Minh Allan Dao
madao@ug.kth.se

Tackling Open Source Software Vulnerabilities, From Culture, to Practices, to Tooling

Minh Allan Dao
KTH Royal Institute of Technology

April 24, 2023

Contents

Disclaimer

Please also note that before offering feedback, intention is to resubmit for re-completed/failed task deadline on May 31, 17h Stockholm time.

Notice: I/We certify that generative AI, incl. ChatGPT, has not been used to write this essay. Using generative AI without permission is considered academic misconduct.

Introduction

It is almost essential for open source code to be present when developing any modern project. From a project itself to the tooling to support it, there are countless interconnected pieces of open source tooling and packages needed to build and maintain software. The prevalence and scale of such materials, known as Open Source Software or OSS, naturally means that any flaws manifest as vulnerabilities for software that thereby use it. Household names have failed consumers due to OSS vulnerabilities countless times, such as in the case of the Equifax data leak in 2017, resulting from the deployed version of their web application using a version of Apache Struts with a known vulnerability. This could have been easily addressed, seeing as the vulnerability was known months before the leak occurred (Guckenheimer). Stories like these are present across all types of industries and are the reason for OSS vulnerabilities leading to key focus of the security industry via expanded tooling by leading security companies, in-depth research and education by leading organizations such as OWASP, and increased DevSecOps practices.

What makes OSS powerful and practical is also what makes it dangerous. Being visible and contributed to by anyone, anywhere means that aside from other complexities such as licensing requirements, a major common concern is that malicious users can identify vulnerabilities in the code of OSS and exploit them, or even introduce them themselves. WhiteSource found that 62.8% of 38,333 open source products that customers manage with their software have security vulnerabilities, even though 87% of them have known fixes (Guckenheimer). Knowing that the scale of OSS is massive both in terms of usage, availability, and tech debt indicates the severity of the concerns regarding OSS vulnerabilities, and underscore why we should explore how to approach this issue systematically, starting from culture, to practices, to tooling. Despite OSS being an older Whether a project for friends or an enterprise project for the masses, these focus areas are the keys to both short-term and long-term protection against notable OSS vulnerabilities that can harm global enterprises, trickling all the way down to the average person.

A citation, uses the ref.bib file.[1] Recommend https://www.doi2bib.org/ for getting .bib style references to insert into the ref.bib file.

Culture

How OSS is approached is key to both addressing the issue from a long-term and short-term perspective, and how effective and efficient organizations can do so. Optimistically, fixing issues at the source would be the best solution. However, a dramatic change would require industry-wide endeavor, as the scale of OSS is immense, manpower isn't. Furthermore, as new challenges (constantly) emerge, new OSS is created, while older OSS continue to be updated or exposed with new vulnerabilities. In the latter case, research shows citation new vulnerabilities are more likely to be introduced when the number of lines of code increases, and when code is modified rather than when a new file is created. This correlates with the key idea of manpower, as developers may take fixes from pull requests for granted or fail to take time to find potential flaws via rigorous testing, especially when code changes are long. This issue stems from the vested interest of developers, partly stemming from commitments to public and private development. Although developers employed by companies or working on projects of their own use open source software, they are not being paid or directly motivated to improve the software they use despite it having vulnerabilities. Teams responsible for the maintenance of open source software, if not affiliated with a company, are often then quite small. The complexity of such projects can make it hard to bring more people on, and testing code and/or spotting bugs is a difficult endeavor. Furthermore, estimates indicate that it can often take 4-6 months to fix a bug? citation. This is why it may be in a large company's best interests to divert excess resources in open source development and help fix vulnerabilities in key pieces of software that they use or perhaps develop themselves. This helps them, the community, expands outreach, and also can act as training for engineers to be able to better identify fixed vulnerabilities other OSS used at the organization or in company code. For DevOps engineers, it is easy to naturally stumble upon issues while utilizing open source software (such as for infrastructure), and encouraging engineers to upskill by offering a few hours a week for personal learning such as through open source contributions offers value to all parties. In the same way that tech debt can seem burdensome to approach, the value that comes from tackling it is clear from a security standpoint for OSS.

For DevOps to be successful, the organization culture must be present and shared across a team and broader organization. In a sense, open source vulnerabilities can become technical debt, as it can be daunting to begin keeping track of everything of all possible issues when efforts are made all at single time rather than over time. Even with tools for automation available, it takes time to set up systems that either align or are ingrained in DevOps infrastructure; this takes dedicated time, funding, and manpower. Despite valuable ideas, tools, and innovations present, there must be a vested interest in addressing the concerns that open source vulnerabilities pose. As organizations still advance towards DevOps, a renewed focus security into DevOps is what leads us to DevSecOps, a more current standard for where best practices currently lie. As Itweb puts it, "DevOps and DevSecOps empowered organizations generally are much more proactive in managing their open source component vulnerabilities" (WhiteSource).

Consistent and timely work in identifying and updating/replacing vulnerable OSS is key to addressing the aforementioned issues, as well as firefight in case of any unexpected exploitations. While it may initially have been tedious, continuous monitoring tooling is making it easier to retrieve a full report (Bill of Materials) of open source materials. This can be helpful for examining changes to dependency usage over time to see trends and examine what package(s) are the problematic ones in case of a critical security issue. Having comprehensive information can help in regards to planning upgrades as well as an emergency response, as teams can find alternatives as well as identify and test all possible tracked vulnerabilities from present software. Past security, there are great side effects, such as increased visibility into project structure for new developers and better documentation for legal purposes, such as compliance for licensing. Even if the talent, know-how, and tooling is available, it takes a genuine commitment to take time to address foundational cracks rather than pile on new features; as with all issues contributing to tech debt, developers must be allowed to recognize risk and address it. From a DevOps lens, this means shifting left is the desired structural change.

Practical Examples

Studies such as BLANK show common issues as DOS attacks and SQL injections are common issues stemming from OSS, which could perhaps be examined with static analysis tools integrated into a pipeline.

There are many tools that directly analyze your code, dependency list, and public databases to combat OSS vulnerabilities. A key notion is automation, as dependencies used can be expansive. GitHub, for example, has dependency insights to offer advice for prioritization, offering a notion of low to critical risk based on the dependencies being used. With a healthy developer community that either reports directly on GitHub or shares reports from external organizations, GitHub and its tools offer one of many ways to take advantage of shared knowledge about OSS flagged as vulnerable. There should always be checks to examine if any installed OSS versions correspond with those in the National Vulnerability Database (NVD), a US government repository issued with CVE, indicating presence in the Common Vulnerabilities and Exposures program. Relating back to culture, even with automation, we see that there must be a direct human effort to populate data that automation picks up on to offer warnings.

A notable increase in complexity stems from how dependencies are often interconnected, whether direct or transitive. Such interconnectedness lends to a dependency graph, and thus allows data to be aggregated about risk at all layers through graph traversal algorithms, where the comprehensive tracking (dependency mapping) helps identify version upgrades as necessary for related OSS and better manage the software lifecycle (“Dependency Mapping for DevSecOps”). As each piece of OSS is examined for connections, each can be recorded and further examined. Other DevSecOps tools include code analysis/scanning tools that are helpful for such deeper inspection; automation of identification of issues such as logic flaws and insecure legacy code can be integrated with monitoring and management systems, container vulnerability scanners, and code review tools. Such flaw identification can be seen with http-proxy, which has over 14 million weekly downloads via npm. http-proxy had a vulnerability fixed in mid-2020 to avoid Denial of Service attacks. For example, an HTTP request with a long body triggered the unhandled exception $ERR\_HTTP\_HEADERS\_SENT$ that crashes the proxy server when the proxy server sets headers in a given request using the proxyReq.setHeader function. Part of the fix relating to proxyReq is seen below:

https://github.com/advisories/GHSA-6x33-pw7p-hmpq https://github.com/http-party/node-http-proxy/pull/1447 This fix was merged May 17, 2020, 3 days after a GitHub issue was raised related to the npm audit that triggered the warning about the vulnerability. This issue had a direct impact on the security of CI pipelines worldwide. Even though we see a healthy amount of discussion about immediate resolutions and a long-term fix, the latter was only present after two days, which presented a window of opportunity for malicious users. Afterwards, upgrades must be installed after the OSS upgrade rollout. Seeing as a random chance is unreliable (such as if a developer happens to be subscribed to a newsletter that also picked up on the issue), an automated tool looking for new CVE ids and scanning new flags from vulnerability research, such as from the npm audit responsible for finding the vulnerability. Clearly, we can learn from this example of auditing that DevSecOps should also include continuous auditing. It is worthwhile to invest in tools that make it easier for both OSS code to be improved and for users of such packages to weigh versioning risks, which could occur in cases such as if a vulnerable dependency can’t be upgraded due to another piece of software relying on it. Gibberish here In turn, the tackling of bloat is a key measure in reducing risk, as there are less variables at play reducing interconnectedness. Less things to keep track of and/or to potentially activate also is key in issues arising from dynamic programs, as further explored below. Thus, parts of DevOps itself are vulnerable to vulnerabilities, but luckily, DevOps practices mindful of this issue can allow us to address them.

Dynamic Challenges

The recommendations for a strong code foundation, including a clean codebase, tracked dependencies, and static analysis for dependencies and tooling all augment a clean and clear environment to do dynamic testing, which is the last area of focus and perhaps the most complex. A tool of particular interest is Veracode Dynamic Analysis, a Dynamic Application Security Testing (DAST) solution for web applications. The tool interacts with a given application or API in a way a malicious user would, examining standard content such as links and forms, as well as header values and cookies that aren’t directly visible. Tests can be both authenticated and unauthenticated, and can be further extended with other Veracode tools that examine within networks and behind firewalls. By crawling your site and finding potential issues, they may be linked to vulnerabilities that stem from dependencies. https://nonamesecurity.com/blog/Log4J-vulnerability-apis-causing-massive-risk-exposure In addition, Eclipse Steady can examine compiled Java code and report vulnerabilities by examining "fix-commits". There is greater complexity in how exploits can appear due to how Java programs load in code, so malicious users can still exploit programs through dependencies in the way code is structured to load in, as mentioned.

Some Reflections and Concluding Ideas

The author had an opportunity to work for a bank, an industry known for a slower pace due to bureaucracy and regulation. Looking back, although the pace could be frustrating, it offered time to be more conscious about overlooked details, such as OSS. For auditing purposes, notable external software had to be submitted to a security team for analysis, which had the two-fold opportunity of ensuring a fair and external security review, and naturally created documentation from the software usage submission. Although it was regarded to be on the more extreme end regarding comprehensive documentation, having a lot of information available made the author's team better prepared for potential future incidents, and frustration was lower since processes and tools were already in place, as OSS issues, both regarding licensing and vulnerability, were at the forefront from the start. For example, remediation for Log4J was initially much quicker to tackle with comprehensive information already available. Shifting left and utilizing automation in deployment pipelines to lessen the burden, despite there still being some manual work, ensured the team had a greater appreciation of their time spent embroiled in compliance culture. There is clear evidence from organizations like WhiteSource that state that higher rates of security scanning correlate to less risk over the long term, indicating lengthened pipeline runs and the cost computing resources dedicated to such activities is worthwhile.

A more recent and infamous example that sparks interest in OSS vulnerabilities is the Log4J vulnerability, which affected the open-source Java-based logging utility tool for a variety of applications. The Log4J library allowed malicious users to execute remote code due to a flaw (now known as CVE-2021-44228) in the way it processed data containing external code. The vulnerability impacted organizations of all kinds worldwide, including government agencies, financial institutions (as detailed above), and tech companies. Although an emergency patch was quickly developed by the Log4J development team, there was still immense panic worldwide to the significant threat imposed on corporate security. The panic was noted for significant manual efforts to address enterprise level concerns, exacerbated considering the scale and quantity of applications at many of the affected organizations, as well as lack of efforts to contain OSS vulnerability sprawl beforehand, especially with so much OSS dependent on Log4J. This is the quintessential example of why OSS vulnerabilities are widely persistent, and why the lack of a culture that drives relevant practices and usage of (automated) tooling creates significant amounts of tech debt and risk. Even as these factors are considered in a transformational shift to DevSecOps, in some ways, it is ironic that DevOps tooling itself is prominently OSS, so the foundation of significant value in the field comes with the very risks we are discussing. Overall, OSS vulnerabilities are at the forefront of the issues, but despite being an issue at a massive scale out of any single entity's ability to address it, there is a clear path to drastically reduce risk if developers are afforded the time and investment to approach development with the right security mindset and if initially investments can be made early on to automate OSS documentation and remediation tooling.

Minh Allan Dao
madao@ug.kth.se

References

[1] Tom L. Beauchamp and Oliver Rauprich. "Principlism". In: *Encyclopedia of Global Bioethics*. Springer International Publishing, 2015, pp. 1–12. DOI: `10.1007/978-3-319-05544-2_348-1`. URL: `https://doi.org/10.1007/978-3-319-05544-2_348-1`.