After All, is Reuse a Security Bottleneck?

Daniel Feitosa
Data Research Centre
University of Groningen
Leeuwarden, Netherlands
d.feitosa@gmail.com

There are many reasons to reuse, from a simple wish of mitigating effort to goals such as improving quality [1]. To avoid reinventing the wheel, reuse in software development is getting easier by the day. Together with associated practices, build systems such as `maven` and `sbt`, and centralized package managers such as `npm` and `pip`, play a role in making reuse widely adopted and advocated by practitioners and researchers alike. However, we are also every so often reminded of the potential risks associated with reuse, in particular regarding security. For example, Heartbleed was a severe security vulnerability in OpenSSL that affected 66% of the active web sites around 2014.[1] More recently, a known vulnerability in a third-party Java library that Equifax reused,[2] led to the stealing of private information of more than 147 million American citizens.

In this context, there is a recurring question of whether systematic reuse is to blame for such incidents. Should development teams strive for reinventing the wheel in order to mitigate security risks? This presentation will attempt to discuss and shed further light on the matter. For that, the content of the talk is based on the reporting of an empirical study recently published on the proceedings of ICSR '19 [2], as well as its extension, which has been recently submitted to a special issue of the Journal of Systems and Software [3].

The reported case studies aimed at exploring and discussing the relationship between software reuse and the number of security vulnerabilities in open source projects. In particular, we examined the distribution of vulnerabilities among the code created by a development team (i.e., native code) and code reused from third-party dependencies. The first study considered 301 projects from the Reaper database [4] and focused on potential vulnerabilities detected through static analysis, while the second study considered 1 244 projects from the GitHub Activity Data database[3], and focused not only on potential but also on disclosed vulnerabilities reported publicly. The datasets of the two separate studies are available on Zenodo,[4] and the toolkit and guidelines to reproduce their creation and analysis processes are available on GitHub[5].

The results suggest that larger projects (in size) are related with an increased amount of potential vulnerabilities in both native and reused code. Furthermore, although native code appears to have a higher vulnerability density, the data does not provide strong evidence to support the hypothesis. These findings place a heavier weight on the development team, as it is responsible for verifying the maturity of reused code and trade off in-house expertise to write the components from scratch with the attached reuse risks.

The aforementioned scenario is worrisome as developers are oftentimes unaware of the security risks introduced by reused code, as found by Kula et al. [5], and the number of disclosed vulnerabilities in open source libraries is increasing at concerning rates[6]. Moreover, identifying the best opportunities for reuse is also not trivial, as we observed that the use frequency of a dependency is not correlated to its level of security. However, on a positive note, our results also suggest that potential vulnerabilities can be indicators of whether actual vulnerabilities may reside in the code.

Altogether, progress on the state of the art and practice in reuse reduced its effort, while the open source culture led to a boom of reusable components. From a certain point, reuse may have become so trivial that security risks may be unconsciously neglected at times. Going forward, there is a growing need for advancements in both research and practice to further: (a) mitigate the risk and fear of updating outdated dependencies, (b) automate and integrate warning systems for vulnerable dependencies, and (c) investigate vulnerabilities at a lower level of granularity, allowing tracking of low-level artefacts (e.g., classes or procedures) involved in the reuse of vulnerable code.

References

[1] M. A. Rothenberger, K. J. Dooley, U. R. Kulkarni, and N. Nada, "Strategies for software reuse: a principal component analysis of reuse practices," *IEEE Trans. Softw. Eng.*, vol. 29, no. 9, pp. 825–837, Sep. 2003.

[2] A. Gkortzis, D. Feitosa, and D. Spinellis, "A double-edged sword? software reuse and potential security vulnerabilities," in *Reuse in the Big Data Era*, X. Peng, A. Ampatzoglou, and T. Bhowmik, Eds. Cham: Springer International Publishing, 2019, pp. 187–203.

[3] ——, "Software reuse cuts both ways: An empirical analysis of its relationship with security vulnerabilities," 2019, (submitted).

[4] N. Munaiah, S. Kroh, C. Cabrey, and M. Nagappan, "Curating GitHub for engineered software projects," *Empirical Software Engineering*, vol. 22, no. 6, pp. 3219–3253, Dec 2017.

[5] R. G. Kula, D. M. German, A. Ouni, T. Ishio, and K. Inoue, "Do developers update their library dependencies?" *Empirical Software Engineering*, vol. 23, no. 1, pp. 384–417, Feb. 2018.

[1] https://news.netcraft.com/archives/2014/04/02
[2] https://www.equifaxsecurity2017.com/
[3] https://console.cloud.google.com/marketplace/details/github/github-repos
[4] http://doi.org/10.5281/zenodo.2566054
[5] https://github.com/AntonisGkortzis/Vulnerabilities-in-Reused-Software
[6] https://snyk.io/blog/88-increase-in-application-library-vulnerabilities-over-two-years/