

Kernel CVEs at AWS Scale Two Years of Empirical Findings

Dylan Johnson, Justinien Bouron,
Alice Zielman, Gunnar Kudrjavets

Amazon Web Services

{dgj, jbouron, zielman, gunnarku}@amazon.com

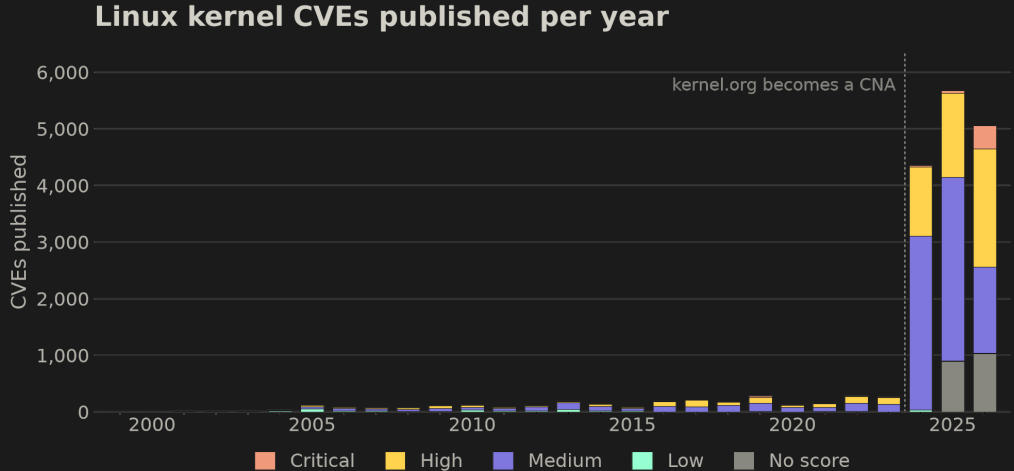
Seattle, WA, United States



**LINUX
PLUMBERS
CONFERENCE**

PRAGUE, CZECHIA
OCT. 5-7, 2026

Linux kernel CVEs over the years



partial data for 2026

Outline

1 Automating CVE triage to keep up

How we improved our process to keep up with the incoming flow of CVEs

2 Upgrading to the latest LTS to stay secure

Why staying on an older LTS no longer works in 2026

3 Findings about CVEs and LTSes

Empirical data about CVEs, LTSes, and upgrades

CVE Triaging - *What*

"We do not know your use case, and we do not know what portions of the kernel that you use, so there is no way for us to determine if a specific CVE is relevant for your system."

CVE process description - [kernel.org](https://kernel.org/doc/bugzilla/bugzilla.html)

- The impact of a CVE can differ drastically between two Linux environments
- Triaging CVEs is about assessing the *impact* of those CVEs in the context of *your* environment, and how to *remediate*

CVE Triaging - *What*

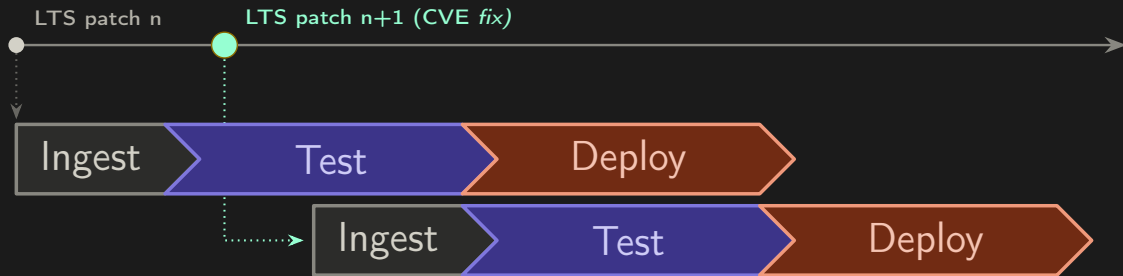
Remediation depends on the impact and your risk tolerance

Option	Speed	Risk	Pick it when
Do nothing	—	none	CVE does not impact your environment
Take the next LTS patch release containing fix	slow	medium	impact is low enough to wait
Cherry-pick the fix	fast	high	faster remediation needed
Work around the impact	fastest	low	kernel update not possible

Decision must be made on case-by-case basis

CVE Triaging - *Why*

Reason 1: Speed up remediation against a CVE

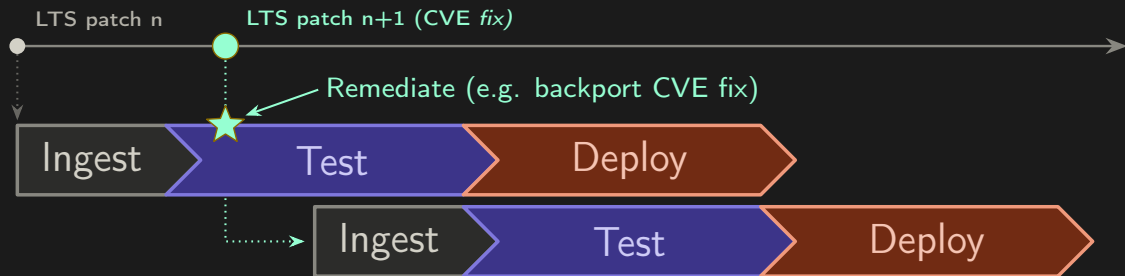


- LTS patch release cadence does not match our deployment schedule
- A new patch needs to wait for the next cycle

This may delay the deployment of a CVE fix!

CVE Triaging - *Why*

Reason 1: Speed up remediation against a CVE



Triaging allows us to identify those CVEs that cannot wait and remediate them sooner

CVE Triaging - *Why*

Reason 2: Some CVEs don't have a fix in *your* LTS

- Backporting CVE fixes gets harder on older LTSes
- Maintainers' time is limited, not all fixes get backported to all LTSes
- Older LTSes accumulate unpatched CVEs

~1,700

unpatched CVEs in 5.10

~1,200

of those have been
unfixed for >90 days

Measured on the latest 5.10 patch release as of Sep 16th 2026

Backporting those fixes in your tree is on you

Triaging CVEs - *How*



More than 50× increase after 2024
Manual triage is no longer possible

- Triaging by hand would require a full-time job
- Spikes of hundreds of CVEs in one day are common
 - 848 CVEs on 2026-08-15 alone!

Automation is the only way to keep up

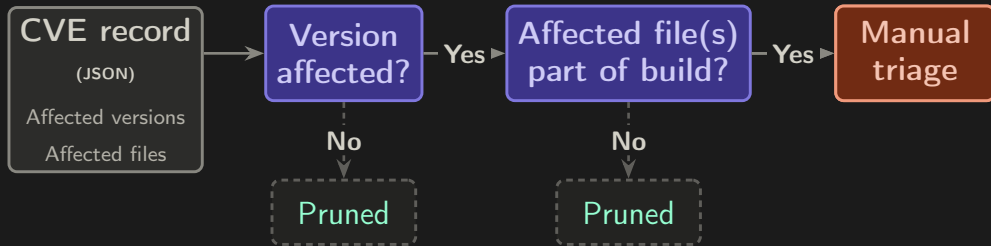
CVE Triage Automation - Auto-pruning

“Very few CVEs are applicable for you!”

from “Linux Kernel CVEs, What Has Caused So Many to Suddenly Show Up?” - Greg Kroah-Hartman, 2024

- Many CVEs are affecting parts of the kernel you do not build into your kernel image. Those CVEs do not impact you!
- Filter out those and manually triage the rest
- CVE records contain lists of affected versions and affected files
 - Everything is available in JSON format, easily scriptable
 - Use those to prune irrelevant CVEs!

CVE Triage Automation - Auto-pruning



~70%* of CVEs get pruned, only ~30% need manual triage
Can we do better?

* Heavily depends on your configuration and environment

Auto-pruning Optimization - Attack Surface Reduction

- How can we auto-prune *more* CVEs? **By removing bloat!**
 - Disable unneeded features/CONFIG_*s at build time

Fewer
features

=

Less code
in your image

=

More
auto-pruning

- Iterative process: manual triage tells you which features to drop next

Auto-pruning Optimization - Attack Surface Reduction

Pros

- + Can significantly increase the number of auto-pruned CVEs
- + Smaller, leaner kernel image

Cons

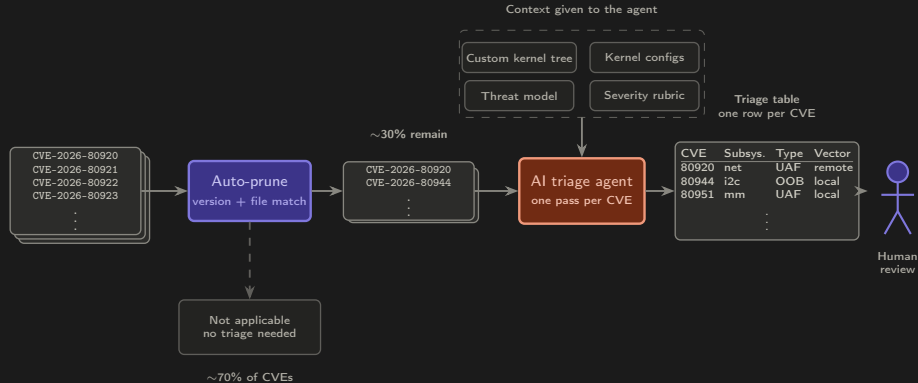
- Requires knowing *all* use-cases of your kernel
- 32k+ CONFIG_*s to evaluate
- Disabling the wrong config can cause subtle bugs or performance regressions

Great idea in theory, deceptively difficult in practice

CVE Triage Automation - AI

- AI models improved significantly between 2024-2026, making them good enough to triage CVEs
- Why not have an AI agent do the triage of the CVEs left after the auto-prune step?
- The only manual step is to review the results

CVE Triage Automation - AI-assisted Pipeline



The only manual step is reviewing the table

CVE Triage Automation - Findings & Limitations

Human		AI agent
minutes	→	~3.4 s
per CVE		per CVE

Triage time, human vs. AI

- Significant speed up in triaging time
- Requires detailed context about your environment

Human review is now the bottleneck
Hard to optimize any further

Can I stay on my old LTS then?

Being able to keep up with the incoming flow of CVEs does **not** mean you can stay on an older LTS

- Older LTSes tend to accumulate unpatched CVEs over time
- Those CVEs will leave your with two choices
 - Accept the fact that you're running a vulnerable kernel
 - Backport the fix yourself
- Backporting fixes is not without risks ...

Backporting CVE fixes yourself is risky

- You are running an untested tree
You may introduce additional bugs or performance regressions
- May require maintainer-level familiarity with a sub-system
Older LTSes show higher code divergence with mainline, which means tougher conflicts. Can use AI for this, but how do you verify its work?
- CVE records may not contain *all* the commits required to fix the issue
You may still be vulnerable!

Backporting CVE fixes in your tree is unsustainable!

What then?

The concept of staying on an older stable kernel is obsolete

Being able to triage through 1000s of CVEs cannot save you
Backporting CVE fixes yourself will backfire

The only path forward is to upgrade!

Upgrade to latest stable?

“... one must choose to either update to latest stables... or one must triage every fix to find only those that need to be applied based on one's threat model.”

Kees Cook, quoted by Greg Kroah-Hartman, Open Source Summit 2025

Late 2024: we ran the math on the first path.

Lifetime of a non-LTS stable tree
Our transition time, at that point

~2–3 months
12 months

Result

EOL before we finished deploying it

Which LTS? The newest one we can qualify.

1 Newer LTSes receive fixes sooner

Measured as time from fix merged to mainline to fix in tree

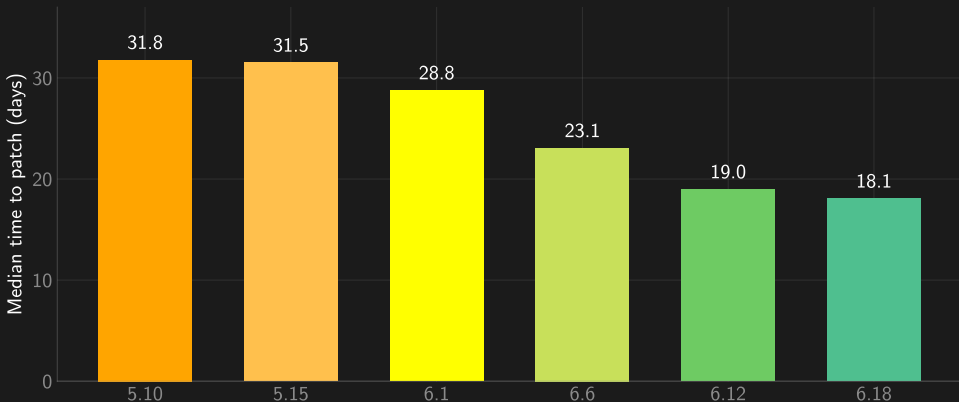
2 Newer LTSes carry less standing debt

Measured as number of CVEs impacted and unfixed at a point in time

3 Newer LTSes receive fewer unfixed CVEs per month

Measured as rate of new unfixed CVEs per month

Newer LTSes receive fixes sooner



Reading: Median time to patch trends lower on newer LTSes.
Measured as time from fix merged to mainline to fix released in LTS

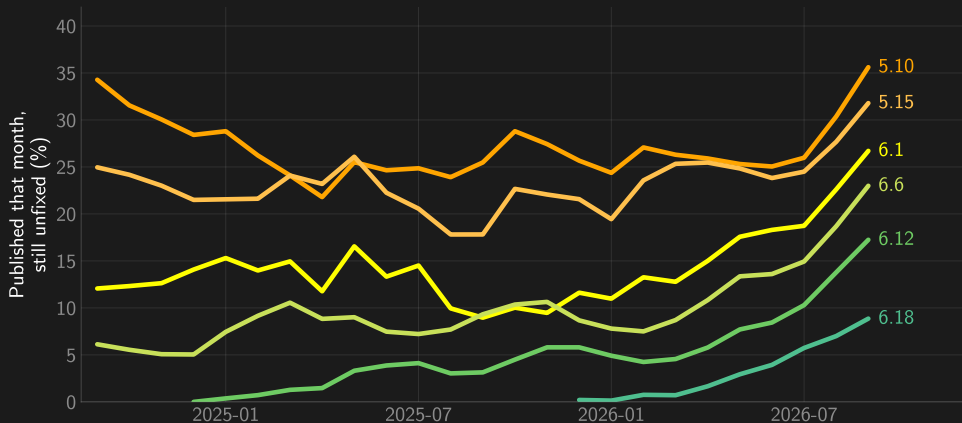
Newer LTSes carry less standing debt

LTS	Affected	Fixed	Unfixed	% unfixed	Unfixed, >90 days
4.19 (EOL)	5,431	1,706	3,725	69%	2,552
5.4 (EOL)	6,351	2,870	3,481	55%	2,130
5.10	7,610	5,901	1,709	22%	1,258
5.15	8,590	7,023	1,567	18%	1,104
6.1	8,227	7,165	1,062	13%	647
6.6	7,702	6,940	762	10%	389
6.12	6,723	6,240	483	7%	172
6.18	5,231	5,002	229	4%	39

Reading: Older LTSes contain more unfixed CVEs.

Measured as number of CVEs impacted and unfixed as of Sep 16th 2026

Newer LTSes receive fewer unfixed CVEs over time



Reading: Older LTSes receive more unfixed CVEs per month.

Measured as percentage of newly created unfixed CVEs impacting each LTS per month

Just upgrade to the latest LTS?

- Newer LTSes get patched sooner
- Newer LTSes have fewer known defects
- Newer LTSes have less backport burden

**If you can't upgrade to stable,
the latest LTS is the next best option!**

How much did it cost to upgrade?

Upgrade	LTS Age Gap	Cost
4.19 → 5.10	25 months	24 months
5.10 → 6.12	47 months	12 months
6.12 → 6.18	12 months	6 months

Better tooling, broader test coverage, and an experienced team mattered more than the LTS age gap

What changes were needed to upgrade?

We tracked each issue that we encountered during the 5.10 to 6.12 upgrade. An issue is defined as ticket, symptom, root cause and fix.

Where we fixed it	Issues
Kernel space	45%
Userspace	43%
Kernel config	7%
Firmware/Hardware	5%

15 % of the kernel space fixes were upstreamed.

What changes were needed to upgrade? (cont)

Upstream change	Effect	Fix
VMA storage → maple tree	mmap()/munmap() slower	reduce mmap ops
CFS → EEVDF	fork() timing shifts	changed fork patterns
Default blk max__sectors raised	firmware I/O timeouts	sysctl tuning

Hyrum's Law: every observable behavior gets depended on by someone.

Can we eventually reach the latest stable?

The gap is smaller than it was in 2024:

Lifetime of a non-LTS stable tree	~2–3 months
Our last transition	6 months
To close it	roughly halve again (?)

What would have to change

Better rebase tooling, faster qualification, and concurrent upgrades.

What we would be trading away

Many issues were found via *soak time*. A quicker update means less soak time.

Does this mean we can stop triaging CVEs?

NO

Not unless we are able to deploy the latest stable branch continuously.

The future of kernel CVEs

- AI models are gaining significant capabilities
- They are now able to find and exploit bugs, **even before a CVE is published**
- Is being on an LTS, or even stable, enough?
- What does the exposure window need to be in the future?

Lots of unknowns ahead of us ...