# Essential Eight on Linux, Part 1 of 8: Application Control on Ubuntu 26.04 LTS

April 26, 2026 / Gavin Jackson

essential-eight asd ism ubuntu ubuntu-pro landscape apparmor linux security

Application control is one of the harder Essential Eight mitigations to translate cleanly into Linux terms.

On Windows, most people immediately think of AppLocker or WDAC. On Ubuntu 26.04 LTS, the control exists, but you build it differently. The reference implementation is less about one giant allowlisting product and more about combining package trust, confinement, service hardening, and tight control over where code can execute.

For Ubuntu fleets, that means **AppArmor**, **snap confinement**, **signed APT repositories**, **restricted administrative access**, and **Landscape** to keep the configuration consistent.

## What ASD is trying to achieve

The Essential Eight intent is straightforward: only approved code should run, especially on workstations and internet-facing systems.

For Linux, that usually breaks into four practical questions:

1. Where can software come from?
2. Which users can install or execute it?
3. What can the application touch once it starts?
4. How do you prove the policy is being enforced at scale?

## Ubuntu 26.04 LTS reference implementation

### Resolute Raccoon highlights

Ubuntu 26.04 LTS "Resolute Raccoon" improves the desktop permission story in ways that help this mitigation:

- improved application permission prompting through the Security Center
- better snap and XDG Desktop Portal permission handling
- a more unified app management experience with better Debian package support

Those changes do not replace AppArmor or repository control, but they do make user-space permission boundaries more visible and manageable than they were on Ubuntu 24.04 LTS.

### 1. Treat signed repositories as the first allowlist

If a package is not coming from:

- Ubuntu official repositories
- an approved internal mirror
- Canonical-delivered snaps
- a tightly governed third-party repository

then it should be considered untrusted by default.

That sounds basic, but it is the foundation of application control on Linux. APT already gives you package signing and repository trust. The security work is making sure engineers do not bypass it with random shell installers, curl-pipes, or ad hoc binaries in home directories.

### 2. Use AppArmor as the main execution boundary

Ubuntu 26.04 ships with AppArmor enabled, and it is the most important native application control primitive in the platform.

AppArmor does not behave like a classic hash-based allowlisting tool. Instead, it confines applications with profiles that define what files, sockets, capabilities, mounts, and execution transitions are permitted. In practice that means:

- browsers can be constrained
- network-facing daemons can be narrowed
- admin tools can be profiled more aggressively
- custom in-house applications can be forced into known-good access paths

For higher-risk workloads, use:

- `apparmor-profiles`
- `apparmor-profiles-extra`
- custom local profiles under `/etc/apparmor.d/`

If you are running internally developed software, writing and maintaining an AppArmor profile for that software is one of the best security returns you can get on Ubuntu.

### 3. Prefer confined packaging formats where they fit

Ubuntu's default Firefox delivery model is a good example here. Firefox is delivered as a **snap**, which means it runs with confinement and a cleaner update path than a random manually installed browser tarball.

Snaps are not a full replacement for application control, but they do help by:

- reducing arbitrary install paths
- centralising update channels
- applying confinement around the application

- making drift more obvious

For desktop fleets, preferring snap or repo-managed applications over hand-installed binaries is a very practical control.

### 4. Remove common Linux execution escape hatches

If you want application control on Linux to be real, you need to close the usual "but it still runs from here" loopholes:

- mount `/tmp` and `/var/tmp` with `noexec` where application compatibility allows
- use `nodev` and `nosuid` on temporary and user-writable filesystems
- restrict interpreters on systems that do not need them
- disable unneeded compiler toolchains on user workstations
- prevent users from writing executables into globally reachable paths
- keep `sudo` tightly scoped so package installation is not broadly available

This is not glamorous, but it is how Linux application control becomes operational instead of theoretical.

### 5. Use Landscape to enforce consistency

Landscape is not an allowlisting engine by itself, but it is useful for the control plane around application control:

- package inventory
- repository governance
- script execution under approval
- profile-based rollout
- compliance visibility across fleets

If AppArmor is the technical enforcement layer, Landscape is the operational layer that helps stop exceptions from multiplying.

## ISM control mapping

The October 2024 Essential Eight to ISM mapping ties this mitigation to the following controls:

| ISM control | Linux implementation on Ubuntu 26.04 LTS |
| --- | --- |
| `ISM-0843` | Use an application control capability for workstations via AppArmor, package trust, and constrained install paths. |
| `ISM-1490` | Extend application control to internet-facing servers with mandatory AppArmor profiles and minimal package sets. |
| `ISM-1870` | Ensure only approved executables, scripts, and installers are allowed on workstations through repository policy and privilege restrictions. |
| `ISM-1871` | Apply the same approved-software model to internet-facing servers, especially reverse proxies, web stacks, and bastions. |
| `ISM-1657` | Validate and maintain application control rulesets, including approved package sources, snap channels, and AppArmor policies. |
| `ISM-1544` | Constrain execution of application control bypass mechanisms by limiting administrative changes and interpreter access. |
| `ISM-1582` | Prevent execution from user-writable locations where possible with `noexec`, packaging standards, and profile enforcement. |
| `ISM-1660` | Review and update application control configurations as the environment changes, especially after new software onboarding. |

## Where Linux does not map neatly

Ubuntu still does not give you a perfect built-in equivalent to a centrally managed, deny-by-default enterprise allowlisting platform for every binary on every endpoint.

That is the real gap.

AppArmor is excellent at confinement, but it is not the same thing as a mature enterprise product that can answer, at a glance, "show me every unapproved executable seen in the fleet this week."

## Commercial alternatives

If you need a more opinionated commercial control on Linux, these are worth evaluating:

- **Airlock Digital** for cross-platform allowlisting and central policy management
- **Canonical Ubuntu Pro + Landscape** where the real problem is package governance and fleet consistency more than binary reputation
- **Teleport** as a supporting control for privileged access to managed Linux targets, so users cannot casually introduce software under standing admin rights

Teleport is not an application control product, but it materially reduces the number of people who can bypass one.

## Practical mitigations when full allowlisting is not possible

If you cannot deploy a commercial Linux allowlisting platform, I would still aim for this stack:

- Ubuntu Pro and Landscape for package governance
- AppArmor everywhere, with custom profiles for business-critical apps
- snap confinement where sensible
- `noexec,nodev,nosuid` on temporary and user-writable mounts
- no local admin for standard users
- package installation only from approved signed repositories
- audit logging for package changes, `sudo`, and profile violations

That combination is not as neat as WDAC, but on Ubuntu 26.04 it is a credible and defensible implementation of the Essential Eight intent.

## The bottom line

Application control on Linux is absolutely possible, but it looks different.

On Ubuntu 26.04 LTS, the strongest pattern is to treat **repository trust as the software allowlist**, **AppArmor as the execution boundary**, and **Landscape as the fleet governance layer**. Resolute Raccoon's improved permission prompting and snap portal controls make the desktop side of that story a bit stronger as well.

## References


- ASD Essential Eight maturity model and ISM mapping (October 2024)
- Ubuntu security overview
- AppArmor on Ubuntu
- Landscape documentation
- Ubuntu Pro



Downloaded from https://www.gavinj.net/post/essential-eight-linux-application-control
Generated November 8, 2026. Copyright Gavin Jackson. All rights reserved.