ActiveState

TECH DEBT BEST PRACTICES: MINIMIZING OPPORTUNITY COST & SECURITY RISK

Executive Summary

Technical debt, aka “tech debt,” is the unavoidable consequence of the fast paced software industry, which results in the need to rework a solution and/or refactor code on a regular basis.

Cybersecurity bad habits are a key driver of tech debt, specifically security tech debt, which arises primarily as a result of:

- Downloading prebuilt, compromised open source components, necessitating remediation.
- Creating new projects with vulnerable codebases, necessitating updates.
- Continuing to work with outdated and/or EOL codebases, necessitating upgrades.

Left unaddressed, all of these pose a growing security risk to both the software vendor and their customers. Previously, vendors could get away with effectively externalizing that risk to their customers. Today, worldwide legislation aimed at enforcing cybersecurity and software supply chain best practices have resulted in litigation being brought against companies and key employees for exposing their customers to undue risk.

Security tech debt is unavoidable as software inevitably becomes more vulnerable over time, but the costs of addressing it can be contained. This white paper provides cost-effective solutions to managing tech debt from the beginning of a project, throughout its lifecycle, and even beyond End of Life (EOL).

Introduction

When it comes to software development in general and open source security in particular, a number of bad habits have become ingrained over the decades to where the industry may no longer even be aware of them. The best example may be downloading prebuilt components from open source repositories despite the fact that:

- No details are provided about how the component was built, or from where its source code originated.
- Few organizations have a relationship with the authors of the open source components they use, requiring blind trust. Worse, most components include multiple dependencies, each of whose authors must also be blindly trusted.
- Precompiled binary components are difficult to scan in order to ensure they haven't been compromised.

But installing prebuilt components is quick, easy and convenient, and eliminates the time and resources required to build them from source.

Similarly, there are two other open source bad habits that are widespread across the industry:

- Creating vulnerable codebases
  - » As Sonatype has reported, 2.1B open source downloads with known vulnerabilities in 2023 could have been avoided because a better, fixed version was available – the exact same percentage as in 2022.
- Working with outdated codebases
  - » As Veracode continues to report year over year, 79% of codebases are rarely updated once created.

The US government has been encouraging the software industry to enhance its cybersecurity capabilities ever since President Biden issued Executive Order 14028 in early 2021 in response to the Solarwinds incident. The incentive was access to lucrative US government agency contracts (DoE, DoD, DHS, etc). But even third-parties who aligned their products with the government's security requirements could benefit by becoming suppliers to government contractors.

However, the spectacular growth in online attacks over 2023 shows this approach has largely failed, resulting in growing emphasis on prosecution and the issuing of $2.2B in fines to date. The US government is not alone: governments around the world are taking a similar approach. In other words, lacking proper security controls in 2024 may be a legal and economic threat to the survival of your business.

The Risks Of Security Tech Debt

All modern software projects make use of open source software. In most cases, >80% of an application's codebase is open source, typically downloaded prebuilt and ready to use. But with the meteoric rise in software supply chain attacks over the past few years, the risk posed by prebuilt open source components is rapidly escalating:

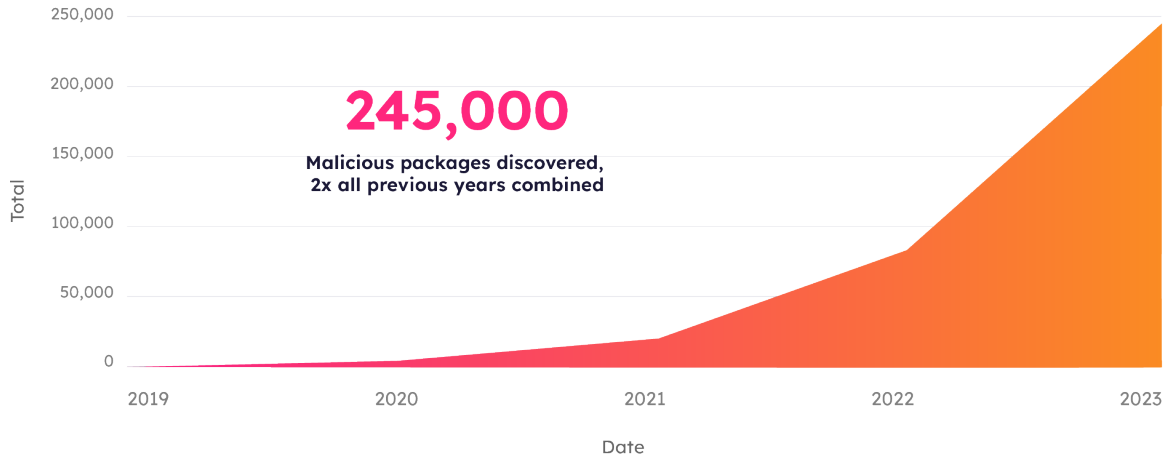


Figure 1:
Software supply chain attacks over time. Source = Sonatype

To counter the risk of downloading a malicious component from an open source repository organizations should consider building all third party dependencies from source code in a repeatable, reproducible manner, because:

> If you can't effectively manage and ensure the security of your open source and third-party software, no other efforts made toward securing your supply chain will work — or frankly, even matter.
>
> – Synopsys OSSRA 2023 Report

But with the exception of large enterprises in security-conscious industries, most organizations build only a few key components (such as OpenSSL) from scratch. As such, setting up a declarative, reproducible build system capable of building third-party components in a reproducible manner is simply not cost effective.

With the rest of the codebase assembled from prebuilt components, it's left to processes and tools later in the software development lifecycle to catch compromised dependencies, at which point the cost to update or remediate the codebase escalates.

The Opportunity Cost of Vulnerability Remediation

There's no denying that vulnerabilities continue to plague the industry. High risk vulnerabilities in particular have increased by at least 42% across all industry sectors since 2019[1]. Worse, most organizations continue to struggle with remediating vulnerabilities:

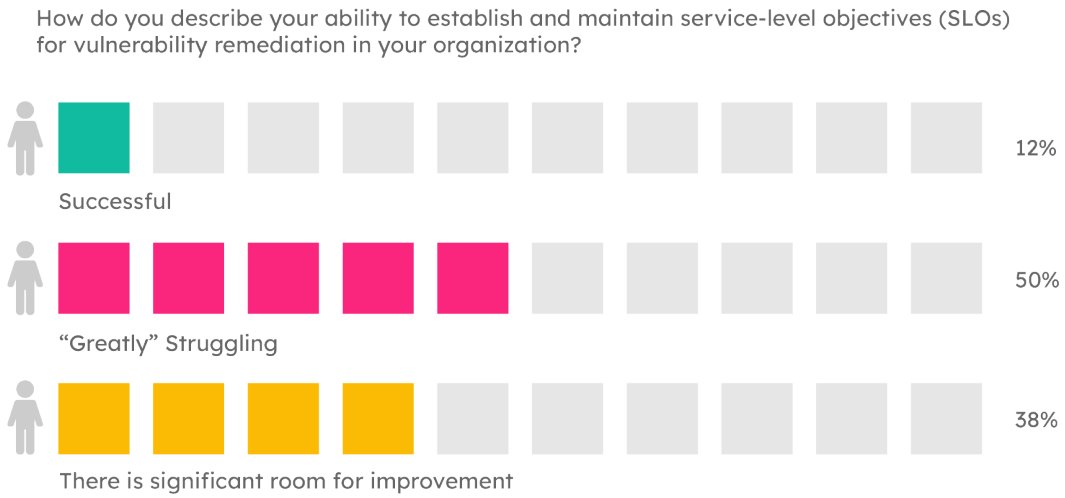


Figure 2:
Vulnerability remediation gap. Source: slim.ai

The result is growing technical debt in the form of vulnerability backlogs that can result in everything from hampering innovation to disrupting application performance to violating uptime service levels.

While there is no silver bullet to the vulnerability problem, there should be no reason why developers can't at least ensure projects start with a secure codebase. However, in both 2022 and 2023, 2.1B open source components with known vulnerabilities were downloaded from public repositories despite the fact that a fixed version was available.[2]

In many cases, this result is due to outdated components that can't easily be updated, removed or substituted. In other cases, it's simply because developers install their environment from a manifest file (i.e, Python requirements.txt, Ruby gemfile.lock, Perl meta.json, etc) that provides no vulnerability information.

While best practices such as minimizing the size of the codebase and using a centralized collaboration platform to share vulnerability information can help, the base problem still exists: the opportunity cost of remediating all but the most critical/exploitable vulnerabilities is simply too high. There is always a better use for an organization's time and resources. However, continuing to ignore vulnerabilities will eventually result in a non-performant, unsecure codebase that will need to be updated despite the opportunity cost.

[1] Synopsys' Open Source Security and Risk Analysis (OSSRA) Report 2023
[2] Sonatype 9th Annual State of the Software Supply Chain

The Opportunity Cost of Codebase Updates

Research by Veracode shows that most organizations prefer to limit the number of times they update or upgrade their codebase:

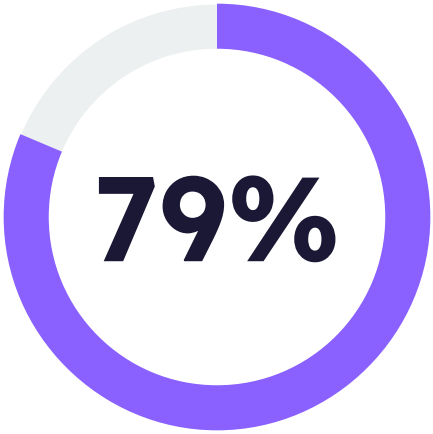
> When developers add an open source library to their application, 79% of the time they never go back to update it.
>
> – Chris Eng, Chief Research Officer at Veracode

There are any number of reasons why organizations are reluctant to make changes to their open source codebase, but they all come back to opportunity cost since updates can result in:

- Pulling in new components or new versions of existing components that can result in dependency conflicts and/or corrupted environments.
- Breaking the build since new components/new versions may not build correctly.
- Refactoring proprietary code since new versions of components may introduce breaking changes to their functionality.
- Redeployment issues since the average time to deploy an update for an existing application is 65 days, during which time another update may become necessary resulting in more tech debt in the form of an update backlog.

Open source libraries are constantly evolving; what appears secure today may not be tomorrow.

Despite this dynamic landscape, 79 percent of the time, developers never update third-party libraries after including them in a codebase.



Source: Veracode State of Software Security

Again, time and resources spent on any of these issues is almost always better spent elsewhere. But ignoring updates means that eventually the codebase will become EOL, forcing an upgrade process that can be far more complex and resource intensive than applying ongoing updates:

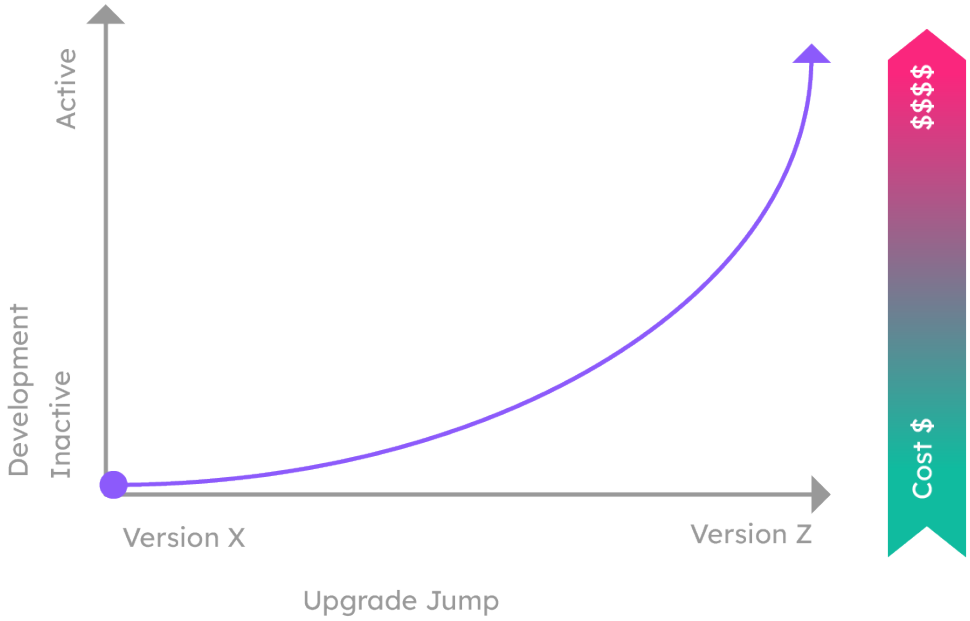


Figure 3:
Cost of upgrading an older codebase

As figure 3 shows, the further the upgrade target is from the original codebase, the more expensive the upgrade will be. This is primarily due to the fact that while a codebase's components change minimally between patch versions for any open source language, they will typically introduce breaking changes between minor versions of the language (to say nothing of differences between major versions), meaning that a greater portion of application code will need to be rewritten/refactored.

But it doesn't need to be like this.

Minimizing the Opportunity Cost of Tech Debt

Security-conscious organizations have long dedicated 10-30% of every sprint to addressing technical debt, which includes updating and/or upgrading aging and vulnerable codebases. But as new projects proliferate, it's often difficult to split a team's focus between maintenance and new development work.

Larger organizations may be able to create dedicated maintenance teams who are measured by their ability to remediate aging codebases using best practices that include:

- An automated Continuous Integration (CI) system for both the existing version and the target upgrade/update version.
- Adequate automated test coverage.
- A strategy to minimize the codebase by continually removing redundant, outdated or unused components.
- An incremental upgrade strategy in order to minimize the amount of broken code between language versions.

Unfortunately, smaller organizations often lack the resources to dedicate to maintenance. In these cases, a service such as the ActiveState Platform can help by providing:

- Automated builds of all open source components from vetted source code using a hardened build service to eliminate the security tech debt that can arise from working with prebuilt components.
- Automatically calculating the minimal set of components necessary for a project based on initial package requirements in order to minimize maintenance work.
- Automatically resolving dependency conflicts when upgrading the language or updating a component version.
- Automatically calculating the delta in components (which ones are new; which have been deprecated and which upgraded) between one version of a language and another, and displaying it graphically so you can better understand the effort.
- Automatically rebuilding the updated/upgraded codebase in order to minimize the opportunity cost of updates/upgrades.
- Automatically creating local virtual environments for the current codebase and upgrade target(s).

By leveraging the automation offered by the ActiveState Platform, organizations can minimize the time and resources required to perform upgrades/updates, and thus minimize opportunity cost.

ActiveState is also well versed in codebase migration, as well, having supported EOL codebases like Python 2 for a wide range of applications. If you have a complex codebase update, upgrade or migration you need help with, please contact us to learn how you can minimize the opportunity cost of getting current and staying current with your codebase.

ActiveState

Secure open source integration

www.activestate.com
Toll-free in NA: 1-866.631.4581
solutions@activestate.com