Your Application is Mostly Written by Strangers

By Edwin Kwan, Head of Application and Software Security, Tyro Payments


Edwin Kwan

Software development has evolved from a waterfall development model to an agile model. Development cycles have shrunk from releasing new versions a few times a year to every couple of weeks or in some organisations, multiple times a day. The applications themselves have also reduced in size, having gone from being large monolithic systems to micro services and now even serverless. And ownership of the applications has also changed. We're moving from a model where applications were once built by developers and then managed by operations, to a "You Build it, You Run it" DevOps model where the team who builds the application is responsible for its operation. Development teams that were once made up of only software engineers are now cross-functional teams and have quality/testing and operations expertise.

The application security landscape has also changed over time. It started as black-box security penetration testing, where the assessors had no knowledge of the application's inner workings This has evolved into white-box testing with the assessor having access to the application's source code. This has improved the quality of their testing as assessors can refer to the source code to determine if a vulnerability exists. We've also seen the introduction of vulnerability scanners and automated security scanning tools. Some of those tools include Static Application Security Testing (SAST), Dynamic Application Security Testing (DAST), and Software Composition Analysis (SCA). SAST does source code analysis to find security vulnerabilities. DAST scans the running application to detect conditions that indicate a security vulnerability. SCA scans the third party, often open-sourced components used by the application for known vulnerabilities.

Penetration testing is still an activity that is performed towards the end of the software development life cycle. However, vulnerability and automated security scanning tools have allowed application security testing to be done earlier. Organisations have shifted security to the left, doing security earlier in the development life cycle, and adopted a continuous application security testing model. This is done by embedding application security testing into the build phase of the software development life cycle, particularly into the Continuous Integration (CI) pipelines. While this approach is a significant improvement to how organisations do application security testing, the approach can be further improved through supply chain management and addressing technical debt in open source components.

As the bulk of modern applications are created using open source components, doing due diligence during the open-source selection process and dealing with stale dependencies will address many potential security vulnerabilities

It is now exceedingly rare for organisations to build their applications from the ground up. Instead, they tend to leverage publicly available open-source components to create the bulk of their applications. Most open source components are designed and supported by a volunteer group of distributed software developers who voluntarily contribute their own time or their company's time to develop the component. According to the 5th Annual Report on Global Open Source Software Development [1], 85% of modern applications are built from open source components. The percentage is higher for modern JavaScript web applications, with 97% of the code in a modern web application coming from open source component packages. So, you can say that a large majority of your application's code is written by a distributed group of strangers rather than your development team.

When it comes to creating applications, the developers usually decide on the programming languages they use. They also select which open-source components to include in their applications. While I am all for empowering developers, there needs to be more due diligence applied to the open source component selection process. Not all open source components are created equal, and in the same annual report [1], 10.3% of all Java libraries downloaded from the maven central repository in 2018 had known vulnerabilities. That figure is higher for JavaScript components, with 51% of the downloaded components having known security vulnerabilities. Vulnerabilities are also prevalent in older components, with those released three years ago or later having 65% more known vulnerabilities [1]. There needs to be an appropriate selection process in place for open source components. This would prevent open source components with known vulnerabilities from being introduced into the application. There has been an uptake of open source consumption in the past five years [1]. And during that time, there has also been a 71% increase in open-source related breaches. The selection process must be lightweight, so it does not impede development, and it should ideally be automated. All new components should be scanned for any known vulnerabilities. It should also be from a reputable source, and the version used should be less than three years old. The benefit of this is not introducing known vulnerabilities into your application and using components that are more likely to be well supported by the open-source community.

As modern applications become more dependent on open source components, one of the biggest challenges we're facing is stale dependencies. Stale dependencies are when an application's open-source components become outdated and are not getting the bug or security fixes that have been addressed by their newer versions. Keeping open source components up to date is not a trivial task as new versions are occasionally not backward compatible. They can introduce breaking changes, and there is potentially a substantial economic cost associated with it. However, as open-source components make up a significant portion of an application's code, usually, most of the security vulnerabilities reside. Letting dependencies become stale and only addressing them once a security vulnerability has been detected is disruptive and slows development significantly. While open source components allow applications to be developed quickly, the associated maintenance effort required is often neglected. This is commonly referred to as the open-source "tax". What organisations need to be doing is scheduling work to address this "tax" on a regular basis. A best practice approach is to mandate that applications must not have stale dependencies when released. Besides, time must be set aside to address stale dependencies in the other applications which are not actively being developed. The benefit of reducing stale dependencies is the reduction in the number of future security vulnerabilities and the time required to address them.

As the bulk of modern applications are created using open source components, doing due diligence during the open-source selection process and dealing with stale dependencies will address many potential security vulnerabilities. These additional controls, coupled with other vulnerability scanners, automated security scanning tools, and penetration testing, will help to speed up development, create more secure applications and reduce business risks. The future of application security is to shift further left.