Essential Eight on Linux, Part 7 of 8: User Application Hardening on Ubuntu 26.04 LTS

May 4, 2026 / Gavin Jackson

essential-eight asd ism ubuntu linux firefox apparmor browser-isolation security

Of all the Essential Eight mitigations, this one is probably the most naturally adaptable to Linux.

The reason is simple: the underlying security idea is not Windows-specific at all. Harden the software users attack with first. That is just as relevant on Ubuntu 26.04 LTS as anywhere else.

Browsers, PDF readers, email clients, collaboration tools, and document handlers are all part of the frontline.

What ASD is trying to achieve

The Essential Eight user application hardening mitigation is about reducing unnecessary functionality in common user-facing software so attackers have fewer ways to execute code, pivot, or deliver payloads.

On Linux, that mostly becomes:

- browser hardening
- document viewer hardening
- script and plugin reduction
- stronger defaults for content handling
- isolation for risky applications

Ubuntu 26.04 LTS reference implementation

Resolute Raccoon highlights

Ubuntu 26.04 LTS is actually a nice step forward for this mitigation:

- GNOME 50 on Wayland reduces some legacy X11 desktop attack patterns, especially around global input and screen access
- application permission prompting is improved, with early integration into the **Security Center**

None of that replaces hardening policy, but it gives you a cleaner base desktop than Ubuntu 24.04 LTS did.

1. Start with the browser

Ubuntu 26.04 ships Firefox as a snap, which is already a helpful default because you get:

- confinement

- central update channels
- a cleaner packaging model

But that is only the start. For managed environments, use enterprise browser policies to enforce settings such as:

- extension allowlists
- password manager decisions
- telemetry or privacy settings consistent with policy
- download restrictions
- homepage and safe browsing controls

For Chromium-based browsers on Linux, the same principle applies. The platform is not the hard bit. The discipline is.

How to implement enterprise browser policies on Ubuntu

This is the part that often gets hand-waved away.

On Linux, the practical pattern is usually:

1. define your browser standards in JSON
2. deploy them with your configuration management tool
3. verify enforcement on endpoints
4. keep the package source for the browser under central control

For **Firefox**, Mozilla supports cross-platform policy management through `policies.json`.

On Linux, the usual system-wide location is:

```
/etc/firefox/policies/policies.json
```

On Ubuntu, this path also works for the Firefox snap. Make the file root-owned and writable only by administrators; otherwise, you have a preference file rather than an enterprise control.

A minimal example might look like this:

```
{
 "policies": {
 "DisableFirefoxStudies": true,
 "DisablePocket": true,
 "DisableTelemetry": true,
 "BlockAboutConfig": true,
 "ExtensionSettings": {
 "*": {
 "installation_mode": "blocked"
 },
 "uBlock0@raymondhill.net": {
 "installation_mode": "force_installed",
 "install_url": "https://addons.mozilla.org/firefox/downloads/latest/ublock-origin/latest.xpi"
 }
 }
 }
}
```

That gives you a simple Linux-native way to:

- block unapproved extensions
- force-install approved ones
- lock down settings standard users should not change
- push a consistent browser posture across an Ubuntu fleet

For **Google Chrome on Linux**, Google documents two policy tiers:

- **managed** policies that users cannot override
- **recommended** policies that set defaults users may change

Typical paths are:

```
/etc/opt/chrome/policies/managed/
/etc/opt/chrome/policies/recommended/
```

For Chromium-based builds managed through distro packaging, the equivalent path may instead be under:

```
/etc/chromium/policies/
```

A simple managed policy file could look like:

```
{
 "BrowserSignin": 0,
 "PasswordManagerEnabled": false,
 "SafeBrowsingProtectionLevel": 2,
 "ExtensionInstallBlocklist": ["*"],
 "ExtensionInstallAllowlist": [
 "ddkjiahejlhfcafbddmgiahcphecmpfh"
 ],
 "ExtensionInstallForcelist": [
 "ddkjiahejlhfcafbddmgiahcphecmpfh;https://clients2.google.com/service/update2/crx"
 ]
}
```

That example blocks user-installed extensions by default and force-installs **uBlock Origin Lite**, which is the Manifest V3-friendly version for Chrome. If your organisation uses Chrome Enterprise Core or a commercial secure web gateway, you may prefer to manage extension policy there instead of hard-coding extension IDs directly into local JSON.

At fleet scale, I would not hand-place these files. I would push them with:

- **Landscape** for script and package driven enforcement
- **Ansible**, **Puppet**, or **Salt** if that is already your Linux config baseline
- an immutable desktop image pipeline if the workstations are provisioned from a gold image

To validate the result:

- in Firefox, check `about:policies`
- in Chrome, check `chrome://policy`

That verification step matters. A policy file that exists on disk but is ignored by the packaged browser is not a control.

2. Remove dead features and dangerous defaults

A lot of historical browser attack surface simply should not exist anymore:

- Java browser plugins or Java Web Start/JNLP helpers unless explicitly required
- unmanaged external protocol handlers
- unmanaged extension sprawl
- automatic file handling for risky content

On Ubuntu, this is partly a browser policy problem and partly a software standardisation problem. The practical control is to define what handlers, extensions, download behaviours, and helper applications are allowed, then test those settings through `about:policies` or `chrome://policy`.

3. Treat ad, script, and active-content control as part of hardening

The ASD guidance includes blocking web advertisements and unnecessary active content for good reason.

For Ubuntu desktops, that can mean:

- enterprise-managed content blocking in the browser
- DNS or secure web gateway filtering
- remote browser isolation for higher-risk use cases

You do not have to make the desktop miserable. But you should absolutely reduce the number of hostile scripts users are expected to process by default.

For many Ubuntu fleets, a sensible pattern is:

- one approved ad or content-blocking extension, centrally installed
- browser-side pop-up and download restrictions
- DNS or SWG filtering for known malicious domains
- a separate isolated path for genuinely risky browsing instead of trying to make every endpoint absorb that risk

4. Harden document and PDF handling

Browsers are not the only target. PDF readers, office suites, and mail clients should also be part of the design.

On Ubuntu:

- prefer simple viewers over feature-heavy document tooling where possible
- keep document handlers patched
- confine risky apps with AppArmor
- open especially risky content in isolated sessions rather than directly on the endpoint

5. Use isolation where hardening is not enough

Some use cases simply carry more risk:

- research on the open internet
- handling untrusted attachments
- accessing third-party portals

That is where isolation tools are more honest than trying to harden a general-purpose desktop into something it is not. Browser isolation, remote workspace patterns, or disposable sessions can all help here.

Kasm as a commercial isolation pattern

One product worth calling out explicitly here is **Kasm Workspaces**.

Kasm is useful because it changes the location where the risky web code executes. Instead of the user's Ubuntu endpoint processing hostile websites directly, the browsing session runs in a **remote, containerised browser** and is streamed back to the user. The endpoint is interacting with the rendered session, not running the full browsing workload locally.

That makes Kasm a very practical commercial answer when your security objective is not just "harden the browser a bit more" but "move high-risk web activity off the endpoint altogether."

From an Essential Eight perspective, Kasm fits especially well as an uplift control for:

- `ISM-1485` around reducing exposure to web advertisements and active content
- `ISM-1412` around restricting risky browser processing paths
- `ISM-1585` where browser security settings need to be enforced rather than left to users

It is not a direct replacement for every Windows-specific Office or PDF hardening control, but it is a very practical way to move high-risk web activity out of the standard endpoint session.

Kasm is especially compelling for Linux-heavy shops because it is not Windows-dependent and does not require you to pretend every risky workflow should be solved on the endpoint itself.

Good Kasm use cases on an Ubuntu fleet

Here are a few situations where Kasm makes a lot of sense:

- **Internet research from secure or sensitive environments**: analysts, engineers, or administrators can browse the open internet without giving the local workstation direct exposure to the full web threat surface.
- **Opening suspicious links from email or chat**: users can push high-risk links into an isolated browser instead of handling them in their standard desktop session.
- **Third-party or BYOD access**: contractors can use a browser-delivered isolated workspace without getting broad direct access from their local machine into your environment.
- **Cross-domain or regulated browsing**: teams working in government, defence, legal, or finance can enforce tighter clipboard, upload, download, and session controls around browser activity.
- **OSINT and investigative work**: Kasm is also useful when you want disposable browsing sessions, alternate egress points, and less attribution back to the analyst workstation.

A practical Kasm pattern

The strongest pattern is usually not "everyone browses everything through Kasm all the time."

It is more often:

- standard browser hardening on all Ubuntu endpoints
- Kasm for high-risk browsing roles
- Kasm for suspicious links or unknown websites
- policy controls on uploads, downloads, clipboard, and printing in the isolated browser
- SSO and MFA in front of the Kasm environment

That gives you a good balance between productivity and containment.

> ***Logs or it didn't happen***

> *There does not appear to be an official Ubuntu 26.04 support listing for Elasticsearch yet, so I would not present it as a certified 26.04 reference build today. That said, Elastic remains one of the top-tier distributed logging and search options in the Linux ecosystem, and it is absolutely worth considering for centralising Ubuntu system and application logs.*
>
> *The classic Elastic Stack is made up of four main pieces: **Elasticsearch** for storage and search, **Logstash** for ingest and enrichment, **Kibana** for dashboards and investigation, and **Beats** such as Filebeat for shipping host and application logs.*
>
> *In an Essential Eight context, the important point is not "install Elastic and declare victory." It is having protected, searchable evidence that browser policy, AppArmor, audit, application, and endpoint events are being collected and reviewed. I will come back to this properly in a future deep-dive on Elasticsearch fundamentals.*

ISM control mapping

The October 2024 Essential Eight to ISM mapping is very Windows-heavy in this area, so the honest Linux answer is a mix of direct controls, compensating controls, and clearly documented non-applicability.

ISM control	E8 requirement in plain English	Ubuntu 26.04 implementation
`ISM-1654`	Internet Explorer 11 is disabled or removed.	Not applicable on Ubuntu by default. Document it as N/A and avoid reintroducing IE through Wine or remote application publishing unless it is isolated and explicitly approved.
`ISM-1486`	Web browsers do not process Java from the internet.	Do not install Java browser plugins or Java Web Start handlers. If Java is needed for internal tools, restrict it to approved internal origins and verify browser/helper handling.
`ISM-1485`	Web browsers do not process web advertisements from the internet.	Use an enterprise-managed content blocker, secure web gateway, DNS filtering, or remote browser isolation for higher-risk roles.
`ISM-1412`	Browsers are hardened using ASD and vendor guidance.	Enforce Firefox `policies.json` or Chrome managed policies, then verify with `about:policies` or `chrome://policy`.
`ISM-1585`	Browser security settings cannot be changed by users.	Store browser policy files as root-owned configuration deployed by Landscape, Ansible, Puppet, Salt, or a managed image pipeline.
`ISM-1667`	Microsoft Office is blocked from creating child processes.	No neat native equivalent exists for LibreOffice. Use AppArmor or isolated sessions for high-risk document handling, and use managed remote Windows Office if exact Microsoft Office control coverage is required.
`ISM-1668`	Microsoft Office is blocked from creating executable content.	Disable office macros unless required, avoid trusted write-execute locations, and treat document-created scripts or binaries as blocked content in endpoint monitoring.
`ISM-1669`	Microsoft Office is blocked from injecting code into other processes.	Mostly Windows-specific. For Linux office suites, reduce exposure with AppArmor confinement and isolation for untrusted documents.
`ISM-1542`	Microsoft Office is configured to prevent activation of OLE packages.	Mostly Windows-specific. Avoid Wine-based Office for standard users, disable external object activation where supported, and open untrusted documents in isolated sessions.
`ISM-1859`	Office productivity suites are hardened using ASD and vendor guidance.	Baseline LibreOffice, OnlyOffice, or browser-based office tooling through managed configuration, and record any settings that cannot be centrally locked.
`ISM-1823`	Office productivity suite security settings cannot be changed by users.	Use root-owned configuration, a managed desktop image, or isolation. If the chosen office suite cannot enforce a setting centrally, treat that as a control gap.

ISM control	E8 requirement in plain English	Ubuntu 26.04 implementation
`ISM-1670`	PDF software is blocked from creating child processes.	Prefer simple PDF viewers, keep PDF handling patched, and use AppArmor or browser isolation for untrusted PDF workflows.
`ISM-1860`	PDF software is hardened using ASD and vendor guidance.	Disable PDF JavaScript or active features where the reader supports it, and avoid feature-heavy readers unless there is a business need.
`ISM-1824`	PDF software security settings cannot be changed by users.	Choose PDF tooling that can be centrally configured, or enforce the workflow through managed images and isolated sessions.
`ISM-1655`	.NET Framework 3.5 is disabled or removed.	Windows-specific. On Ubuntu, avoid unsupported legacy .NET or Wine runtime stacks unless they are isolated and business-owned.
`ISM-1621`	Windows PowerShell 2.0 is disabled or removed.	Not installed on Ubuntu by default. If PowerShell is installed, standardise on a supported PowerShell 7 package and remove legacy or unmanaged copies.
`ISM-1622`	PowerShell uses Constrained Language Mode.	There is no strong WDAC/AppLocker-backed Ubuntu equivalent. Avoid broad PowerShell use on desktops, or restrict it to controlled admin workstations and jump hosts.
`ISM-1623`, `ISM-1889`	PowerShell and command-line process creation events are centrally logged.	If PowerShell is present, forward its logs. For Linux command execution, use `auditd`, Sysmon for Linux, osquery, Wazuh, Elastic Agent, Microsoft Defender for Endpoint, or another EDR path.
`ISM-1815`, `ISM-1906`, `ISM-1907`, `ISM-0109`, `ISM-1228`, `ISM-0123`, `ISM-0140`, `ISM-1819`	Logs are protected, analysed, reported, and used in incident response.	Forward browser, application, audit, and EDR telemetry to a central platform with retention, access control, and tamper protection.

Linux-friendly commercial additions

If you need more than native browser policy and AppArmor can offer, the cleanest commercial additions are usually:

- **Cloudflare Remote Browser Isolation**
- **Kasm Workspaces**
- **Menlo Security**
- **secure web gateways** that enforce content filtering and isolation policies

These work well with Linux because the control is delivered through the browser or network path, not through a Windows-only endpoint dependency.

A practical Ubuntu hardening baseline

For Ubuntu 26.04 desktops, I would start with:

- one or two approved browsers only
- enterprise browser policies with controlled extensions
- no unmanaged extension installs
- controlled protocol handlers and automatic file handling
- ad and script reduction for high-risk groups
- AppArmor on user-facing apps
- patched PDF and document tooling
- Kasm for risky browsing, unknown links, or regulated web access roles
- isolated browsing for risky internet research roles

That is a sensible baseline without becoming unmanageable.

The bottom line

User application hardening is one of the more achievable Essential Eight mitigations on Ubuntu 26.04 LTS.

The trick is not to overcomplicate it. Harden the browser with real managed policies, reduce active content, control extensions, keep document handlers patched, and use isolation for higher-risk workflows. Linux gives you enough native leverage here that the control can be both practical and strong, and products like Kasm give you a credible commercial step-up when endpoint hardening alone is not enough.

References

- ASD Essential Eight maturity model and ISM mapping (October 2024)
- Canonical releases Ubuntu 26.04 LTS Resolute Raccoon
- Firefox enterprise policies
- Customize Firefox using policies.json
- Chrome Browser quick start for Linux
- Set Chrome policies
- uBlock Origin Lite - Chrome Web Store
- AppArmor on Ubuntu
- Kasm browser isolation docs
- Kasm web isolation use cases
- Cloudflare Remote Browser Isolation
- Menlo Remote Browser Isolation
- Elastic Support Matrix

Downloaded from https://www.gavinj.net/post/essential-eight-linux-user-application-hardening
Generated November 14, 2026. Copyright Gavin Jackson. All rights reserved.