Whitepaper

TuxCare

The Patching Process that Keeps Open Source Secure and Enterprise-Ready for the Long Run



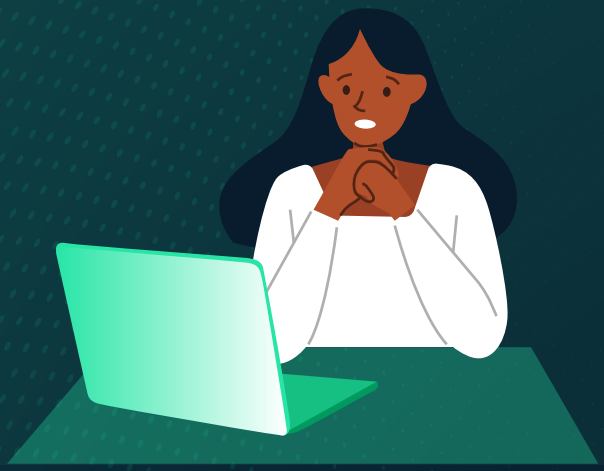
When Open Source Outlives Its Maintainers

When an open-source component reaches end of life (EOL), it ceases to receive security patches from its original maintainers.

The software behind that component, however, keeps running in production – just without the security updates flowing in.



Organizations still depend on these components to power critical systems, keep revenue flowing, and meet regulatory expectations tied to workloads that can't be retired overnight.



Without ongoing patching, every new vulnerability increases security, operational, and compliance exposure – pressuring teams into rushed upgrades that risk breakage, downtime, or costly rewrites.

The need to stay secure and compliant while minimizing disruption puts engineering teams under considerable strain.

The journey most enterprises follow when continuing to run EOL open-source software typically unfolds like this:

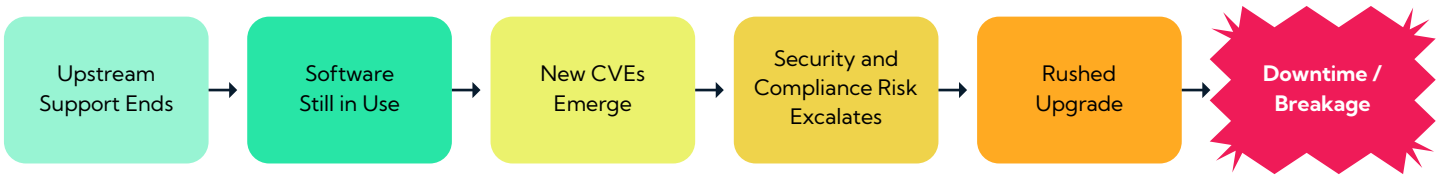


Figure 1. How risk accumulates when software remains in use after EOL.

Keeping EOL software secure and compliant requires continuous monitoring of upstream ecosystems, rapid analysis of emerging CVEs, expert handling of dependencies, and the engineering capability to produce safe and stable backports every time a new vulnerability appears. However, few internal teams have the specialized expertise or the bandwidth to sustain this level of vigilance over the long term.

Organizations that can't upgrade immediately are left with three choices:

1. Absorb the risk and hope for the best, accepting potential ransomware incidents, service disruptions, or costly regulatory penalties.

2. Build and maintain their own EOL patching pipeline – which includes the burden of developing internal tooling, processes, and expertise to support codebases that no longer receive upstream fixes.

or...

3. **Rely on a dedicated provider** with the visibility, engineering depth, and proven methodology to deliver continuous, enterprise-grade security throughout the EOL phase.

For many enterprises, this is why TuxCare's Endless Lifecycle Support has become the trusted way to keep critical systems secure and compliant – all without forced upgrades or operational disruption.

More than a Decade of Engineering in Every Patch

10+ years building the discipline behind TuxCare's enterprise-grade vulnerability patching

TuxCare's patching methodology is the result of more than a decade spent refining a predictable, repeatable process trusted in some of the world's most demanding environments.

With visibility across thousands of packages and diverse ecosystems, we spot patterns that individual maintainers can't – allowing us to detect issues early, prioritize fast, and ship secure fixes without slowing teams down.

Our engineers have lived through it all, including:

- Dependency chain breakages
- Cascading build regressions
- Zero-day vulnerabilities with no margin for error

This depth of experience means we don't just secure open-source code – **we understand how it's built, deployed, and sustained in real enterprise systems.**

Every patch is designed to be stable, compatible, and production-ready from day one, not just technically correct.

This strong foundation has enabled TuxCare to scale effectively...

from a handful of supported frameworks to hundreds of open-source projects (and on track for hundreds of thousands)

Between 2024 and 2025 alone, we expanded coverage from one framework to hundreds of projects across three major languages – a trajectory powered by a mature, automated pipeline built to scale without compromise.

Why Scale Matters

This evolution reflects a long-term commitment to enterprises that can't afford to fall behind security threats or compliance requirements.

The more we support, the better we get: each new project strengthens the process that powers all others.



TuxCare's Automated Patching Process

Three pipelines power TuxCare's automated patching system:

1. **Project onboarding**
   bring new projects under our maintenance fast
2. **CVE remediation**
   identify and fix issues early
3. **Project release**
   deliver secure, tested, production-ready updates

Together, they ensure continuous security coverage at scale.

Inside the Engine:
The End-to-End flow that Powers Continuous EOL Security

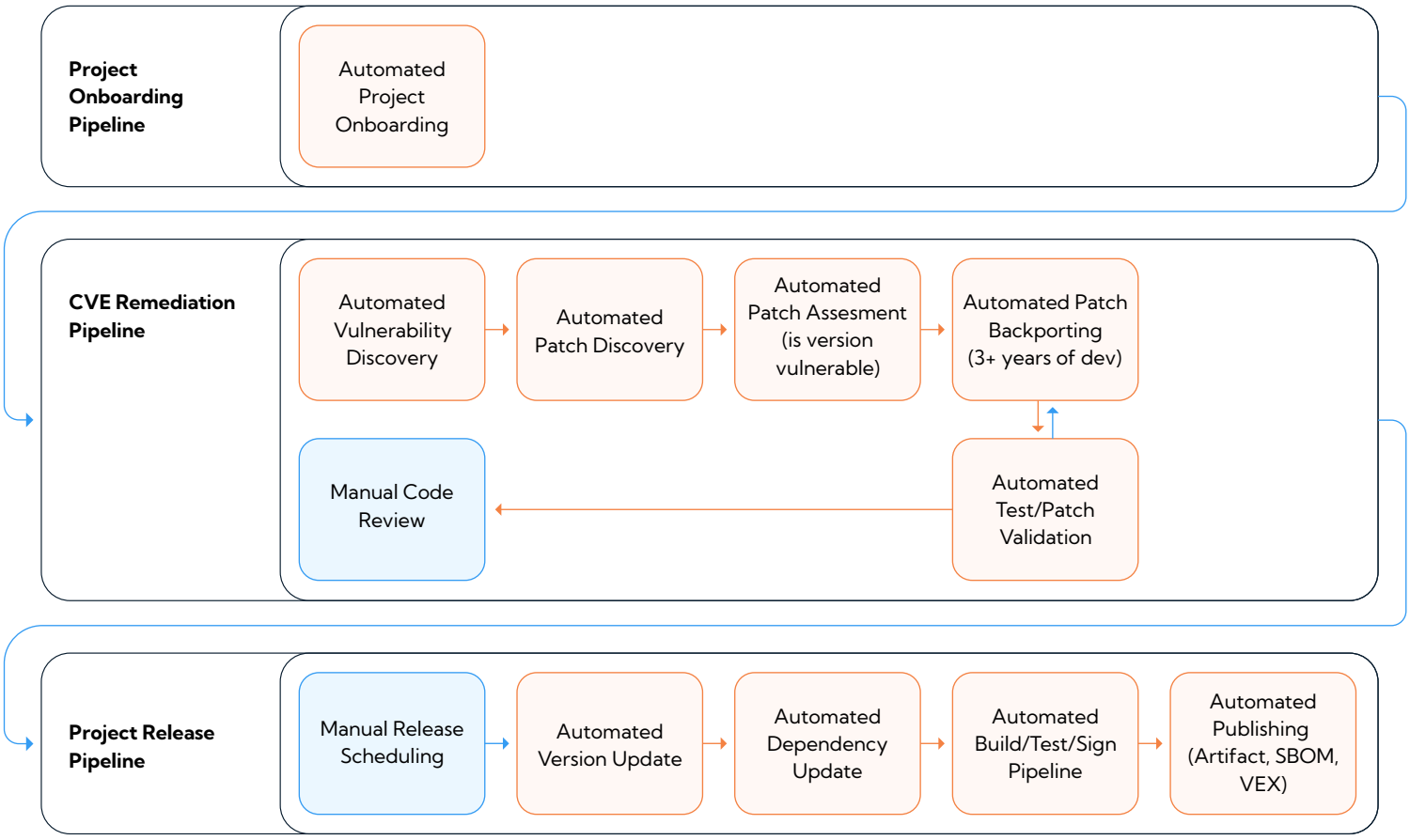


Figure 2. TuxCare's patching process.

1.

The Project Onboarding Pipeline

The onboarding pipeline is activated whenever support is required for a project not yet in our catalog, whether it's a customer-requested framework or library, a vulnerable dependency of an already supported project, or a newly EOL version of an existing one.

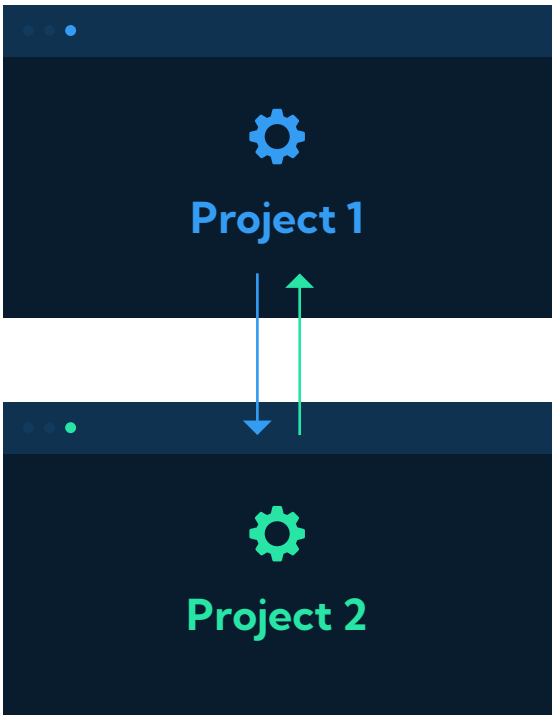
It establishes everything required to **build, test, and maintain** the project long term, ensuring we can deliver reliable backports throughout its extended lifecycle.

When No Two Projects Look Alike

When onboarding an open-source project, we must establish the ability to fully build, test, and maintain the software to deliver backported security fixes after EOL. This process is essential because open-source projects vary significantly in structure, tooling, and dependency management.

For example, Java projects built with Ant are highly heterogeneous: each one typically defines its own directory structure, build targets, and dependency management logic. By contrast, projects built with Maven or Gradle introduce some level of standardization, but still not enough to guarantee uniformity.

However, even though open-source projects vary significantly in structure and build practices, having a broad view across many ecosystems helps us recognize recurring patterns and distill them into reusable onboarding templates and automated workflows.



From Diversity to Determinism

Reusable onboarding templates capture the common patterns across languages and build systems, accelerating much of the onboarding work.

But they remain flexible enough to support unique or highly customized projects that fall outside those norms.

The example to the right is an excerpt from one of our onboarding templates. It captures the key rules that govern how an upstream project is built, versioned, and updated, allowing it to be maintained automatically over the long term.

Instead of listing dependency versions, it lists the dependency artifacts and describes where their version numbers are defined within the build configuration, allowing patched components to be rebuilt and published consistently.

Onboarding templates also specify how the project's own version should be incremented during releases and define the commands and environment required to compile, test, and validate the project reliably.

By encoding these details in a machine-readable format, the template provides a repeatable blueprint that enables deterministic builds – the prerequisite for safe patch backporting and consistent releases throughout the project's extended lifecycle.

yaml Копировать код

```yaml
package_name: auto
type: maven
run_strategy: legacy
runner_command: ...

dep_artifact_set:
  # Example: jakarta.xml.bind-api
  - id: jakarta.xml.bind-api
    group: jakarta.xml.bind
    files:
      - strategy: set_property_pom
        filename: pom.xml
        property: jakarta.xml.bind.version
      - strategy: no_version

  # Additional dependencies use the same pattern
  # ...
```

Figure 3. Example onboarding template.



The template ensures that updated artifacts are produced in a predictable and deterministic manner throughout the project's extended lifecycle.

Handling the Outliers

When an existing template cannot fully interpret a project, AI-assisted analysis helps us understand its build structure, dependency layout, and versioning logic.

This insight is then distilled into a new deterministic template usable for future onboarding – ensuring AI accelerates understanding without replacing predictability.

2.

CVE Remediation Pipeline

This pipeline provides continuous, automated detection and remediation of vulnerabilities across supported projects, powered by 24/7 monitoring from multiple authoritative sources.

Once a vulnerability is detected, upstream fixes are backported using deterministic automation and AI-assisted analysis – followed by expert validation to ensure accuracy, stability, and compatibility.

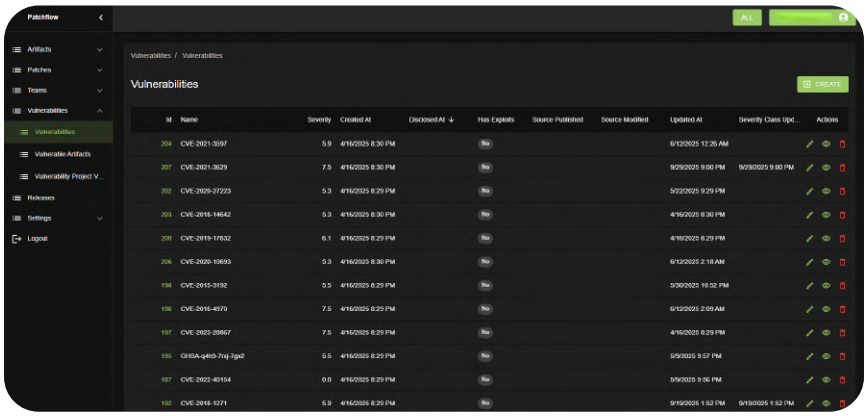


Figure 4. Vulnerability monitoring view in Patchflow, our internal patch management and backporting platform.

Automated Patch Discovery

When a vulnerability needs remediation, we automatically search multiple sources for existing upstream fixes – including GitHub repositories, advisory databases, security trackers, and other official channels.

Automated Patch Assessment

The majority of patches will, of course, target the newer community-supported versions, so we must determine whether the older versions we support are also vulnerable.

Our tooling automates this analysis by identifying whether the patched code exists in the older version, whether the affected functionality or dependency was present at that time, whether the vulnerable code path can actually be triggered, and several additional factors.

Altogether, this provides a sophisticated and reliable assessment of whether our supported version is truly impacted.

If it's not vulnerable,

we mark it as not vulnerable and move on.

If it is vulnerable,

we proceed to the next stage of automated patch backporting, i.e., taking the patch for the newer version and adapting it to the older version.

Adapting Today's Fixes for Yesterday's Code

Patch backporting is where much of the engineering work happens.

Because upstream code evolves, applying patches to old versions requires careful adaptation – not a copy-and-paste job.

In practice, effective backporting relies on a combination of algorithmic automation, deep domain expertise, and selective LLM-assisted analysis to adapt an upstream fix to an older EOL codebase safely and deterministically.

While some patches can be applied as is, many backports require significant adaptation. Several structural challenges make this stage the most technically demanding part of the remediation pipeline.

Here are the main challenges that enterprises will face when backporting patches:

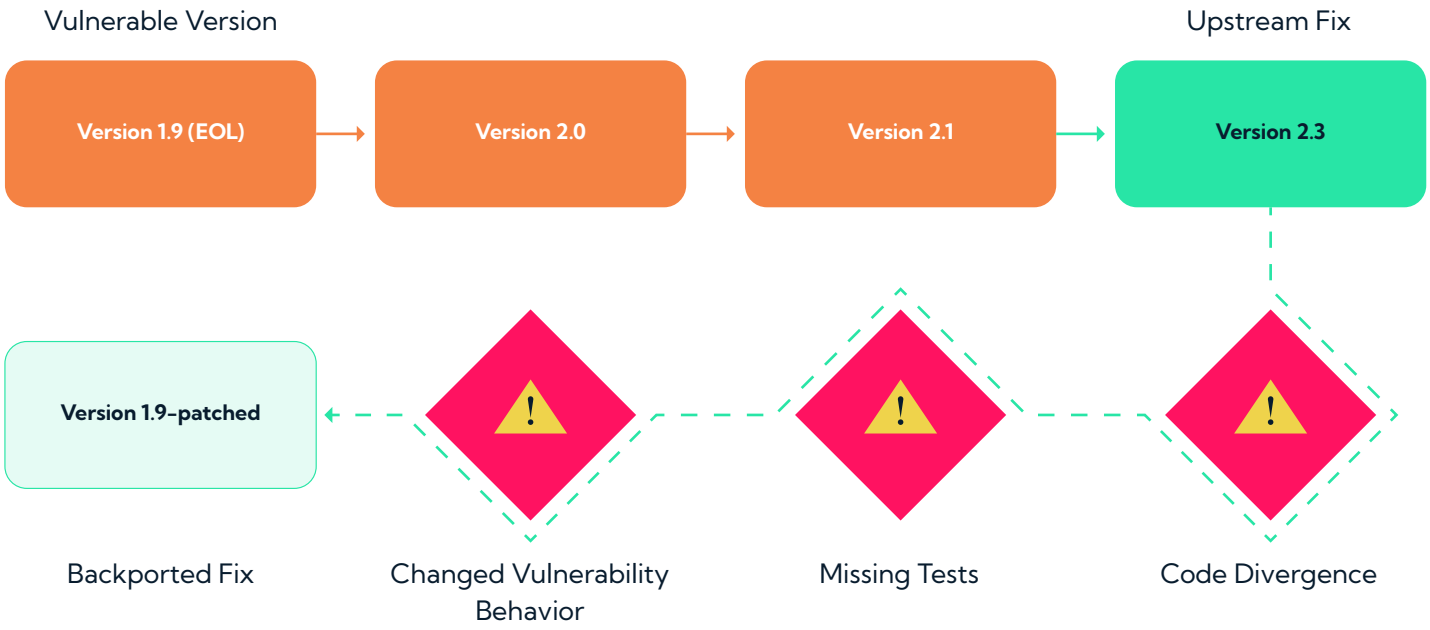


Figure 5. Main patch backporting challenges.

The challenges of backporting aren't just technical.

They're the cumulative result of time, ecosystem drift, missing context, and the hard reality that yesterday's code no longer maps cleanly to today's fixes.

These are the main challenges enterprises face when backporting:

Code divergence over time

Upstream changes may rely on APIs never present in older versions.

Because upstream code evolves, patches are usually written for a codebase very different from the older EOL version. The patched code may no longer exist in the same form, or it may rely on APIs or behaviors introduced long after the version we support was frozen. Backporting, therefore, requires careful adaptation, not blind application.

Gaps in test coverage

Outdated or missing tests mean extra validation is needed.

Older versions frequently include limited or outdated test suites, and, in some cases, no tests at all. Even when tests exist, they may not cover the vulnerable code path or the interactions introduced by the patch. This means that backported fixes must be validated using additional methods.

Deep domain knowledge

(especially in areas like crypto, networking, memory safety, or serialization)

Many vulnerabilities involve subtle behavior in networking, cryptography, concurrency, serialization, memory management, or specific protocol implementations. Understanding the security implications and the architectural intent of the original fix often requires specialized expertise. Automation can accelerate the process, but interpreting the accuracy and security of a backported fix frequently demands human judgment.

AI as a Force Multiplier in Patch Backporting

AI accelerates analysis, contextual understanding, and expert workflows – especially in older, sparsely documented codebases.

But, to remain reliable, every AI-assisted change is evaluated through controlled workflows and expert oversight.

AI increases speed.

Human expertise guarantees precision.

While deterministic automation forms the backbone of our patching pipelines, AI has become an increasingly powerful accelerator in the process. When used with the proper guardrails, AI enables faster analysis, deeper contextual understanding, and more efficient expert workflows. It helps our engineers summarize upstream diffs, understand code intent, identify missing dependencies, and highlight areas that require adaptation. This contextual assistance is particularly valuable when working with older, highly divergent, or sparsely documented codebases.

To ensure reliability, because security patches must behave consistently and predictably, AI-generated changes are always validated through a carefully controlled workflow. **Human engineering expertise provides the final verification that ensures each backport is not only correct but maintains the stability and security guarantees enterprise environments require.**

This balanced approach enables AI to meaningfully enhance speed, comprehension, and scalability, while maintaining the trust, predictability, and correctness that define enterprise-grade patching.

```
AI-powered analysis coupled with
expert validation delivers
enterprise-grade security.
```

Automation of Patch Backporting

That said, automated patch backporting is achievable when supported by the right combination of deterministic algorithms and carefully constrained AI assistance.

Algorithmic analysis handles the aspects that must be precise, while AI provides contextual insights, summarizing failures, and assisting experts when human intervention is needed.



To handle the wide variety of situations encountered in legacy codebases, we employ several complementary backporting strategies, each tailored to different levels of code divergence and context availability:

Auto-apply when patches can be used as is

It works well in relatively recent versions and rarely works if the destination version is significantly outdated (more than one major release behind).

Scope-based approach

If a patch can't be applied using just the nearby lines of code as context, we fall back to a bigger anchor, e.g., the entire function that contains the change.

Interdependent patch backporting (patent pending)

We examine the list of commits (patches) between the vulnerable repository state ("target") and the fixed repository state ("after fix"), identify all commits in between, and intelligently filter out irrelevant ones, leaving only the minimal required commits for the vulnerability fix.

AI-assisted approach

Used only as a last resort due to its non-deterministic nature. We rely on an agent-based setup, where the agent is given the repository, CVE details, contextual information, and the upstream patch to propose a backport, which is then reviewed by human experts.

The resulting patch then moves into the validation stage, where we confirm its correctness and security before release.

Automated Patch Validation

All backported fixes undergo automated validation using the existing tests, supplemented as necessary by additional methods, including:

- syntactic checks through compilation
- selective AI-assisted review
- final human inspection.

These steps ensure that the patched code compiles correctly, behaves as intended, and does not introduce regressions.

Unlike LLM-based code-review assistants that assess only the correctness or consistency of the patch itself, our AI-assisted review is designed specifically for security backporting. Instead of analyzing the patch in isolation, the model is provided with the original vulnerability description, the upstream fix, and a structured explanation of how that fix was adapted for the older codebase.

This additional context allows the AI to confirm that the backported patch actually fixes the vulnerability, not just that the code is syntactically correct. This is something generic LLM-driven review tools are not designed to determine.

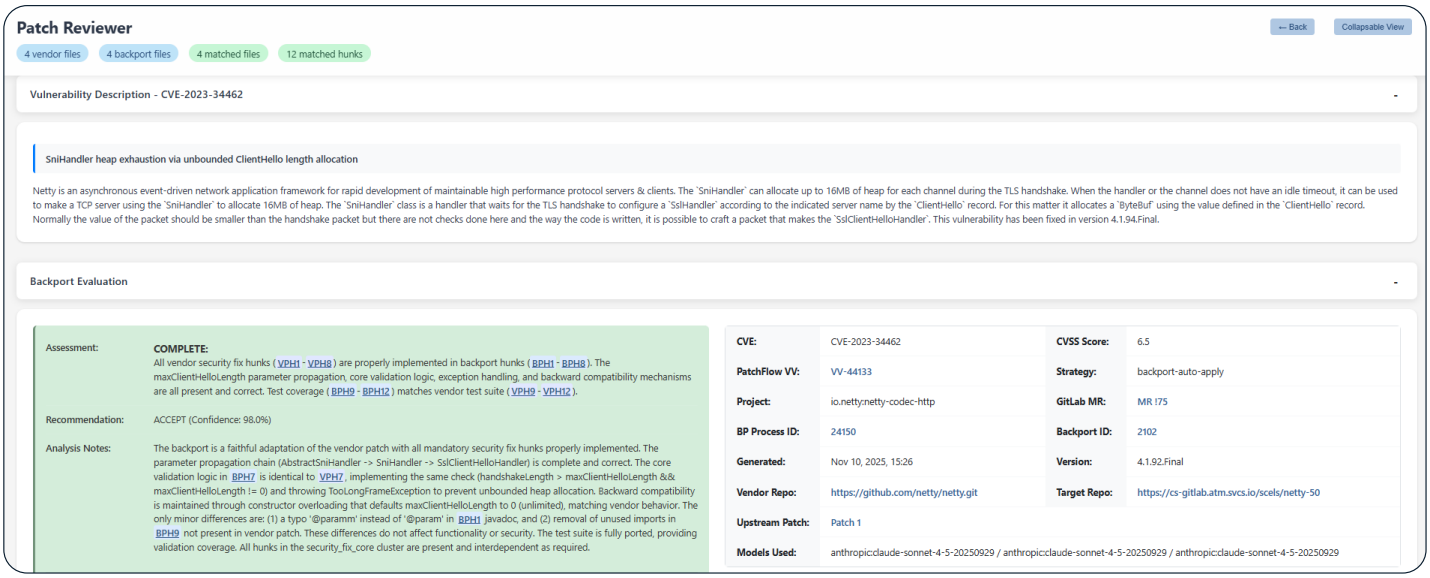


Figure 6. Example of the AI-assisted patch review in our internal system.

Once a patch has been backported, verified to compile successfully, and any existing tests pass, we often create additional tests specifically to validate the vulnerability, which confirm that the patched version fully mitigates the issue.

The patch is then submitted for human review. We don't blindly trust automation: our experts must confirm that the fix is effective and addresses the vulnerability before advancing to the next pipeline.

3.

Project Release Pipeline

When a project or update is ready, the release pipeline prepares, validates, and deploys all artifacts, ensuring each meets modern standards for secure, transparent, and auditable software supply chains.

The release pipeline automatically updates project versions, resolves and upgrades dependencies, compiles and builds the artifacts, and runs integrity, functional, and regression testing. The final outputs are cryptographically signed and published to TuxCare's repositories, enabling customer package managers, build tools, and orchestrators to consume updates automatically.

In addition to building and testing the software, the release pipeline also embeds essential safeguards that ensure every update meets modern software-supply-chain security standards:

Software Bill of Materials (SBOM)

For every patched release, the pipeline automatically generates a comprehensive SBOM that provides complete visibility into all components, libraries, and dependencies included in the build, along with their precise versioning and cryptographic identifiers. It also lists transitive dependencies that might otherwise go unnoticed.

With this level of transparency, customers always know exactly what is running in their environments, enabling stronger compliance, faster audits, and more reliable risk assessments.

Vulnerability Exploitability eXchange (VEX)

Each release also includes a VEX file that communicates the exploitability status of vulnerabilities associated with the components in that release.

This information enables customers to immediately understand which vulnerabilities have already been remediated by TuxCare patches and which reported CVEs are not actually exploitable in the supported version.

By providing this clarity, VEX context reduces alert fatigue, enabling security teams to focus their attention on real risks rather than reacting to every CVE flagged by automated scanners.

SLSA Level 3 Compliance

COMING SOON

The release pipeline meets SLSA Level 3 requirements, providing customers with verifiable assurance that every update is sourced, built, and delivered through a tightly controlled and fully authenticated process.

All build inputs are securely verified, and every build produces tamper-evident provenance that cannot be falsified.

As a result, customers receive packages that align with rigorous industry standards for software supply-chain security and verifiable build integrity.

Together, these safeguards ensure every TuxCare patch is secure, validated, and ready for enterprise deployment.

A Long-Term Security Foundation for Modern Enterprises

What began with extended support for critical Linux operating systems has grown into a platform capable of securing the entire open-source stack – from EOL operating systems and language runtimes to development libraries, frameworks, and applications.

Over the past two years, TuxCare has expanded its support to hundreds of open-source projects, with coverage on track to reach tens of thousands of them in the years ahead.

This accelerated growth is powered by unmatched ecosystem visibility and a perfected patching process that brings precision and consistency to every fix, ensuring that the technologies customers depend on remain secure and compliant – from direct dependencies to the deeply nested transitive ones.

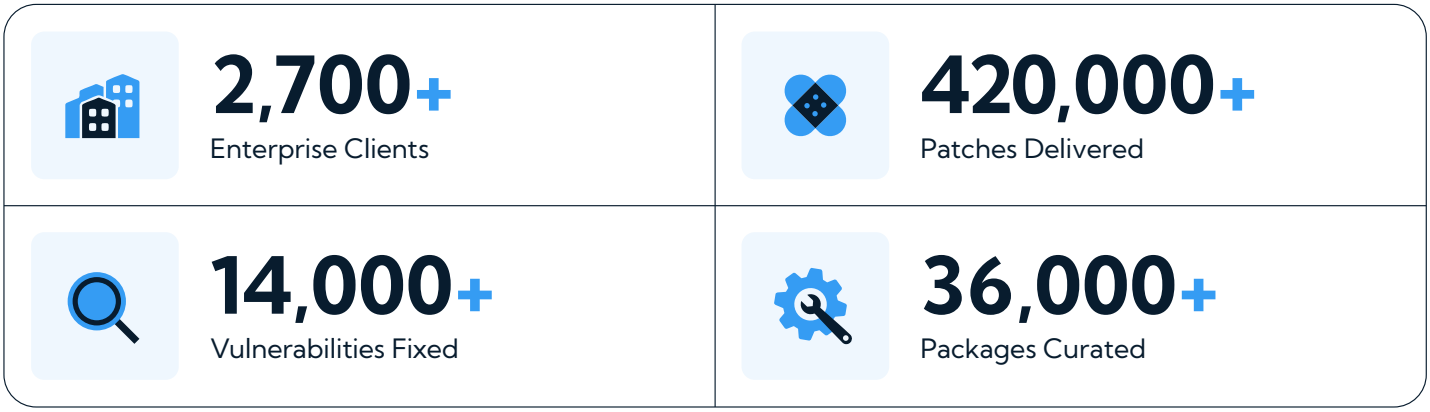


Figure 7. TuxCare by the numbers.

Our customers operate in sectors where security and uptime are crucial – including financial services, telecommunications, healthcare, industrial manufacturing, aerospace, defense, and critical technology infrastructure, as well as numerous **Fortune 500** organizations.

For these enterprises, the benefits are clear: reduced security risk, continuous regulatory compliance, and significant cost savings by avoiding forced upgrades or disruptive code rewrites.

By strengthening resilience and ensuring business continuity across mission-critical environments, TuxCare becomes more than a patch provider. It becomes a long-term security foundation for the open-source software that powers modern enterprise operations.