LTIMindtree

Point of View

Stay Ahead of Security Vulnerabilities

in Open Source Software Components

by **Rakesh M S** and **Vengadesh Babu**

Contents

01 Introduction

The usage of open source software components in enterprise applications have grown over the years. Organizations have greater range of responsibility to protect the software components, including the external open source components that they use. Security weakness in enterprise applications poses huge risk, potentially exposes the organization's data and impacts businesses at a greater scale. But many organizations are not prepared to deal with open source software security threats. In this paper, we cover key best practices to keep the open source components secure.

1.1 Role of Open Source Software in Enterprise Application Development

Open source software plays a key role in enterprise application development today. According to Open Source Security & Risk Analysis (OSSRA) report published by Synopsys in 2020, 99% of codebases audited in 2019 contained open source components, which contributed to 70% of overall code base. There are many reasons to use open source over proprietary. Major reasons are lower cost of ownership, faster access to latest innovations, higher overall quality, and ease of customization.

As the usage increases, open source software plays a major role in the overall security posture of an enterprise application. It is widely believed that open source software is more secure than proprietary, primarily because more eyes would have seen or reviewed the code, hence there is a lesser possibility of hidden flaws. But major security vulnerabilities reported in

open source software like Heartbleed and Shellshock, which allowed attackers unprecedented access to millions of affected servers had existed many years without notice. As per Synopsys, 75% of codebases audited contain open-source components with known security vulnerabilities. It is an eye-opener that organizations continue to struggle in tracking and fixing open-source vulnerabilities effectively.

1.2 Challenges in Keeping the Open Source Components Secure

Organizations face multitude of challenges to keep open source modules secure. Some of the major issues are listed below:

- No well-defined process and policy established
- No full visibility of all the open source components and dependencies in use
- Unavailability of tools to manage open source security efficiently
- Automatic patch updates are not available
- Emphasis on feature releases over security updates

02 Suggested Practices

2.1 Enforce Solid Security Practices

Even though patch management processes are around for many years, it is questionable if organizations are following them effectively when it comes to open source software components. Following are some of the recommended practices to effectively manage security of open source software components in an application.

2.1.1 Keep Track of Open Source Components

From both security & compliance standpoint, it is critical to know all the open source components and its dependencies for an application. When a software component relies on another one to function, then it is called a software dependency. Even for a simple application, there can be multiple levels of dependencies where security vulnerabilities could be hidden. It is recommended to make an inventory of all the software modules/components and keep them updated on a regular basis.

Depending on the language/technology, we have different options. Let us consider Python, where we have a native package manager to extract the dependencies. The command 'pip freeze' will list all package dependencies

of your application. In addition to Python package manager, we have alternative tools such as pipdeptree and pipenv, which offer additional functionalities such as identifying conflicting dependencies, package dependency views, etc.

Like Python, for Javacript-based applications, we can make use of a native package manager (NPM) command 'npm ls' to extract all dependencies. There are several build automation tools like CMake, Maven, etc. can also help us in extracting all the packages and dependencies.

2.1.2 Proactive Identification of Vulnerabilities

There is no single source to refer reported vulnerabilities in an open source software. The following sources covers majority of publicly disclosed vulnerabilities:

- Public Vulnerability Databases
  - Common Vulnerabilities and Exposures (CVE)
  - National Vulnerability Database (NVD)
- Vulnerability Databases from Security Product Vendors
- Security Advisories from Software Maintainers or Community/Vendor
- Bug Trackers

There are tools available such as vulnerability scanners which help you scan the application for identifying vulnerable components against these sources. For example, 'NodeJsScan' is a tool which helps to identify known

vulnerabilities in a Node-based application. Similarly, 'RetireJS' works with JavaScript application, and 'Safety' for python-based application. There are tools which have the capability to support multiple technologies and offer end-to-end security solutions which are covered in section **2.3**.

We need to keep in mind that there is no single tool to comprehensively identify all vulnerabilities since there is no single source for tracking all vulnerabilities reported. But with a strong open source usage policy to allow only reputed open source components, we can maintain a good level of security.

2.1.3 Include Component Patching as Part of Regular Product Development Plan

Maintaining an inventory of all the open source libraries and identifying vulnerabilities is only one part of the puzzle, but keeping them up-to-date is another crucial step. Dedicated efforts should be allocated for this activity to identify components with reported vulnerabilities and apply the right patches as quickly as possible. To do so, we need to have a security-focused patch management practice tied to our regular product development plan. Once a vulnerability has been identified in an open source component, add that to the development team's backlog, it should be treated with utmost importance. Regression testing should be done after the patching to make sure that there is no impact to the existing functionality of the application.

We need to prioritize updates based on severity of patches so that critically vulnerable modules are updated sooner. It is recommended to have a formal prioritization process in place involving both security and development teams, so that a decision can be taken quickly on which patch/update needs to be applied sooner than later.

2.1.4 Replace Retired Open Source Components

It is critical to identify and replace software components, which are retired or no longer supported by vendor/community. Keep using these non-supported components will pose significant security threat. The method to identify retired modules differ based on technology/frameworks/languages. For example, in python, we can use the native package manager command 'pip list --outdated' or more advanced tools like 'pipe-check' to identify outdated modules in your environment.

We should have replacement plan for the identified outdated/retired modules. Possible options are:

A
Replace the module with latest supported version (if available)

B
Replace it with alternate components (if available)

C
Develop one in-house (if above options are not possible/feasible)

Whichever the option we choose, it would require dedicated efforts from the development team. The efforts should also include regression testing to make sure the updates not impacting the application. Continue using unsupported modules is a huge security threat. Hence, we should enforce a security policy or a checklist to block releasing software with unsupported/retired software components.

It is also vital to identify modules which are no longer in use and remove it. The presence of obsolete modules will increase the security risk. For example, let us consider a Python-based application, where keeping an

unused library reference in the requirements file will lead to installing it on a target machine. Keeping all the unused modules will bring unnecessary burden to the patch management process. There are tools available to help us in identifying unused modules like depcheck for npm javascript packages, Apache maven dependency plugin, and so on.

2.1.5 Upgrade to Latest Stable Version of a Software as Quickly as Possible

Usage of older version of software will potentially end up in scenarios where the version is reaching its end-of-support (EOS). In this scenario, we will not have enough time to migrate to latest version before EOS and leads to security risk. Hence, it is recommended to migrate to latest version of open source components as fast as possible. There are additional benefits in using latest version. For example, along with feature or functionality enhancement, the latest version of software usually contains direct security advancements or indirect security improvements due to better architecture, technology, etc.

2.1.6 Mandatory Vulnerability Assessment and Penetration Testing

Even if we have ensured that the individual application components are secure, there can be scenarios where a simple misconfiguration potentially create a security lapse. Vulnerability Assessment & Penetration Testing (VAPT) helps to uncover hidden security loopholes that exists in the entire application.

The **Vulnerability Assessment** is performed to evaluate the application to identify if it is susceptible to any known vulnerabilities. If a vulnerability is identified, then we can follow suggested solution or best practices to address them.

Penetration Testing is an ethical hacking technique, which simulates the actions of a malicious external or internal actor with an attempt to get unauthorized access to application or data.

It is recommended to enforce VAPT with the following conditions,

- For every major release of the application
- On a periodic basis. For example, every six months to uncover new types of vulnerabilities that are potentially introduced as part of the minor releases

2.2  Establish a Strong Open Source Software Usage Policy

If we do not have any mechanism to control on what open source components can be used, developers may end up using potentially vulnerable or untrusted software. The development team may not always know the level of security and community support of different open source components. There are other issues like potentially violating license terms if we do not define and control the type of opens source license allowed to consume.

When defining an open source software usage policy, the following parameters can be considered (but not limited to)

- Type of open source components can be used
- Conditions such as reputation, level of community or commercial support
- Restriction on component age and end-of-support components
- Prohibit components with known vulnerabilities
- License types that are acceptable
- Approval mechanism

It is advisable to have a software compliance group, who is responsible to make sure if the software meet all conditions defined in the open source usage policy and provide necessary approval for consumption.

2.3 Bring Tools to Automate the Process

Even though above steps can be done manually, it is not scalable especially when we have complex applications using hundreds of open source components. There are open source & proprietary tools available to automate many of the steps described above, starting from identifying open source components used, identifying those with vulnerabilities and even suggesting steps to mitigate the vulnerabilities. Some of them have capability to seamlessly integrate with the software development life cycle, so the development team can routinely and effectively use them.

Following are the various type of tools which can be used to automate part of the process described above.

Software Composition Analysis (SCA) Tools

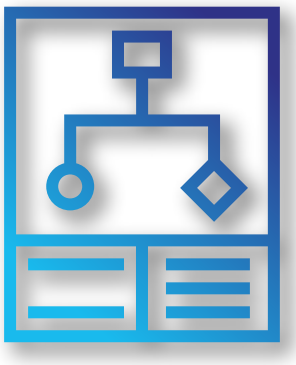
SCA tools help to automate many of the steps described above such as,

- Identifying all software components including dependencies
- Discover all open source components and license information
- Detect publicly disclosed vulnerabilities in open source components

Some advanced SCA tools can even help in open source component selection, license compliance, approval workflow, continuous tracking and integrate with your build/release pipeline.

Some commonly used open source SCA tools are Snyk Open Source, OWASP Dependency-Track. There are proprietary tools available from Veracode, Black Duck Software, Whitesource, Sonatype, etc., which provides end to end solution covering security, licensing & compliance.

We should carefully select the tools-based on the variables such as programming language or technology support, coverage of vulnerability databases & security advisories, integration capability with our CI/CD or build systems, type of license & support model, etc.



B Static & Dynamic Application Security Testing Tools

Static Application Security Testing (SAST) tools examine the code or even binary file to identify potential vulnerabilities. Dynamic Application Security Testing (DAST) Tools simulate a real attacker approaching the application from outside on a running application.

Some examples of open source SAST tools are JSHint which supports Javascript, NodeJSScan for Nodejs based applications, Bandit for Python. There are commercial products available which supports multiple languages and some of them offers SAST, DAST & SCM bundled. Some examples of commercial offering are HCL AppScan, Veracode Application Analysis solution, Microfocus Fortify Application Security and Synopsys Application Security Testing.

A combination of SAST, DAST and SCA tools will help to cover all our application security bases effectively. Integrating these tools in your DevOps lifecycle will help you to better prepare to adopt DevSecOps, but that is another topic for another time!

03 Conclusion

It is not a secret that open source software is an inevitable component of most of the enterprise applications. Open-source software has its fair share of advantages, but when it comes to security, the responsibility falls into us to ensure that there are no vulnerabilities hidden in our code base.

Enterprises that do not have good security practices should revisit to formalize one, and ensure it is being followed strictly across multiple teams, practices, or business units within the organization. Every organizations that use open source software should have a solid usage policy to protect them from both security threats and potential lawsuits due to license issues. When it comes to application security, it is virtually impossible to achieve the desired results without the help of security tools. In this paper, we have provided key best practices along with different tools and automation to balance the usage of open source components and adhere security policies and at the same time to have stable ecosystem.

About the authors

Rakesh M S
Technical Architect

Rakesh, currently working as a technical architect in the Automation Solution Team at LTIMindtree focused on IT automation solution design. He carries more than 10 years of experience covering application development and maintenance, business process automation, and IT Process Automation. He is passionate about technology and puts forth efforts to solve customer problems by leveraging technology.

Vengadesh Babu
Technical Lead

Vengadesh comes with more than 4 years of experience in application development and support. He is currently working as a technical lead in the Automation Solution team at LTIMindtree. He has involved in the development of multiple enterprise applications/IPs which are the foundation for many of our solutions to customers.

LTIMindtree is a global technology consulting and digital solutions company that enables enterprises across industries to reimagine business models, accelerate innovation, and maximize growth by harnessing digital technologies. As a digital transformation partner to more than 750 clients, LTIMindtree brings extensive domain and technology expertise to help drive superior competitive differentiation, customer experiences, and business outcomes in a converging world. Powered by nearly 90,000 talented and entrepreneurial professionals across more than 30 countries, LTIMindtree — a Larsen & Toubro Group company — combines the industry-acclaimed strengths of erstwhile Larsen and Toubro Infotech and Mindtree in solving the most complex business challenges and delivering transforma tion at scale. For more information, please visit **www.ltimindtree.com.**