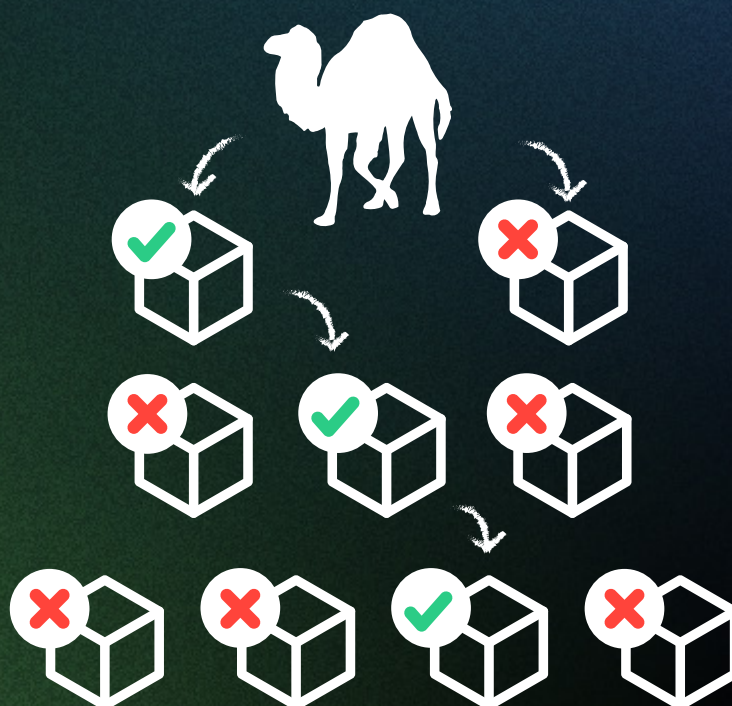


Perl Package Management Guide for Enterprise Developers



Executive Summary

Package management continues to evolve, but traditional Perl package managers are slow to catch up. Enterprise developers must deal with the consequences, including:

- **Poor Environment Reproducibility** – slightly different configurations across environments result in “works on my machine” issues and time wasted reproducing bugs, delaying time to market.
- **Supply Chain Security** – installing unsigned binaries with package managers is convenient, but risky. On the other hand, building packages from source for multiple operating systems is painful, especially if they require linked C libraries.
- **Choosing the Right Packages/ Versions** – how can you be sure you are always choosing the correct, approved open source components and versions required by your organization?
- **Finding & Fixing Vulnerabilities** – investigating vulnerabilities, patching/updating components and rebuilding environments are time and resource intensive, leaving less time for coding.

The ActiveState Platform addresses these issues, helping Perl development teams in enterprises to:

- Create consistent, reproducible Perl environments that can be deployed to all systems with a single command.
- Automatically build Perl environments from source code, resolving dependencies and packaging the result for all popular OS's.
- Always know which components/versions are approved for use.
- Identify vulnerable components, fix them, and automatically rebuild secure environments quickly and easily.

If you wrestle with any of these issues, adopting the ActiveState Platform will allow you to spend more time coding and less time managing packages and environments. All of which means you're more likely to complete your sprint deliverables on time.

The State of Package Management

As a Perl programmer, you know that package management has been a work in progress for decades. Defined narrowly, package management is the ability to install, configure, upgrade and uninstall a package/module and its dependencies. In practice however, package management is more broadly concerned with managing the development environment created by installing multiple packages against a specific version of a programming language on a specific version of an Operating System (OS).

Modern package management is concerned with solving the dependency and environment management issues that arise from this combination of components, languages and OS's that original package managers were never designed to deal with, including:

- **Multiple Environments** – how can I work with multiple projects on my local system if each requires a different version of the language and/or different versions of packages?
- **Dependency Conflicts** – if one package requires version X of dependency Z, but another package requires version Y, how can the conflict be resolved?
- **Reproducibility** – how can I ensure that my project can be consistently deployed and run on other systems?

PERL PACKAGE MANAGEMENT

For more than 20 years, the Perl ecosystem has featured a command line tool called CPAN, which was distributed with the Perl core and provided a functional way to download and install CPAN packages and their dependencies within the cpan shell. CPANPLUS was later introduced in the Perl core as an ambitious, full-featured alternative to CPAN starting with Perl 5.10.0, but never achieved its lofty goals and was subsequently removed in v5.20.0. Today, the most widely recommended package management tool is a third party module called cpanminus, which is a “zero configuration” tool that doesn't require a separate shell like CPAN.

Dependency management functionality is provided by numerous third-party libraries, such as carton, which tracks Perl module dependencies in order to ensure your project can be packaged for deployment in a consistent way. Environment management is addressed by virtual environment creation and management tools like plenv and perlbrew (or berrybrew on Windows OS).

There are a number of alternative package managers available from other ecosystems, as well, such as apt and yum for Linux distributions. Linux package management tools are capable of managing packages and environments, as well as resolving dependencies. However, they do not natively support virtual environments, so you'll have to rely on one of the solutions listed above.

Given the strengths and weaknesses of traditional package managers, seasoned Perl developers often gravitate to their favourite sets of tools to help manage their environments and dependencies. However, when it comes to issues like dependency conflicts, fixing vulnerabilities, or troubleshooting “works on my machine” issues, today's package managers leave developers to manually implement their own workarounds.

Why Modern Perl Package Management is Needed

Developers have worked around the shortcomings of Perl package management tools for decades. They've also found creative ways to better create and manage each of their development and CI/CD environments, as well. However, as organizations have adopted agile software development processes, the pressure to deliver code faster has increased, making creative workarounds for common package and environment management shortcomings less and less viable.

ActiveState is no stranger to this pressure. We handcrafted our ActivePerl distribution, which contains the latest version of Perl and hundreds of popular packages, for decades. But depending on how drastically each version of Perl, key packages, compilers, and patches changed between releases, the process could take weeks to months. To speed things up, we built the ActiveState Platform, which automates everything from dependency resolution to compiling linked C libraries to packaging the environment for Windows, Linux and macOS. The process now takes days, most of which is manual verification.

We've made the ActiveState Platform free for use so that Perl developers can use it in combination with the ActiveState Platform's command line interface (CLI), the State Tool, to manage packages, environments and dependencies in a standard, reproducible way across Windows, Linux and macOS.

This section discusses a mix of traditional and evolving use cases that are either not addressed, or else poorly addressed by traditional package management solutions. The ActiveState Platform has been specifically designed to address these gaps.

DEPENDENCY RESOLUTION & CONFLICTS

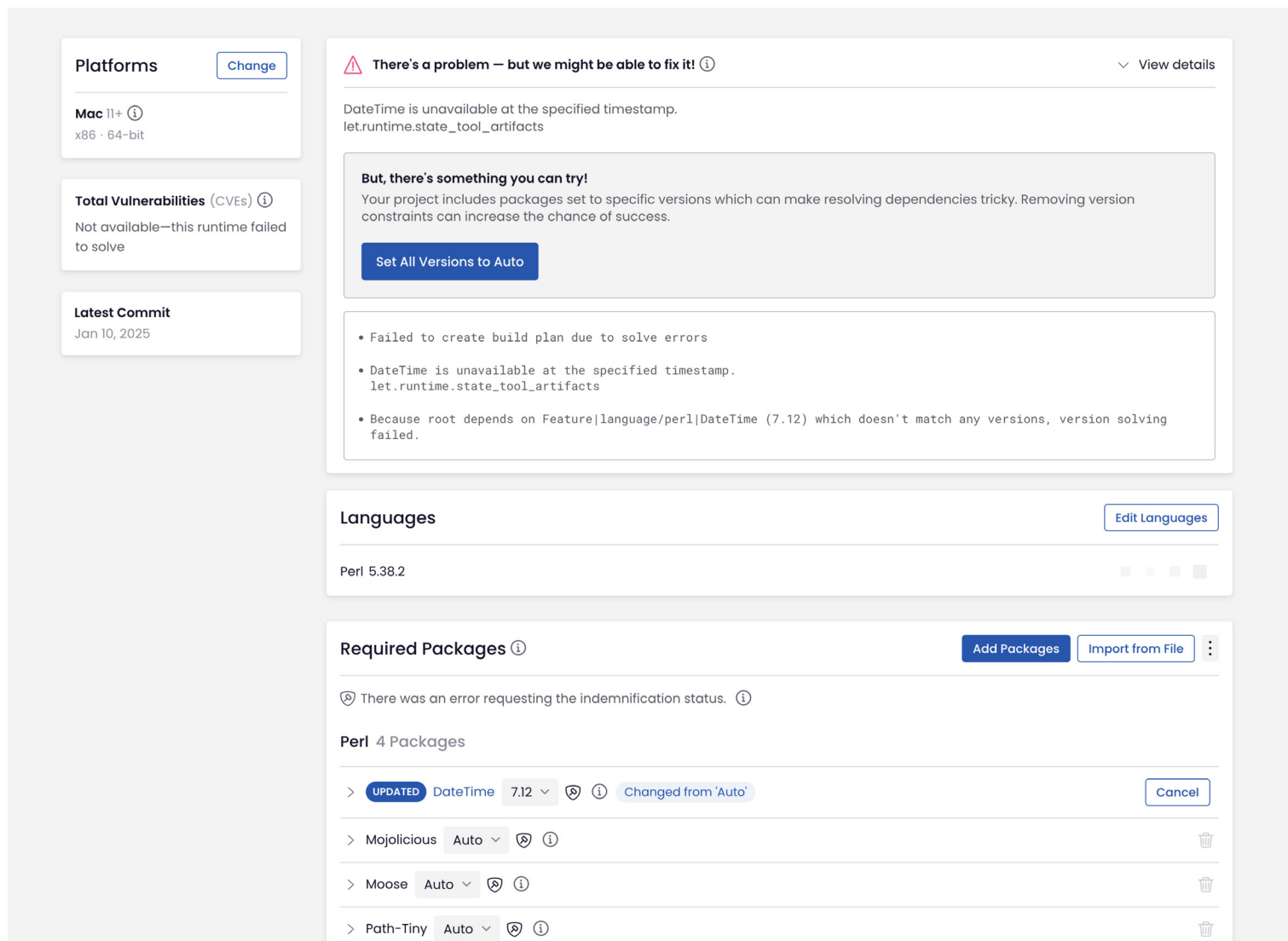
While some package management solutions will resolve dependencies, and even flag conflicts, most are incapable of resolving a dependency conflict. This results in developers manually testing various versions of packages to see if they can resolve the conflict themselves. In some cases, the solution is fairly straightforward, but in other cases developers waste time and resources in dependency hell.

The ActiveState Platform includes a dependency solver that not only resolves dependencies but also flags conflicts at multiple levels, including:

- Top level package dependencies
- Linked C library dependencies
- OS-level dependencies
- Shared libraries across multiple languages (ie., openssl)

Most package managers only resolve dependencies at the language level (ie., package dependencies), which can lead to problems when your project is deployed on a different operating system, for example.

The ActiveState Platform is also unique in suggesting ways to solve a dependency conflict if it is unable to resolve the issue automatically. For example:



The screenshot shows the ActiveState Platform interface with a sidebar on the left and a main content area. The sidebar includes sections for 'Platforms' (Mac 11+, x86 - 64-bit), 'Total Vulnerabilities' (Not available), and 'Latest Commit' (Jan 10, 2025). The main content area features a 'There's a problem' notification with a warning icon and a 'View details' link. The notification text states: 'DateTime is unavailable at the specified timestamp. let.runtime.state_tool_artifacts'. Below this, a box titled 'But, there's something you can try!' provides advice: 'Your project includes packages set to specific versions which can make resolving dependencies tricky. Removing version constraints can increase the chance of success.' A 'Set All Versions to Auto' button is present. A list of error messages follows: 'Failed to create build plan due to solve errors', 'DateTime is unavailable at the specified timestamp. let.runtime.state_tool_artifacts', and 'Because root depends on Feature|language/perl|DateTime (7.12) which doesn't match any versions, version solving failed.' Below the notification, the 'Languages' section shows 'Perl 5.38.2' with an 'Edit Languages' button. The 'Required Packages' section includes 'Add Packages' and 'Import from File' buttons. It displays a message: 'There was an error requesting the indemnification status.' and lists 'Perl 4 Packages': 'DateTime 7.12' (UPDATED, Changed from 'Auto'), 'Mojolicious Auto', 'Moose Auto', and 'Path-Tiny Auto'. Each package has a version dropdown, a shield icon, an information icon, and a trash icon.

Simply following the instructions and changing the version of Mojolicious to be ≥ 8 (ie., by clicking on the dropdown beside Mojolicious and selecting a more recent version) would solve this conflict, eliminating dependency hell.

SUPPLY CHAIN SECURITY

Most developers prefer to install Perl packages as binaries offered by the community, even though they haven't been signed. If you're careful to avoid typo-squatted packages, the risk of being compromised is typically acceptable in most organizations. After all, the alternative would be building every package and dependency from source for every operating system required, which can be a huge time sink over the life of a project. But what if there was a way to automate builds from source code?

Most pure Perl packages are trivially easy to build from source. The problem arises when a module has a dependency on a linked C library. In this case, you'll need to source, create and maintain a build environment for your operating system, featuring:

- C compiler
- Build scripts
- Installer/packager

But you may also have to manually:

- Download Ingredients – data files containing the all the metadata (version, requirements, build/install commands, etc) for each component to be built
- Correct all metadata errors for each component to be built
- Patch any known vulnerabilities
- Resolve all conflicts between components and their dependencies, as well as any OS-level dependencies

Finally, you can now compile and resolve the inevitable issues.

By contrast, the ActiveState Platform provides a cloud-based build farm that will automatically build Perl packages (as well as their dependencies) from source code in parallel, including any linked C libraries, and then package them for Windows and Linux. As a result, there's no need to maintain a local build environment, or even a need for language or operating system expertise. While not every package/module can be automatically built at any point in time (ie., newer versions may introduce a new build method), the ActiveState Platform often provides the simplest way for developers to build their environment from source.

ENVIRONMENT REPRODUCIBILITY

Ensuring that your project can be deployed in a consistent, reproducible manner is a key goal for any software development team.

Perl has CPANfile and META.json files to help specify the exact set of dependencies your project requires. These files go a long way to ensuring consistent deployability, given a specific version of the programming language

Of course, you need to remember to update these files before deploying, but there are other issues, as well:

- CPANfile and META.json files can get out of sync as different developers on a team update their development environment for their own purposes.
- Even when all development team members are using identical Docker images or Virtual Machines (VMs) for their development environments, you still need to ensure that they have been built with the latest Perl environment, and that all development environments are up to date.
- When developing on one OS but deploying on a different OS, you may be missing OS-level dependencies.
- DevOps is often left to resolve multiple conflicting code check-ins, leading to CI/CD environments that differ from development environments

All of these issues are likely to lead to environment inconsistencies. The result can be bugs found in the CI/CD process that developers need to spend time reproducing by rebuilding the CI/CD environment where the bug was found.

The ActiveState Platform takes a different approach to ensuring environment consistency and reproducibility by:

- Automatically building your Perl environment for all major operating systems: Windows and Linux.
- Providing a central “source of truth” for the Perl runtime environment that all developers (and DevOps) can pull to build their local environments, ensuring everyone is using the same environment, no matter their OS.
- Updates to the environment are made centrally, and can be updated locally with a single command.

Centrally managing and deploying your Perl environments means that development and DevOps teams remain in sync, eliminating “works on my machine” issues.

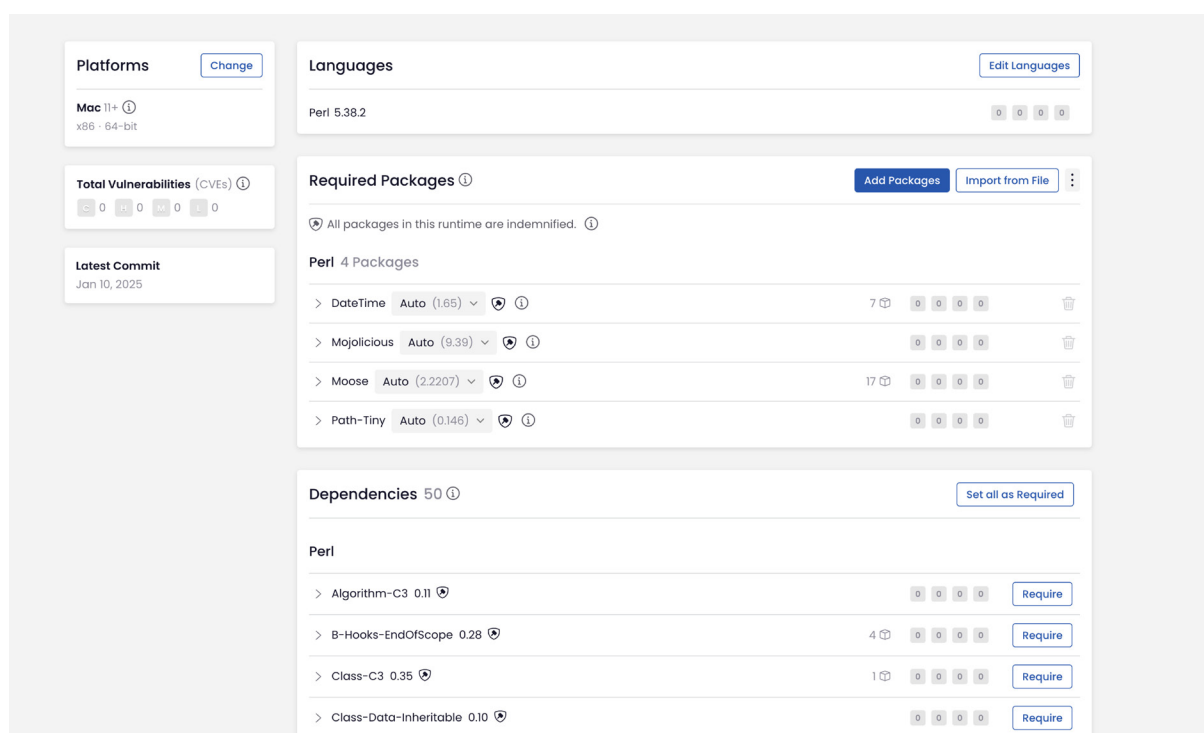
CHOOSING THE RIGHT PACKAGES

While not traditionally considered a package management issue, one of the most common problems we hear from customers is, “How can I ensure my developers are using only the approved set of components/component versions?”

Currently, most enterprises either:

- Use a local “walled garden” repository of packages, which can severely limit developers that want to experiment with new packages and/or new versions since they are simply not available. Typically, the approval process for adding new components to the walled garden can be quite lengthy and/or complex.
- Manually maintain a list of approved packages/versions, which can quickly get out of date.
- Use a third party tool, which can help restrict use of packages that feature unapproved licenses and/or those that have a vulnerability. However, developers can still gain access to unapproved packages that meet these criteria.

The ActiveState Platform offers an unrestricted solution whereby developers gain access to all packages in our catalog (which is updated from CPAN, GitHub, and other sources on a regular basis), but are provided guidance as to which components/versions are appropriate for use through our indemnification offering. Indemnified packages are well maintained, vulnerability free, and feature licenses appropriate for creating commercial offerings.



The screenshot displays the ActiveState Platform interface for managing packages. It includes several sections:

- Platforms:** Shows 'Mac 11+' and 'x86_64-bit' with a 'Change' button.
- Languages:** Shows 'Perl 5.38.2' with an 'Edit Languages' button.
- Total Vulnerabilities (CVEs):** A bar chart showing 0 vulnerabilities across different categories.
- Latest Commit:** Shows 'Jan 10, 2025'.
- Required Packages:** A section titled 'Perl 4 Packages' listing:
 - DateTime:** Auto (1.65) with 7 dependencies.
 - Mojolicious:** Auto (9.39) with 0 dependencies.
 - Moose:** Auto (2.2207) with 17 dependencies.
 - Path-Tiny:** Auto (0.146) with 0 dependencies.
- Dependencies:** A section titled 'Perl' listing:
 - Algorithm-C3:** 0.11 with 0 dependencies, marked as 'Require'.
 - B-Hooks-EndOfScope:** 0.28 with 4 dependencies, marked as 'Require'.
 - Class-C3:** 0.35 with 1 dependency, marked as 'Require'.
 - Class-Data-Inheritable:** 0.10 with 0 dependencies, marked as 'Require'.

FINDING AND FIXING VULNERABILITIES

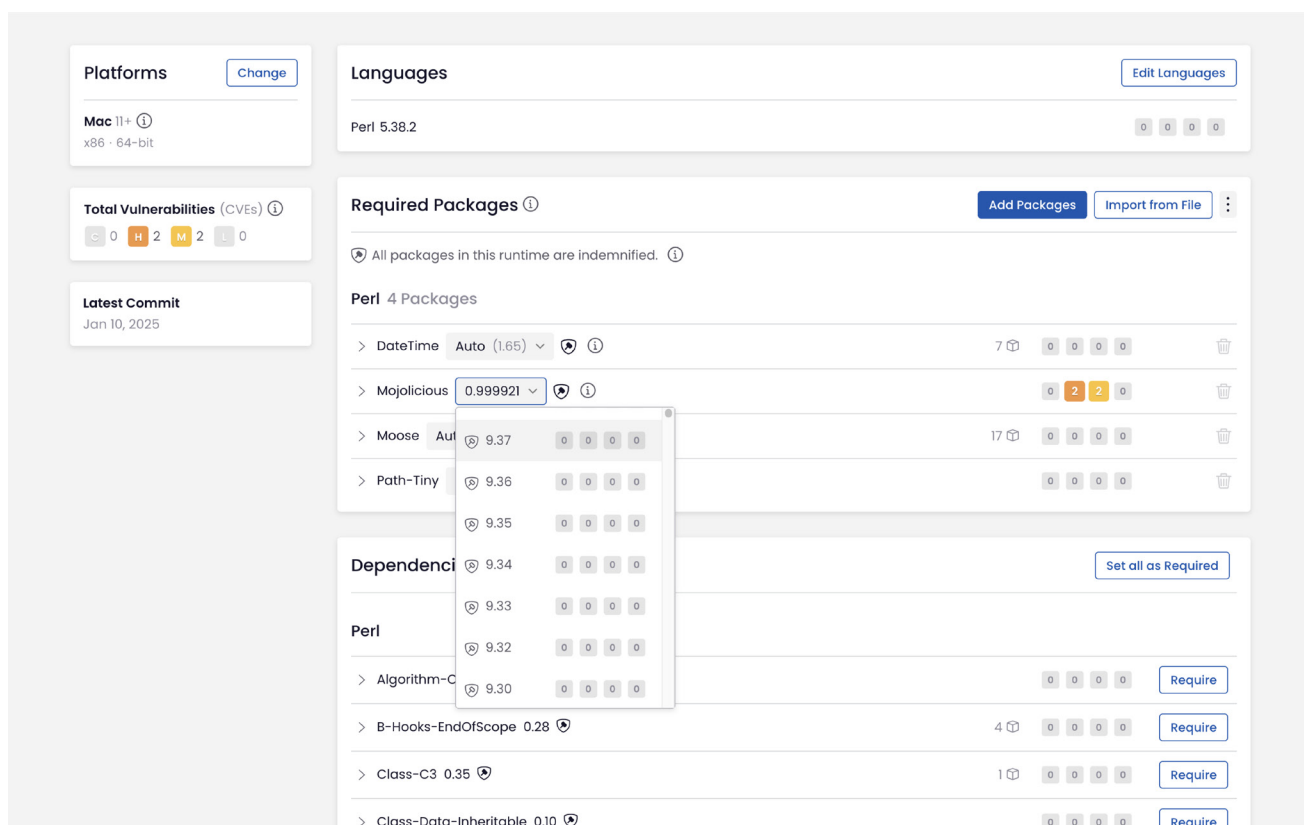
Again, although fixing package vulnerabilities are not typically considered a key requirement of package management, securing Perl environments is a common use case that currently requires an inordinate amount of time and resources to:

- Discover the Common Vulnerabilities and Exposures (CVEs)
- Investigate the impact
- Patch/upgrade/downgrade the affected package/module
- Rebuild the environment
- Retest

While there are a number of solutions that can notify developers when a vulnerability is discovered, and even suggest a solution, all the other steps remain manual tasks. It's no wonder that the Mean Time To Resolution (MTTR) for these kinds of issues is weeks instead of days or hours.

The ActiveState Platform helps automate a number of the time consuming tasks associated with finding and fixing vulnerabilities, including:

- Automated notifications
- A list of non-vulnerable packages versions that you can simply point and click to upgrade or downgrade to
- Visibly showing the cascading effect on all other dependencies when you upgrade/downgrade a package
- Automatically rebuilding and testing the environment for you



As a result, developers can spend less time finding and fixing vulnerabilities, and enterprises can dramatically decrease MTTR.

How ActiveState Can Help

The ActiveState Platform takes a holistic approach to package management, providing developers with a single, unified, cloud-based toolchain that works for both Python and Perl on Windows and Linux.

By adopting the ActiveState Platform, enterprise developers can benefit from many of the same advantages, including:

- Automated building of packages from source, including link C libraries without the need for a local build environment.
- Automated resolution of dependencies (or suggestions on how to manually resolve conflicts), ensuring that your environment always contains a set of known good dependencies that work together.
- Central management of a single source of truth for your environment that can be deployed with a single command to all development and CI/CD environments, ensuring consistent reproducibility. Automated installation of virtual Perl environments on Windows or Linux without requiring prior setup.
- The ability to find, fix and automatically rebuild vulnerable environments, thereby enhancing security and dramatically reducing time and effort involved in resolving CVEs.
- Visually seeing which versions of which packages are approved for use, thereby taking the guesswork out of development.

Those that prefer to work from the command line can leverage the ActiveState Platform's command line interface (CLI), the State Tool, which acts as a universal package manager for Perl, and provides access to most of the features offered by the Platform.

Conclusions

Most enterprise developers manage and maintain multiple package managers, environment management tools and other solutions in order to address the issues that a single tool, the ActiveState Platform can solve today. By adopting the ActiveState Platform, developers can:

- Increase the security of Perl environments
- Improve the transparency of your open source supply chain
- Eliminate dependency hell
- Reduce “works on my machine” issues

Ultimately, developers that are willing to adopt the ActiveState Platform will spend less time wrestling with tooling and more time focused on doing what they do best: coding.

To try the ActiveState Platform for yourself, sign up for a free account at <https://platform.activestate.com>.



About ActiveState

ActiveState enables DevOps, InfoSec, and Development teams to improve their security posture while simultaneously increasing productivity and innovation to deliver secure applications faster.

With a single platform that tames open source complexity, teams get a continuously secure software supply chain, unparalleled observability, robust vulnerability management, continuous upgrades, and governance support that enhance collaboration across the organization.

All from the trusted partner that pioneered and continues to lead enterprise adoption and use of open source software.

Start An Enterprise Trial