Preliminary Findings on FOSS Dependencies and Security

A Qualitative Study on Developers' Attitudes and Experience

Ivan Pashchenko
ivan.pashchenko@unitn.it
University of Trento, IT

Duc-Ly Vu
ducly.vu@unitn.it
University of Trento, IT

Fabio Massacci
fabio.massacci@unitn.it
University of Trento, IT

ABSTRACT

Developers are known to keep third-party dependencies of their projects outdated even if some of them are affected by known vulnerabilities. In this study we aim to understand *why* they do so. For this, we conducted 25 semi-structured interviews with developers of both large and small-medium enterprises located in nine countries. All interviews were transcribed, coded, and analyzed according to applied thematic analysis. The results of the study reveal important aspects of developers' practices that should be considered by security researchers and dependency tool developers to improve the security of the dependency management process.

KEYWORDS

Dependency management, security, vulnerable dependencies, qualitative study, interviews

ACM Reference Format:
Ivan Pashchenko, Duc-Ly Vu, and Fabio Massacci. 2020. Preliminary Findings on FOSS Dependencies and Security: A Qualitative Study on Developers' Attitudes and Experience. In *42nd International Conference on Software Engineering Companion (ICSE '20 Companion), October 5–11, 2020, Seoul, Republic of Korea.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3377812.3390903

1 RESEARCH PROBLEM AND MOTIVATION

Components with known vulnerabilities (#9 from OWASP Top 10 list of Web Application Security Risks[1]) are the most frequent cause of severe security breaches: according to the Snyk report[2], known vulnerable components were the root cause of 24% of severe security breaches, like the Equifax breach[3] due to an outdated Apache Struts library, the Panama Papers data leak[4] due to an old unpatched version of Drupal, and the Ubuntu forum breach[5] due to an outdated Forumrunner add-on. Still, developers often keep third-party components used in their projects outdated.

There is a strong temptation to fingerpoint FOSS developers for lack of care. However, a more careful analysis reveals more nuances in these broad findings. For example, a study of the android ecosystem [3] argued that many libraries were vulnerable and could be easily updated. A later study by the same group [5] showed that the original claim was too optimistic: the 'easy' update would instead create breaking changes in around 50% dependent projects. Similarly, an initial study on the Maven ecosystem [7] argued that many libraries included vulnerable dependencies. A later study [9] showed that several of those vulnerabilities were in test/development libraries and thus not shipped with the product and, therefore, irrelevant. Developers may, therefore, not be entirely irrational in *not always* updating their libraries.

Hence, understanding the developers' decision-making strategies while selecting and/or updating dependencies of their projects is important for both security researchers and dependency tool developers, so they can design appropriate methodologies and tools to improve the security of the dependency management process.

[1]https://owasp.org/www-project-top-ten/
[2]https://snyk.io/blog/owasp-top-10-breaches/
[3]https://investor.equifax.com/news-and-events/news/2017/09-15-2017-224018832
[4]https://www.icij.org/investigations/panama-papers/
[5]https://ubuntu.com/blog/notice-of-security-breach-on-ubuntu-forums

2 BACKGROUND

Quantitative empirical studies of software dependencies (e.g., [7, 9]) mainly focus on the techniques, and therefore, facilitate the ways *how* developers perform dependency management. However, they provide limited insights on the developers' motivations while managing software dependencies, such as *why* developers adopt new dependencies or update/not update the already used ones. Instead, we are interested in understanding the developers' reasoning while selecting and updating dependencies.

On the other hand, qualitative dependency studies (e.g., [1, 2]) suggest that dependency issues might affect developers' decisions. However, the studies focus mainly on functionality issues and do not investigate the influence of security concerns.

The qualitative studies of technologies and tools for automating the software engineering process (e.g., [6, 11]) provide interesting insights into the software developers' experience, but do not consider dependency analysis tools, and therefore, do not study how developers can use them to discover and mitigate security issues introduced by software dependencies.

The studies on information needs (e.g., [8, 10]) provide useful insights on developers' decision-making strategies, however, the currently available studies do not show how the developers' actions and decisions change due to security concerns and the presence of the issues introduced by software dependencies.

3 APPROACH

To find the incentives of developers' motivations for (not) updating dependencies of their projects, we interviewed developers of 25 different companies located in 9 countries and analyzed their strategies for (i) selecting new dependencies, (ii) updating currently used dependencies, (iii) using automatic dependency management

Table 1: Interviewees in our sample

By location, we specify the current country of the developer workplace. We have clustered the companies as follows: free and open-source project (FOSS project), large enterprise (LE), small and medium-sized enterprise (SME), and user group (UG).

#	position	company type	country	exper. (years)	primary languages
#1	CTO	SME	DE	3+	Python, JS
#2	Moderator	UG	IT	10+	Java
#3	Developer	LE	IT	10+	Java, JS
#4	CEO	SME	SI	7+	Python, JS
#5	Developer	SME	NL	3+	Python
#6	Freelancer	SME	RU	3+	Python, JS
#7	Developer	SME	DE	5+	Python, JS
#8	Developer	LE	RU	4+	Python, JS
#9	CTO	SME	IT	4+	JS
#10	Developer	LE	DE	10+	C/C++
#11	Developer	LE	VN	5+	C/C++
#12	Developer	SME	DE	4+	Java, Python
#13	Team leader	LE	RU	10+	JS
#14	Developer	SME	RU	4+	Java
#15	Project Leader	FOSS	UK	10+	Python, C/C++
#16	Developer	SME	IT	8+	Java
#17	Developer	LE	VN	3+	Java
#18	Senior Software Engineer	LE	IT	10+	Python, C/C++
#19	Developer	SME	RU	3+	Java
#20	Security Engineer	LE	DE	3+	JS
#21	Developer	SME	HR	3+	JS
#22	Developer	SME	IT	8+	JS
#23	Developer	LE	IT	9+	Java
#24	Full stack developer	SME	IT	3+	JS, Python
#25	Developer	SME	ES	3+	C/C++

tools, and (iv) mitigating bugs and vulnerabilities for which there is no fixed dependency version. The interviewees have at least three years of professional experience at various positions spanning from regular developers to company CTOs. Table 1 describes the sample of the developers in our study.

We followed the qualitative process of the 'grand-tour' semi-structured interviews. Each interview (lasting 30' on average) was recorded and transcribed. The transcripts were anonymized and sent back to the interviewees for confirmation. Each conversation was then coded along the lines of applied thematic analysis [4] to provide a quantitative assessment of the qualitative data so collected. After completing the analysis, we also returned the overall findings to the participating developers to check that we have not misinterpreted their thoughts (MemberCheck).

4 PRELIMINARY FINDINGS

We summarize our findings of the facts the developers' reported us during the interviews as follows:

Library selection. When selecting a new dependency, developers pay attention to security only if it is required and enforced by the policy of their company. Otherwise, they mainly rely on popularity and community support of libraries (e.g., number of stars, forks, project contributors).

Updating software dependencies. As generally, developers lack resources to cope with possible breaking changes, they prefer to avoid updating dependencies for any reason. Security vulnerabilities motivate developers for updating only if they are severe, widely known, and adoption of the fixed dependency version does not require significant efforts.

Automation of dependency management. Developers perform sensitive dependency management tasks (e.g., updates) manually. Current dependency analysis tools (if used) only facilitate the identification of vulnerabilities in the project dependencies. Developers complain that dependency tools produce many false-positive and low-priority alerts.

Unfixed vulnerabilities. The interviewed developers suggested the following actions when a vulnerability is discovered in a dependency, but no newer version fixes it:

- assess whether this vulnerability impacts them since maybe they may not use that particular functionality;
- wait for the fix or a community workaround;
- adapt own project, i.e., disable vulnerable functionality or rollback to a previously safe version of the library;
- maintain own fork of a dependency project (possibly fixing and making a pull request to the dependency project).

5 CONCLUSIONS

We present a qualitative study of developers' decision-making practices in the presence of security concerns for selecting new dependencies, updating the already used ones, usage of dependency analysis tools, and coping with vulnerable dependencies that do not have a fixed version.

Our study provides important insights and suggestions for security researchers on how to design better approaches for secure dependency management and dependency tool developers on how to improve the dependency analysis tools.

ACKNOWLEDGMENTS

This research has been partly funded by the EU under the H2020 Programs H2020-EU.2.1.1-CyberSec4Europe (Grant No. 830929) and the NeCS: European Network for Cyber Security (Grant No. 675320).

REFERENCES

[1] Christopher Bogart, Christian Kästner, James Herbsleb, and Ferdian Thung. 2016. How to break an API: cost negotiation and community values in three software ecosystems. In *Proc. of FSE'16*. ACM, 109–120.
[2] Joël Cox, Eric Bouwers, Marko van Eekelen, and Joost Visser. 2015. Measuring Dependency Freshness in Software Systems. In *Proc. of ICSE'15 (ICSE '15)*. IEEE Press, Piscataway, NJ, USA, 109–118. http://dl.acm.org/citation.cfm?id=2819009.2819027
[3] Erik Derr, Sven Bugiel, Sascha Fahl, Yasemin Acar, and Michael Backes. 2017. Keep me updated: An empirical study of third-party library updatability on Android. In *Proc. of CCS'17*. ACM, 2187–2200.
[4] Greg Guest, Kathleen M MacQueen, and Emily E Namey. 2011. *Applied thematic analysis*. Sage.
[5] J. Huang, N. Borges, S. Bugiel, and M. Backes. 2019. Up-To-Crash: Evaluating Third-Party Library Updatability on Android. In *Proc. of EuroS&P'19*. 15–30.
[6] Brittany Johnson, Yoonki Song, Emerson Murphy-Hill, and Robert Bowdidge. 2013. Why don't software developers use static analysis tools to find bugs?. In *Proc. of ICSE'13*. IEEE Press, 672–681.
[7] Raula Gaikovina Kula, Daniel M. German, Ali Ouni, Takashi Ishio, and Katsuro Inoue. 2017. Do developers update their library dependencies? *Emp. Soft. Eng. Journ.* (11 May 2017). https://doi.org/10.1007/s10664-017-9521-5
[8] Lucas Layman, Madeline Diep, Meiyappan Nagappan, Janice Singer, Robert Deline, and Gina Venolia. 2013. Debugging revisited: Toward understanding the debugging needs of contemporary software developers. In *Proc. of ESEM'13*. IEEE, 383–392.
[9] Ivan Pashchenko, Henrik Plate, Serena Elisa Ponta, Antonino Sabetta, and Fabio Massacci. 2018. Vulnerable Open Source Dependencies: Counting Those That Matter. In *Proc. of ESEM'18*.
[10] Khaironi Y Sharif, Michael English, Nour Ali, Chris Exton, JJ Collins, and Jim Buckley. 2015. An empirically-based characterization and quantification of information seeking through mailing lists during open source developers' software evolution. *Information and Software Technology* 57 (2015), 77–94.
[11] Carmine Vassallo, Sebastiano Panichella, Fabio Palomba, Sebastian Proksch, Andy Zaidman, and Harald C Gall. 2018. Context is king: The developer perspective on the usage of static analysis tools. In *Proc. of SANER'18*. IEEE, 38–49.