Third-party library mismanagement: How it can derail your plans

Ruchir Garg

Ruchir.Garg@Intel.com

Abstract

Third-party libraries are synonymous with most software applications as they enable faster development. While third-party libraries provide value, but if not managed properly, they could later become a major source of project delays or unhappy customers. In this paper, we discuss multiple suggestions that can help alleviate issues arising from third-party library mismanagement. Adopting these suggestions will help you mitigate these risks to a large extent, while allowing to you build a more predictable release strategy.

Biography

Ruchir Garg is a Security Architect at Intel Security, where he is working on securing enterprise class applications. He has also worked as an automation architect, developing brand new test automation frameworks. Ruchir has spent more than 14 years of industry experience developing, testing and supporting software products in various domains like embedded, mobile, wireless and desktop applications.

Ruchir holds a degree in Electrical Engineering from Nagpur University, India.

1 Introduction

Most decent sized software products use third-party libraries for faster development. The topic we'd discuss here is something that is usually given lesser importance than it deserves i.e. the management of third-party libraries. We describe processes to minimize the negative impact of using third-party libraries to your release plans. We identified and adopted best-practices that helped us manage third-party libraries in a more predictive manner. We also observed that adopting these best-practices reduced the number of hotfixes we shipped.

In this paper, we'll discuss various issues like managing different library versions, its compatibility– API or platform support, maintenance, and security vulnerability management.

2 Problem/Opportunity

Along with the obvious benefits, using third-party libraries also bring along its own share of possible challenges and complexities. When third-party libraries are not managed in an organized manner, developers use and include numerous external libraries, without giving attention to their maintenance.

This may lead to a situation where the products used by the customers include obsolete and possibly, vulnerable libraries. If a vulnerability introduced by an installed product gets exploited, not just Intellectual Property (IP) is at risk, but the organization that owns the vulnerable product may even face a litigation.

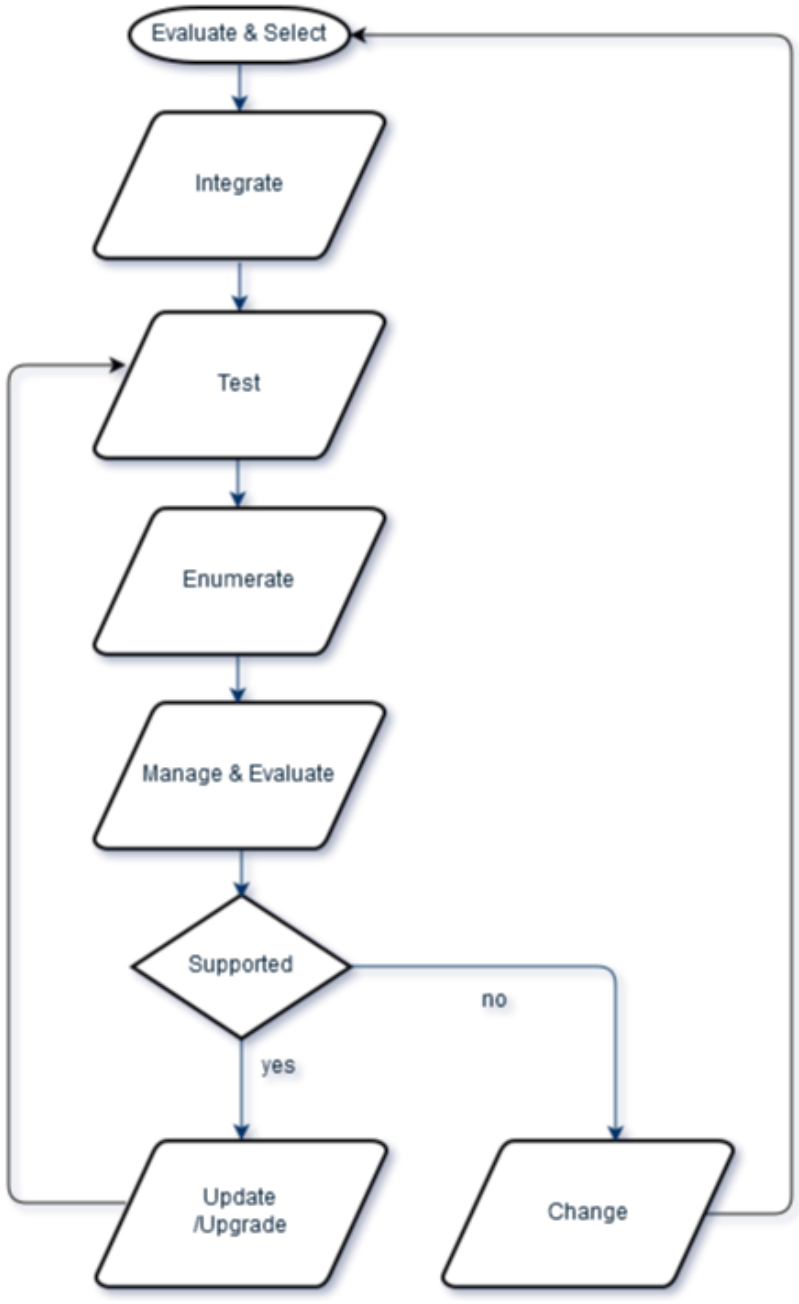
These challenges may create unwarranted situations where the ongoing product releases get affected. For example, consider a situation where a security researcher discloses a critical vulnerability in a third-party library version that you are using and your customers are demanding an immediate fix within a week. Now when you try to upgrade to the newest version of the library, which has the fix, but you realize that an important OS platform support has been dropped. This may affect a large customer base and definitely not a good news for the organization. Worse, if you lost track, the library might have gone end-of-support and the fix for the vulnerability is simply not available. This is a nightmare scenarios for any organization.

Project delays can hurt any organization, especially when faster delivery to market is the norm and completion is breathing down the neck. Delays are inherent to most software projects, but a delay due to external dependencies is something on which we've least amount of control, and has to be managed very efficiently.

The problem described here will resonate with all software professionals, who are dealing with third-party libraries in their code, as they are challenged with these issues every day. Participants can expect to learn from the best practices described in this paper and apply them to minimize the unexpected delays to their releases.

3 Solution

The diagram [1]given below represents a typical lifecycle of a third-party library:



[1] *Figure 1:- Lifecycle of a third-party library*

3.1 Libraries enumeration

The recommended process starts with building an active list of all such libraries, including the version used and the latest version available in the field. Keep an eye on the date when the library was last updated.

3.2 Vulnerability tracking

Actively search and track all security vulnerabilities published against the third-party libraries used in the project. A simple search in the National Vulnerability Database[2] or tracking the product specific security bulletins will help. This should give you a good indication of your current position. In an ideal scenario, a library should have vulnerabilities listed against it and those should have been fixed too. Finding no vulnerabilities either means not many are using it, or no one has actively researched it yet. What it definitely does ***not*** indicate that it's a secure library.

3.3 Keeping things updated

Keeping all libraries up-to-date at frequent intervals will help in eliminating the risk of a major version upgrade on a short notice. A major version upgrade may possibly introduce a risk of API changes or platform deprecation.

A frequency of once every quarter to review and update the libraries is reasonable. Any greater frequency could be an overhead and a lesser frequency poses a risk of taking in too many changes together, which may increase the quality risks. Retaining source and packages of all versions of the libraries used in your source control repository is also good practice, which helps in debugging the field issues.

3.4 Library selection

Selection process of new libraries is a big deal as various factors that need to be carefully looked into. Factors like their active maintenance schedules, platform support, number of vulnerabilities reported and fixed, EOS schedule, and third-party libraries (if any) used within. We can get the list of any third-party used within by looking up its licensing documentation.

Organization must setup a whitelisting team, which evaluates third-party libraries based on these factors and then recommends them for general use by all teams. This will ensure uniformity and compliance.

I'd reiterate that having no reported vulnerability for a library is not a sign of a secure product.

3.5 Support considerations

There could be libraries in your product that are either reaching their End-Of-Support (EOS) or are already EOS. It's strongly recommended to migrate away from such libraries to their alternatives. As such libraries will not receive any security fixes, continuing to use them pose a danger to our customers and also to our organization's reputation.

Keep an eye on the extended support fine-print to ensure that extended support is not limited to critical issues or technical assistance only, but not fixes.

3.6 Quality matters

We can ensure the quality of our code by employing rigorous validation, static analysis, vulnerability scanning, fuzzing and penetration testing. However, we have no control over the quality processes followed during the development of the third-party libraries we consume. Such libraries could be either

[2] National Vulnerability Database is a good resource to search vulnerabilities in software: https://web.nvd.nist.gov/view/vuln/search

open-source or closed source, but we should definitely apply all ***applicable*** quality checks described above to ensure that the libraries meet the minimum quality criteria. We must always remember that a chain is only as strong as its weakest link.

3.7 Management commitment

The factor of management support cannot be overlooked, if we've to make any of these changes discussed above. All these process and technical changes require time and investments, before they start to show the results. Hence, it's important that management invests in this strategy by allowing additional time in the projects for the reduction of technical debt, and prioritizes relevant stories from the backlog to enable library upgrades.

Libraries that are feature-rich and are also maintained efficiently often come at a cost, so the management must be prepared to pay for it as its going to be more cost-effective in the long run compared to a cheap, but poorly written library. Using open-source is always an option, which has its own pros and cons, but that's another discussion, outside the scope of this paper. Even for open-source libraries, all these learnings would certainly apply.

4 Results

Teams generally follow their own processes of managing third-party libraries. We also had our own process that we followed, but frequently ran into issues with that. Adopting these best-practices reduced the probability of manifestation of risks associated with external library usage to a large extent, if not completely. We don't claim that our process is the best, but we can definitely claim that it works.

We've seen that following these best practices minimized the risk of running into project risks, if not to totally avoid them. Moving away from current practices to the ones described here is not an easy task and will require significant investments. My team has realized that adopting these best-practices saved us from the problems of unplanned security hotfixes and sudden platform support deprecation. It also means that our existing projects will no longer suffer from unwanted interruptions and we are better compliant with security incident response SLAs. We can now afford to have a more predictable release cycles. This is good news for our support team and our customers, which will eventually means a higher Net Promoter Score (NPS)[3].

There are software tools available in the field that could be used for automatic patch and update management, but those are limited to only well-known applications. Until a generic product becomes available that can automate this process, library management process as described in this paper is a possible solution.

5 Final words

The changes suggested in this paper are both technical and process related. Developers need to work closely with the security architects while selecting external libraries, existing libraries need to be evaluated and risk need to be addressed. Product Managers also need to prioritize such tasks so that it gets into the release. It's a shift in the mindset and all stakeholders must support to get it addressed. Finally, this template once applied to all products in the organization will ensure process uniformity and compliance, bring efficiency into project releases, and increase customer confidence into your products.

[3] Net Promoter Score is a customer loyalty metric: https://en.wikipedia.org/wiki/Net_Promoter