CHAPTER 11

About

このマシンに関する、ただひとつの、網羅的な診断シートで、タブを 開くたびに更新されるため、Control Center を開き直さなくても、 他のパネルで変わったことを常に反映しています。

11.1 ビジュアルガイド



FIG・一番上までスクロールしたパネル：SYSTEM セクションには 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」；下には DESKTOP & GRAPHICS があり、Session、GPU、Driver、Mesa、Vulkan； SECURITY は AppArmor「● enabled」と Firewall「● active · protected · 0 ports open」を表示し始めています。

順番に三つのセクション：**SYSTEM**（カーネル、init、ホスト名、 稼働時間、ファイルシステム、パッケージ数）、**DESKTOP & GRAPHICS**（セッションの種類、GPU、ドライバー、Mesa/Vulkan の サポート状況）、そしてさらに下に **SECURITY**（AppArmor と Firewall の状態を、単なるテキストではなく色分けされたインジ ケーターで——有効/アクティブなら緑）と **HARDWARE**（マザーボード、 オーディオ）。

WHAT	About → 三つのセクションをスクロールして、任意の値を 確認する——カーネルのバージョン、使用中のドライバー、 AppArmor とファイアウォールが本当にアクティブかどうか。
WHY	五つの異なるパネルを開かなくても、一箇所でシステム全体の 状態を確認する最も速い方法です。
RESULT	タブを訪れるたびに更新される、完全で最新のスナップ ショット。過去のセッションの古い情報が表示されることは決して ありません。
RISK	なし——完全に読み取り専用です。

11.2 Copy report

パネル上部にひとつだけボタンがあります。**COPY REPORT** は、 シート全体をプレーンテキストとしてクリップボードに置きます—— すべてのセクション、すべての値——フォーラムの投稿や bug tracker のチケットに直接貼り付ける準備が整った状態で。

WHAT	About → COPY REPORT → バグレポートやフォーラムの投稿に 貼り付ける。
WHY	バグレポートは、添付されたシステムの詳細情報と同じ程度に しか役に立ちません。カーネルのバージョン、ドライバー、AppArmor の状態、その他十数個のフィールドを手動で入力することは、まさに多くの人がこのステップを省略してしまう摩擦の 原因です。
RESULT	完全で、正しくフォーマットされたシステムレポートが クリップボードにあり、貼り付ける準備ができています——本来 メンテナーがフィールドごとにあなたに尋ねなければならなかった はずの、まさにその情報です。
RISK	あなたのシステムに対するリスクはありません。このレポート は診断情報（バージョン、状態、ハードウェアのモデル）であり、個人ファイルや認証情報は含まれません——ただし、どんな診断 データの出力とも同様に、それが何を含んでいるか確信が持てない 場合は、どこか公開の場に投稿する前に読み直してください。

▸ ADVANCED

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

11.3 Advanced

各フィールドがどこから来るのか

このパネルの中には、インストール時点から作り話をされたものや キャッシュされたものは何ひとつありません——タブが開かれるたびに、 すべての値がライブに問い合わせられます：

```
$ 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
```

具体的に Security セクションでは：

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

まず ABOUT を確認する価値がある理由

訪れるたびにシステムを再度問い合わせるため、About は、他のパネル で行った変更が本当に効果を発揮したかを確認する最も速い方法です ——Firewall であるモードを適用したら、Firewall 自体を再度開く のではなく、About の Security セクションを確認してください。 これは、Control Center の他の部分全体に対して、「これは本当に うまくいったのか？」というクロスチェックとして機能するように 設計されています。

CHAPTER 12

NVIDIA driver

> **統合待ち**
>
> この章はプレースホルダーです。NVIDIA モジュールは、NVIDIA ハードウェアが検出されたマシン上でのみ Control Center に表示 され、そのスクリーンショットもそうしたマシン上でしか取得でき ません——それらは別途提供され、この章は本ガイドの将来の改訂版 で完成する予定です。この章が意図的に About の後、最後に配置 されているのは、後から追加しても、他のどの章のページ番号や セクション参照もずれないようにするためです。

すでにわかっていて、完成した章の土台となる内容：このパネルには 目立つ **BETA** バッジが付いており、冒頭には、ドライバー自体が プロプライエタリであるという注記があります——Void によって NVIDIA の公式インストーラーから、固定されたチェックサム付きで パッケージ化されていますが、独立した監査は行えません。これは、 Odyssey の他の部分が自分自身に対して課しているのと同じ誠実さ です。パネルは、あなたの具体的な GPU を内蔵データベースと照合 することで検出し、正しいドライバーブランチを自動的に選択します（current/Turing 以降、あるいは古いカードのための三つのレガシー ブランチのいずれか。プロプライエタリなものが何も適用できない 場合は、オープンソースの `nouveau` ドライバーにフォールバック します）。インストールは、可視の進行状況インジケーター付きの、七段階の追跡可能なパイプラインとして実行され、パネルは、あなた がインストールをクリックする前に、あらかじめ自身の復旧コマンド ——テキストコンソールからの `odyssey-nvidia --revert`——を表示 します。ハイブリッドのラップトップでは、PRIME render-offload のセクションが自動的に現れ、バッテリーの持続時間のために他の すべてが内蔵 GPU に留まる一方で、特定のゲームやプログラムだけを NVIDIA GPU 上で実行できるようになります。