

Endor Patches

Reduce Risk and Accelerate Remediation



ENDOR LABS

Contents



Introduction	01
What are Endor Patches?	02
Using Endor Patches	03
Prerequisites	03
Configuring a single project	03
Selecting the patch version	06
Using automatic patching	08
Integrating with your artifact repository	09
Reporting in your software bill of materials (SBOM)	11
Build, testing, and licensing practices	11
Testing for security and compatibility	11
Requesting new patches	12
Defining service level objectives	13
Understanding licensing	13
Leaving Endor Labs	13
Conclusion	14



Introduction

If you're running an AppSec program, you're familiar with this workflow: you scan your applications, identify the packages you're using, and try to figure out which vulnerabilities need fixing. And, inevitably, there's a critical vulnerability. Or two. Or ten. Or if you're a typical enterprise, you're struggling with **more than a million code security alerts each year.**¹

Of course, maybe you're even using [reachability analysis](#) to identify which vulnerabilities are false positives so you can better prioritize your remediation efforts. But at the end of the day, you're still left with a list of security issues in your dependencies that need to be resolved—and resolving them isn't straightforward. In fact, it comes with a lot of risk.

As your developers will tell you: upgrading is frustrating and disruptive. In fact, many organizations are paying a **productivity tax of \$28,000 per year per developer**¹ to address security issues—and they still aren't meeting their SLAs for remediation. So what makes upgrading dependencies so hard?

Upgrading is hard because of how the open source community handles security fixes. Imagine this: a developer makes a mistake in version 1.1.0 of a library, but no one notices for months or years.

Eventually, a security researcher identifies the problem, reports it, and the library maintainers release a fix. But by then, the library has moved far ahead—let's say to version 1.9.1—so the fix is included in version 1.9.2.

Between versions 1.1.0 and 1.9.2, a lot may have changed. It's not just the security patch; new features, updates, and breaking changes could have been introduced. Fixing a vulnerability by upgrading to 1.9.2 means adopting all those changes, which often break how your application works. This forces developers to apply the upgrade and also rewrite, test, and debug parts of their own code—a time-consuming and complex process.

Things get even messier if the vulnerability hides in a [transitive dependency](#) (that's a dependency of a dependency). In that case, you're stuck figuring out which direct dependencies to update or how to override that nested transitive one. This adds more time, more complexity, and more headaches. And just when you think you've finally finished — a new vulnerability pops up, and the whole cycle starts again.

Endor Labs provides a better way

Updating open source libraries to patch vulnerabilities shouldn't be a constant disruption. With Endor Labs you can use [upgrade impact analysis](#) to evaluate and prioritize upgrades by complexity and impact. And when upgrading is too risky, complex, or time consuming due to regressions, breaking changes, or new bugs, you can use [Endor Patches](#) to stay safe now while still meeting your SLA requirements.

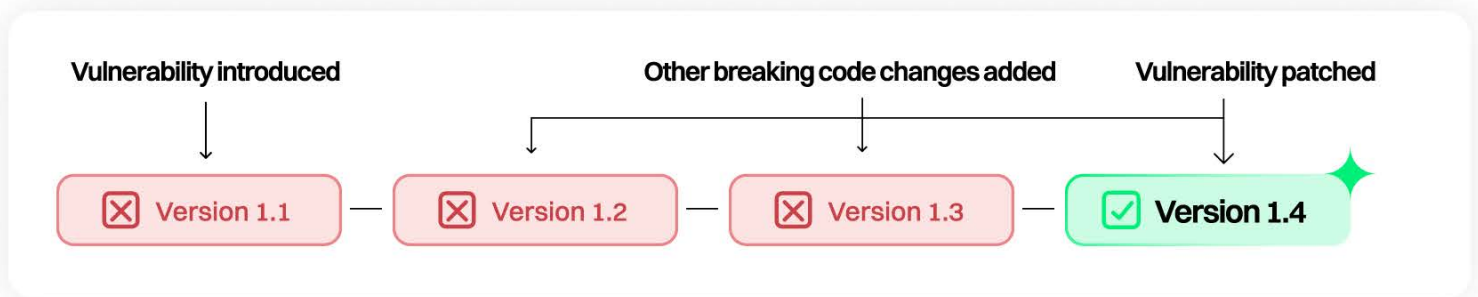
¹ IDC, The Hidden Cost of DevSecOps, 2024

What are Endor Patches?

Endor Patches take the security fix from the latest version of the open source project and apply it directly to the older version you're already using. Each patch is built to ensure:

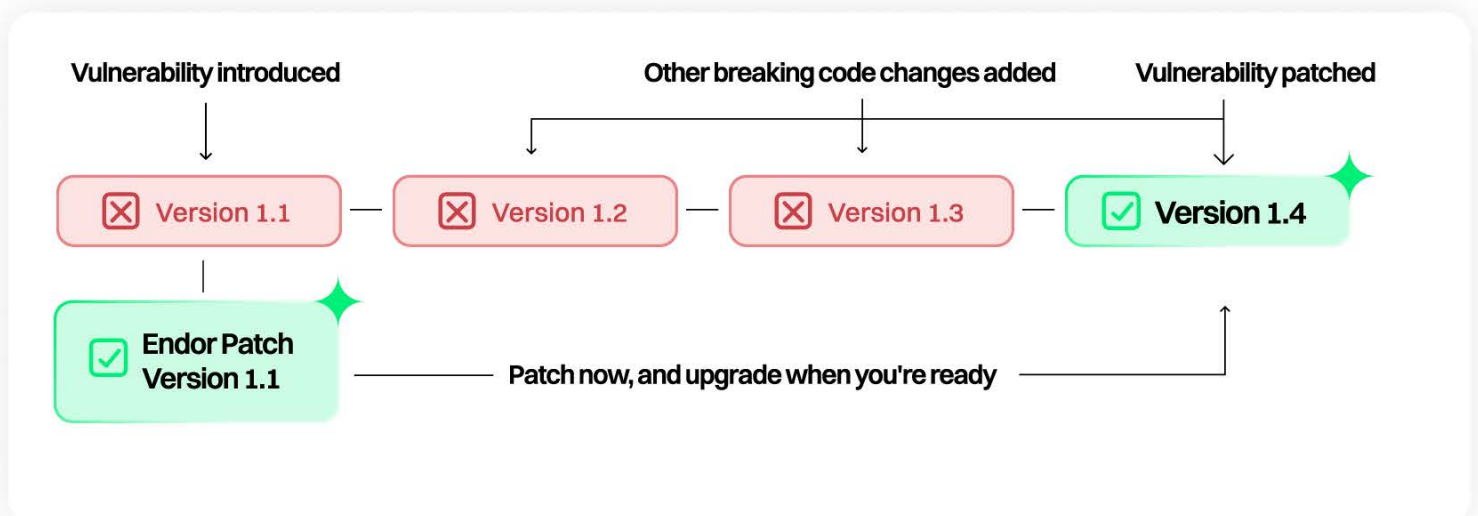
- ◆ **Minimal changes:** Provide only what's necessary to fix the vulnerability.
- ◆ **Maximum compatibility:** Avoiding breaking changes that disrupt workflows.
- ◆ **Focus on security:** No attempts to fix non-security bugs or other functionality.

When creating a patch, we use the security solution that has already been vetted, approved, and implemented by the open source maintainers. These are typically released in the latest version of the open source software. That means there may be several versions between the version used in your code and the version with the security fix.



Open source maintainers could (and sometimes do) go through all of the affected versions and apply the same patch. This process is known as backporting. But because that's a time-consuming and complex process, most maintainers choose not to backport patches.

When Endor Labs backports a security patch, we apply the same security fix, with as few changes as necessary to keep it functional, to every affected version. This way you can use vetted security solutions without introducing complexity from other code changes. Your software engineering teams can upgrade to the latest version of the open source package when they're ready.



Using Endor Patches

You can usually get started with Endor Patches in a few simple steps. You can choose to apply the patched upgrades manually, or automatically apply them whenever they are available. We refer to these methods as **manual patching** or **automatic patching**.

The chart compares the different approaches:

We recommend starting with manual patching for a single project so you can understand how it works before scaling it out further.

	Manual Patching	Automatic Patching
Scope	Applied case-by-case in every place a dependency is used	Automatically overrides any vulnerable version with the secure Endor Patch
Deployment	Can roll out incrementally across projects	Automatically applied on next build
Engagement	Requires developers to review and implement Endor Patches in their projects	No code changes required to apply patches
Versions	You can either pin a version or use the latest version of an Endor Patch (see patch versions below)	Automatically apply the latest secure version of an Endor Patch

Prerequisites

To get started, you will need access to the following:

- ◆ A buildable software artifact to test
- ◆ Access to Endor Labs with optionally the ability to run a scan
- ◆ An Endor Labs license to use the Endor Labs Patch Repository
- ◆ Access to your artifact repository (optional)

Depending on your organization, you may need a small team to get up and running. In some cases one person might have all the skills and access to make it work! Or you'll need at least one security engineer who can configure Endor Labs and run scans and someone who can build software (e.g. a developer). If you want to configure Endor Patches to work with your internal artifact repository, you'll need an administrator for that system.

Configuring a single project

This is the easiest way to try out Endor Patches on a small scale before rolling them out more widely in your organization. In this example we're going to use the project's package manager to integrate with the Endor Labs Patch Factory. In most production environments you will most likely want to [integrate with your artifact repository](#) for the most streamlined experience.

You can get up and running in three easy steps:

01 Create an API key

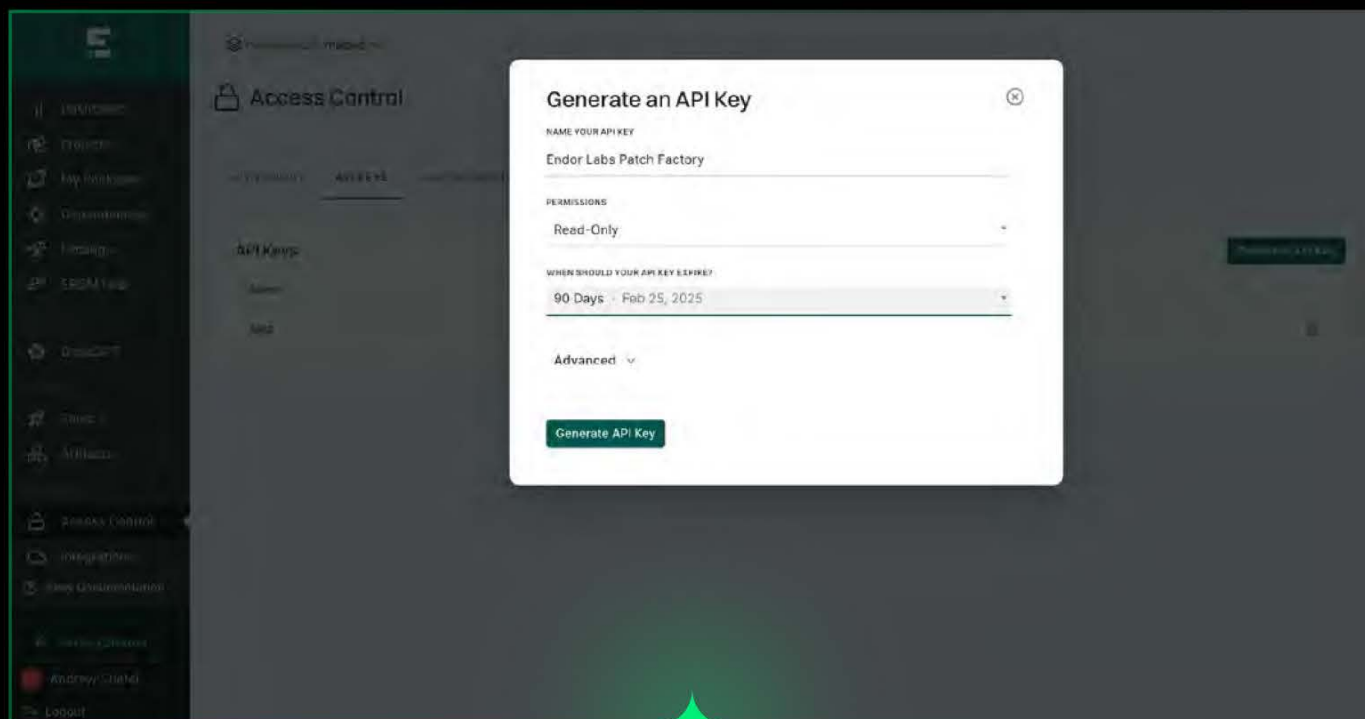
02 Configure your package manager to use Endor Patches

03 Specify the Endor Patch you want to use

01 Create an API key

You must generate an API key to access the Endor Labs Patch Factory. You can do this in a few steps within the Endor Labs Dashboard.

To start, from **Access Control**, navigate to **API keys**. From here you can generate a key, give it a name ("Endor Labs Patch Factory"), and set permissions (Read-Only) and an expiration date.



02 Configure your package manager to use Endor Patches

You can configure either [Gradle](#) or [Maven](#) to work with Endor Patches. In this example we'll use Gradle as the package manager for our project. We need to configure a connection to the Endor Labs Patch Factory.

Open the `build.gradle` in your project and add the Endor Labs Patch Factory to the repositories section. In the example we have added the Endor Labs API Key and API Secret as environment variables in our project.

```
repositories {  
    mavenCentral()  
    maven {  
        url "https://factory.endorlabs.com/v1/namespaces/<namespace>/maven2"  
        credentials {  
            username "$ENDOR_API_CREDENTIALS_KEY"  
            password "$ENDOR_API_CREDENTIALS_SECRET"  
        }  
    }  
}
```

03 Specify the Endor Patch you want to use

Finally, include the Endor Labs patch version you'd like to use in your dependencies. The example below uses Gradle.

And that's it! If you scan the project with Endor Labs, the vulnerability should be resolved.

```
dependencies {  
    implementation("com.fasterxml.jackson.core:jackson-databind:2.9.10.3-endor-2024-09-25")  
}
```


Selecting the patch version

In addition to the initial patch, Endor Labs also releases new patches if additional vulnerabilities are discovered in an open source library in the future. You can control how your software interacts with these versions. When you specify which Endor Patch to use, you have the option to choose between three different versions of each patch:

Pinned

A version associated with a specific patch date for build reproducibility. The patch is pinned at the date you specify.

For instance:

```
v2.9.10.3-endor-2024-07-11
```

Latest

A version with the latest patched version of a library, incorporating all current patches. This can be used by appending `-endor-latest` to a package version.

For instance:

```
v2.9.10.3-endor-latest
```

Auto

A version matching the upstream open-source version, allowing users to use the patched version without code changes.

For instance:

```
v2.9.10.3
```

See use [auto patching](#) for more information on how to automatically use an Endor

You can read more about the advantages and disadvantages of each version below.

Use a pinned version

Using an Endor Patch with a specific date makes it easier to ensure the reproducibility of a build in the future. Your application will only use that specific version of the patch, even if Endor Labs releases a new version in the future. This gives you the most control over the build of your application, and ensures you can repeat the process in the future.

It does come with limitations, however. To start, you will need to implement and update the patch in every repository that uses the vulnerable library or package. You will also have to return to each project in the future to introduce a new pinned version if a new vulnerability is identified.

Here's an example of a **pinned version** in a Gradle project:

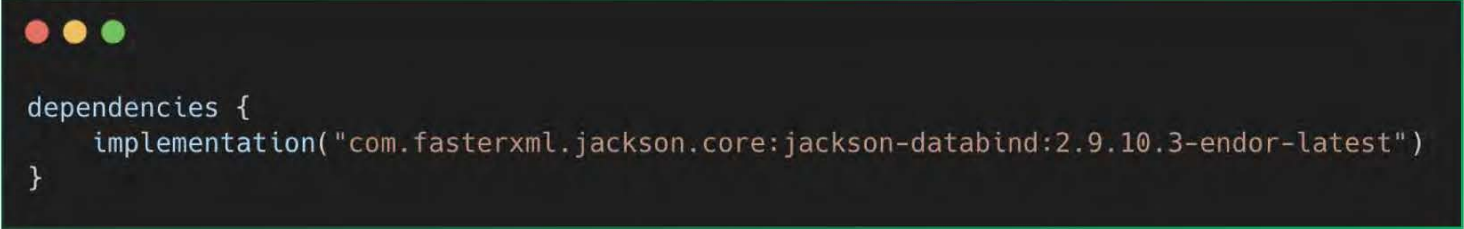
```
dependencies {
    implementation("com.fasterxml.jackson.core:jackson-databind:2.9.10.3-endor-2024-09-25")
}
```


Use the latest version

Alternatively, you can choose to always use the latest version of an Endor Patch by appending `-endor-latest` to a particular library or package. Although you still have to manually configure it the first time, once implemented your project will always be updated with the latest version of the patch from Endor Labs.

This approach sacrifices control over future builds of your application. A new version of the patch may alter the build process or the resulting binaries in unpredictable ways, potentially affecting build reproducibility.

Here's an example of the **latest version** used in a Gradle project



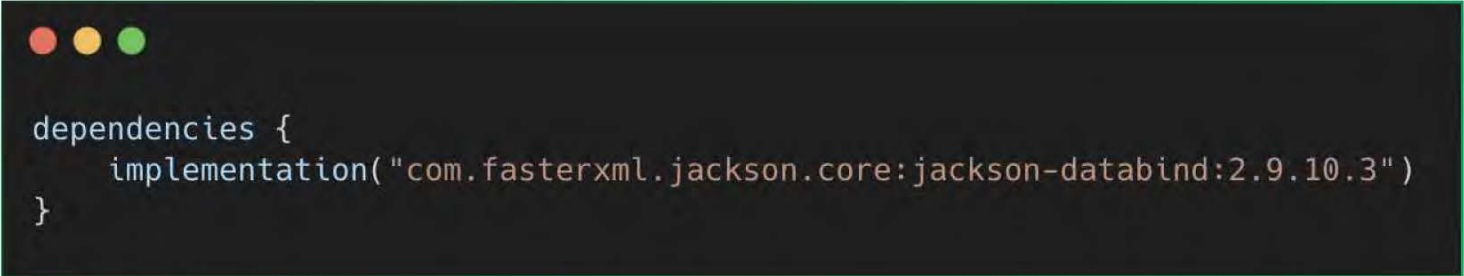
```
dependencies {  
    implementation("com.fasterxml.jackson.core:jackson-databind:2.9.10.3-endor-latest")  
}
```

Use automatic patching

[Automatic Patching](#) allows you to fix security vulnerabilities during each software build, minimizing the effort required to maintain a secure codebase. Unlike the other approaches outlined above, no code changes are required. Instead, once a project is configured, the patches are applied at build time.

While automatic patching greatly simplifies security, there are trade-offs. Automated patching might affect build reproducibility because patches can introduce unpredictable changes. However, Endor Labs mitigates this by applying only the minimum necessary security patch, keeping disruptions to a minimum.

Here's an example of **auto patching** used in a Gradle project, where you can see there are no code changes to the original application:



```
dependencies {  
    implementation("com.fasterxml.jackson.core:jackson-databind:2.9.10.3")  
}
```

We'll explore automatic patching in more depth in the next section.

Using automatic patching

If you work in an organization with dozens or hundreds of repositories, a single open source library or package might be used hundreds, thousands, or even tens of thousands of times in both direct and transitive dependencies. Managing patching across multiple repositories typically means opening pull requests, finding the right people to approve them, and ensuring everything passes the tests. This is hard to accomplish with any scale.

Using Endor Patches isn't just convenient, it's also scalable with **automatic patching**. Whether you're managing a few repositories or thousands, you can patch vulnerabilities consistently and automatically, without causing friction for development teams. Instead of manually creating pull requests and tracking down approvals from developers, patches are applied automatically during the build process. This reduces the back-and-forth, cuts out delays, and lets teams focus on development rather than paperwork.

Addressing transitive dependencies

Fixing transitive dependencies is especially tricky. While most package managers do have a way for you to force an update of a transitive dependency on its own, it's often a very bad idea. Not only are you taking a risk of breaking something else in your dependency graph — because updates are not risk-free — but you're adding a new maintenance challenge. When you override the dependency choices your direct dependencies make, you also run the risk of future updates to the direct dependency not working with the transitive version you chose.

Automatic patching simplifies the complexity of upgrading both direct and transitive dependencies. Endor Patches are designed to be minimal and specific, which means they only address the security issue without causing broader changes. This means developers don't have to manually sift through dependency trees or worry about breaking changes. And your code stays secure without the usual risk of destabilizing the entire application.

Configuring automatic patching

You can set up automatic patching in a few simple steps:

01 Configure your project or artifact repository

02 Enable auto patching in Endor Labs

03 Rescan your projects

01 Configure your project or artifact repository

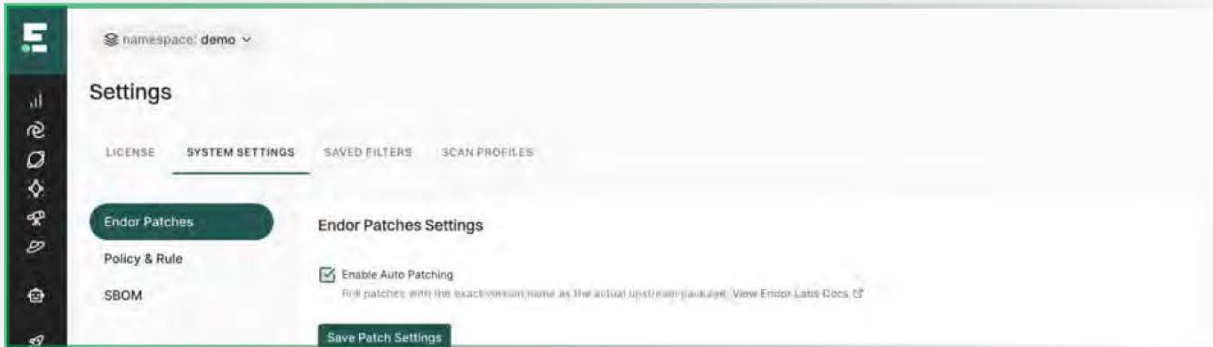
You can configure automatic patching to work either in an artifact repository (like JFrog or Nexus) or directly in the settings of your manifest file (like pom.xml, build.gradle or settings.xml).

To start, you'll need to [create an API key](#). Then you'll need to set the Endor Labs Patch Factory as the top priority package repository in your package manager (see [using Endor Patches with a single project](#) above) or artifact repository (see [integrating with your artifact repository](#) below).

02 Enable auto patching in Endor Labs

Once you have configured your package manager or artifact repository, you can enable auto patching in the Endor Labs dashboard. Within your tenant, navigate to **Settings > System Settings > Endor Patches**. Click **Enable Auto Patching** and then click **Save Settings**.

It may take up to 10 minutes for enabling or disabling auto patching to take effect. During this period, changes to your patch settings might not be immediately applied.



03 Rescan your projects

Rescan the project to update the inventory and associate the findings with Endor Patches. The patches will be applied the next time the project is built.

Integrating with your artifact repository

If your organization has thousands of repositories, you may not be able to configure the package manager for each one to work with the Endor Labs patches. Instead, you can integrate the Endor Labs Patch Factory with your internal artifact repository like JFrog or Nexus. This ensures that Endor Patches are correctly applied during the build process. **This is the simplest, one-time setup option.** This approach works also well with the **automatic patching** method described above and takes just a few steps:

- ◆ **Create a remote repository** – Follow instructions in the docs to add the Endor Labs Patch Factor as a remote repository in [JFrog](#) or [Nexus](#).
- ◆ **Prioritize Endor Patches** – Ensure the Endor Labs Patch Factory is the highest priority repository so it can replace vulnerable versions.
- ◆ **Enable automatic patching** – Following the instructions above to [configure automatic patching](#) within the Endor Labs dashboard.

Once implemented, you can automatically apply a patch from the Endor Labs dashboard and the patch will be applied on the next application build. If you scan project again the vulnerability will be resolved.

Reporting in your software bill of materials (SBOM)

Endor Patches are reported in your software bill of materials (SBOM) so you can show that the vulnerabilities have been resolved. We always show the following in your SBOM:

- ◆ The **lineage of the patch** is indicated in the pedigree section and shows the open source package that was patched (e.g. `jackson-databind@2.9.10.3`).
- ◆ The **version** follows your manifest file and will show either `-endor-latest` or the pinned version (e.g. `jackson-databind@2.9.10.3-endor-2024-07-10`)
- ◆ The **resolved vulnerabilities** are indicated in the resolves section.
- ◆ The **license of the patch** is reported and follows the upstream open source project.
- ◆ The **content** of the patch itself is added in base64 encoding.

[illegible]

Build, testing, and licensing practices

In security, trust is crucial. Therefore, the build details of each Endor Patch are fully transparent. You can audit the exact code changes, builds, build steps, and logs. All builds of Endor Patches are **hermetic** and **reproducible**:

- ◆ **A hermetic build** is when all transitive build steps, sources, and dependencies were fully declared up front with immutable references, and the build steps ran with no network access.
- ◆ **A reproducible build** is when re-running the build steps with identical input artifacts results in bit-for-bit identical output.

You can find details about each patch in the Endor Labs dashboard, or via the CLI, including the code changes in the patch, vulnerabilities addressed, the code source to reproduce the build and the build, test, and deployment commands and logs.

The screenshot displays the Endor Labs dashboard for a project named 'endorlabs/vuln-spring-boot'. The interface includes a sidebar with navigation links such as Dashboard, Projects, My Packages, Dependencies, Findings, SBOM Hub, and various tools and settings. The main content area shows a table of findings with columns for Dependency, Affected Package, Current Version, Endor Patch Available, and Fixed Vulnerabilities. A specific finding for 'com.fasterxml.jackson.core:jackson-databind' is highlighted, showing a remediation risk of 'Low' and a recommended patch version of '2.9.10.3-endor-2024-09-25'. The right sidebar provides a detailed view of the selected patch, including its instructions, overview, patches, potential conflicts, and breaking changes. It also shows a diff of the code changes, highlighting the addition of a new patch version and the removal of an old one.

Testing for security and compatibility

Endor Patches consumers need to verify that patches fix the vulnerability without introducing any regressions or breaking changes into their application. We meet this requirement by running a comprehensive test suite (including the unit tests and integration tests of the dependencies being fixed) before shipping a patch, including both manual and automated quality gates.

When possible, we adopt and run the unit and integration tests from the upstream open source project for all patches. If no such tests are available, or if we had to develop a custom fix, we decide on a case-by-case basis whether to develop a custom test case. This decision is primarily driven by the complexity of the patch, and whether it deviates from the original fix.

The actual patch, build and test commands, as well as other process input and output, are made available to customers. You not only get visibility into the patch, build, and tests performed by Endor Labs – but can reproduce and re-run the very same steps yourself.

How we test to prevent breaking changes

Whenever you upgrade a software library from one version to another—say from version 1 to version 2—the first concern is that changes in the new version could cause parts of your application to stop working as expected. After we identify the security fix we plan to backport, we use [upgrade impact analysis](#) to identify potential problem areas by comparing the public interfaces of the two versions and flagging the differences.

These differences are what we call breaking change candidates. They are changes in the new version that could cause your application to malfunction. From a business perspective, the risk here is clear:

if the upgrade breaks a critical part of your system, it could lead to downtime, security vulnerabilities, or lost productivity.

When we patch a library we identify and review breaking change candidates between the original vulnerable version of a library and the patched one. We then manually review each breaking change candidate to make sure it doesn't introduce any API incompatibilities or other behavioral changes that could impact your application. If any issues are found, these must be reviewed and addressed before the patch is created.

Requesting new patches

Our research indicates that a small number of open source packages disproportionately impact most customers, and we've prioritized building patches for these open source projects first. We similarly prioritize patches for critical or high-severity vulnerabilities in common open source packages.

If you need a new patch to address a vulnerability in your software, you can ask Endor Labs to create a new patch for you. To request a new patch, you can email support@endor.ai or request a patch directly from the Endor Labs dashboard.

Defining service level objectives

At this time we do not provide a service level agreement (SLA) for Endor Patches. We do, however, seek to be responsive to new requests and emerging vulnerabilities. We provide a service level objective for the following situations:

- ◆ **Request a patch** – You request a new patch for a library or package not previously covered by Endor Patches.
- ◆ **New vulnerability** – A new vulnerability is identified in an open source package or library that we have already patched

In both cases **our target service level objective (SLO) to deliver a new patch is two weeks**. Actual delivery times can vary, however, depending on the complexity of the changes.

Understanding licensing

Endor Patches inherit the same license as the upstream open source projects they patch. For example, the [Spring open source project](#) uses the Apache License 2.0. If you used an Endor Patch for it, the patch would also use the Apache License 2.0. This way you can use Endor Patches without worrying about license changes within your project.

Leaving Endor Labs

You may choose to leave Endor Labs at some point in the future. You should understand how that might impact any applications using Endor Patches. Most importantly, **your applications will continue working even if your contract with Endor Labs expires**. Any changes will occur on the next build of your application:



If you are using automatic patching:

- You will revert back to the original open source version.
- Patched vulnerabilities will return and will need to be fixed.



If you are using endor-latest or a pinned version in your manifest file:

- You will need to update your manifest file to remove Endor Patches.
- If you don't update your manifest file, the next build will fail.
- Patched vulnerabilities will return and will need to be fixed.

In both situations there should be **no regressions at the application level** because Endor Patches backport security fixes in a way that minimizes the introduction of breaking changes. Your engineering teams, however, will need to plan work to remediate vulnerabilities that are no longer patched.

Conclusion

Addressing risk in open source software is a balancing act between staying up to date with security patches and ensuring your applications continue to function with interruption. Endor Labs helps you strike the right balance using [upgrade impact analysis](#) to identify easy-to-fix updates developers can apply now, and Endor Patches to avoid time-intensive upgrades.

This streamlined approach allows developers to focus on building and delivering value without the added burden of complex upgrades. In fact, customers using Endor Labs typically experience a **70-80% reduction in their remediation workloads** and can **remediate vulnerabilities 6.2X faster**.

One financial services customer found they could **remediate 35,000 critical and high vulnerabilities using Endor Patches**. In most organizations, a few open-source packages account for the majority of vulnerabilities—and this case was no different. By applying just nine Endor Patches, they could achieve 98.35% coverage of all critical and high vulnerabilities.

Patch Count vs Cumulative Critical and High Vulnerability Coverage



 Secure **everything**
your code depends on.

To learn more about how Endor Patches can support your security efforts, request a demo at endorlabs.com/demo-request.