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

May 22, 2023

Contents

Disclaimer

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. There are countless interconnected pieces of open source tooling and packages needed to build and maintain software, both for solo developers, teams, and companies. For example, free/libre Open Source Software or OSS saves the European economy roughly €114 billion per year directly, and up to €399 billion per year overall [1]. The prevalence and scale of OSS naturally means that any flaws manifest as vulnerabilities in the software and thereby the software that uses it. This is a highly relevant issue: countless household names have failed consumers and loss significant revenue due to OSS vulnerabilitiy exploitation. This was readily seen in the Equifax data leak in 2017, due to their deployed web application using a version of Apache Struts with a known vulnerability [2]. This could have been easily addressed, seeing as the vulnerability was known months before the leak occurred, and the fallout is still being dealt with by Americans today. Stories like these are present across all industries and are the reason for OSS vulnerabilities being a key focus of the security industry. WhiteSource's 2021 report 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. What makes OSS powerful and practical is also what makes it dangerous. The scale of use, high visibility, and openness to contribution means that good actors can update the software, while bad actors can find and exploit vulnerabilities. Fortunately, expanded tooling by leading security companies, in-depth research, and education by leading organizations such as OWASP are key initiatives to promote safer OSS practices.

Culture

How OSS is approached is key to reducing the burden of OSS, as it is often a matter of tech debt. Optimistically, fixing issues at the source would be the best solution. However, such a dramatic change would require industry-wide endeavor, as the scale of OSS is immense, manpower isn't. The complexity of such projects makes it hard to bring more people on, and even if bugs are found (which is difficult itself), there are estimates indicating that it can often take 3-5 months to fix a bug due to a variety of factors [3]. One interesting idea is that it may be in a company's best interests to divert excess resources to key OSS. This helps the company's security and perhaps image (to an extent), the community, and also upskills engineers. Anecdotally, companies have started to offer a few hours a week for long-term growth and learning, time which could be spent on open source contributions.

Looking deeper into culture, we can examine internal organizational culture and note that a healthy engineering mindset must be present and shared across teams and stakeholders for effective DevSecOps practices to be in place and to be practiced. Common practices for OSS vulnerabilities including upgrading software to safer versions and tracking usage, but it can be daunting to track so much information. Although tools for automation available, it takes time to set up systems that can live in DevOps infrastructure; this takes dedicated time, funding, and manpower that organizations must be willing to commit. No matter the research and tools present, there must be a vested interest in OSS best practices. Itweb puts this agreed sentiment best: "DevOps and DevSecOps empowered organizations generally are much more proactive in managing their open source component vulnerabilities".

Consistent and timely upkeep may be tedious, but continuous monitoring tooling is making it easier to identify upgrades retrieve a full report (Bill of Materials) of OSS [4]. This can be helpful for examining changes to dependency usage over time and examine what package(s) are the most problematic and thus cause the most risk, which would be the best to examine sooner in the case of an attack occurring. Having comprehensive information helps with upgrade plans, identification of alternatives, and preparedness to firefight. If attackers are successful, having a list to identify possible weaknesses is the key in systematically identify and removing the software that could've been exploited. 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 OSS licensing [4]. This all factors into strong supply chain management, with standardized and systematic practices company wide [1].

Practical Examples

There are many tools that directly analyze your code, dependency list, and public databases to combat OSS vulnerabilities. Automation here is the key, given the scale of dependencies and issue types. DOS attacks and SQL injections are common issues via OSS, which could found with static analysis tools integrated into a pipeline [3]. These tools may also include checks to examine if any installed OSS versions correspond with those in the National Vulnerability Database (NVD), a US government repository issued with CVE, the Common Vulnerabilities and Exposures program that tracks notable risks in the industry[2].

An 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

Minh Allan Dao
madao@ug.kth.se

function. Part of the fix relating to proxyReq is shown below:

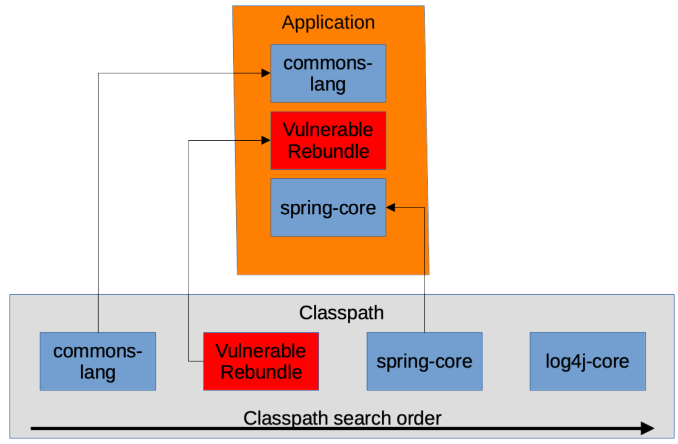
```
    // Enable developers to modify the proxyReq before headers are sent
    proxyReq.on('socket', function(socket) {
-     if(server) { server.emit('proxyReq', proxyReq, req, res, options); }
+     if(server && !proxyReq.getHeader('expect')) {
+       server.emit('proxyReq', proxyReq, req, res, options);
+     }
    });
```

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 [5] [6]. 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. From this, we can learn that DevSecOps should also include continuous auditing to flag the most up to date issues.

Dynamic Challenges

A strong code foundation, including a clean code base, tracked dependencies, and static analysis for dependencies and tooling can create a clean and clear environment to do dynamic testing, which is the last area of focus and perhaps the most complex. Fortunately, tools such as Veracode Dynamic Analysis, a Dynamic Application Security Testing (DAST) solution for web applications are available. Of particular interest is how Veracode can crawl an application to find vulnerabilities that stem from dependencies.

Other tools such as Eclipse Steady can examine compiled Java code and report vulnerabilities by examining "fix-commits," as analyzing construct changes (ones impacting the AST) is less expensive and still efficient [7]. One study compared Steady with the OWASP Dependency Check (OWASP DC), scanning 300 large enterprise applications under development with a total of 78165 dependencies and found all code resulting code errors to be true positives, while only using OWASP DC led to 88.8% false positives with a differing code-centric approach [7]. Although we should be wary of greater bloat, clearly multiple tools and strategies are helpful to tackle vulnerabilities due to less obvious minute details such as code bundling and dynamic loading, as seen in the following diagram:



Example of rebundle in a Java program via SAP [8]

Ultimately, this is indicative of how complex OSS vulnerabilities can be, which is all the most reason that organizations need to invest in the tooling available to automatically scan software. Again, we should also minimize risk, considering newer practices such as debloating, in which we remove as much extra code and dependencies from our own software and from OSS as possible in a manner that does not change its functionality, thereby reducing available "attack surface" [8].

Final Thoughts

The bank industry is well-known for a slower pace due to bureaucracy and regulation. With more time, developers could be more conscious about OSS for auditing and security purposes. For example, in a notable American bulge bracket, external software had to be submitted to a security team for analysis, naturally creating documentation from the submissions and more careful usage of tooling. Although the manual review process did cause frustration, OSS was less of a concern since there were already standardized procedures. For example, remediation for Log4J was initially much quicker with comprehensive information about which dependencies also had Log4J as a dependency. Auditing OSS, both regular and suprise, as well as internal and external, were also made it easier [1].

In some ways, it is ironic that DevOps tooling itself is prominently OSS, so the very foundation of infrastructure that can address the risks we are discussing also have them as well. OSS vulnerabilities are deeply rooted in all of software engineering, with potential massive scale and impact for any given exploit. Despite this, many practices exist to reduce risk if time and investment is put into the right security mindset where automation augments OSS tracking and active remediate.

Minh Allan Dao
madao@ug.kth.se

References

[1] Nikolay Harutyunyan. "Managing Your Open Source Supply Chain-Why and How?" In: *Computer* 53 (June 2020), pp. 77–81. DOI: 10.1109/MC.2020.2983530.

[2] Ilya Kabanov and Stuart Madnick. "Applying the Lessons from the Equifax Cybersecurity Incident to Build a Better Defense". In: *MIS Quarterly Executive* 20 (June 2021), p. 4. DOI: 10.17705/2msqe.00044.

[3] Gede Prana et al. "Out of sight, out of mind? How vulnerable dependencies affect open-source projects". In: *Empirical Software Engineering* 26 (July 2021). DOI: 10.1007/s10664-021-09959-3.

[4] Andreas Bauer et al. "Challenges of Tracking and Documenting Open Source Dependencies in Products: A Case Study". In: *Open Source Systems*. Ed. by Vladimir Ivanov et al. Cham: Springer International Publishing, 2020, pp. 25–35. ISBN: 978-3-030-47240-5.

[5] GitHub. *Denial of Service in http-proxy*. URL: https://github.com/advisories/GHSA-6x33-pw7p-hmpq. (accessed: 20.05.2023).

[6] jcrugzz. *Skip sending the proxyReq event when the expect header is present*. URL: https://github.com/http-party/node-http-proxy/pull/1447. (accessed: 20.05.2023).

[7] Serena Ponta, Henrik Plate, and Antonino Sabetta. "Detection, assessment and mitigation of vulnerabilities in open source dependencies". In: *Empirical Software Engineering* 25 (Sept. 2020). DOI: 10.1007/s10664-020-09830-x.

[8] Serena; SAP Ponta. *The little-known side of vulnerabilities in open source dependencies of applications*. URL: https://blogs.sap.com/2022/08/18/the-little-known-side-of-vulnerabilities-in-open-source-dependencies-of-applications/. (accessed: 20.05.2023).