Applying countermeasures to vulnerabilities in code, through AI-driven refactoring and co-evolution

1. PhD title

Applying countermeasures to vulnerabilities in code, through AI-driven refactoring and co-evolution

2. PhD topic

The purpose of the thesis is, from a known catalogue of vulnerabilities in the code and their already classified fixes or mitigations, to use artificial intelligence techniques to learn 1) to find these vulnerabilities or their variants in a source code, and 2) to apply the countermeasures or their variants on the source code, in order to remove the vulnerability. AI provides adaptability, and thus robustness with respect to the code under examination, both in detection and in application of the countermeasure (including code co-evolution in depth, i.e. change propagation).

3. PhD description

By analysing the 1,487,956 versions of all 73,653 software libraries available in Maven Central, we observe that more than 30% of the libraries (and 50% for the most popular libraries) have multiple versions that are currently in active use [1, 8]. These empirical observations reveal that many projects (i.e. library clients) do not track the development of their dependencies and do not co-evolve their code (i.e. adapt their code in response to library changes) to use the latest versions, due to the manual effort this may require from developers. As a result, when vulnerabilities are fixed in the latest version of a library, the applied fixes do not benefit all its clients.

A well-known example is the LOG4Shell vulnerability in the Log4J library which was not fixed at once, but in fact on four successive versions, namely 2.15, 2.16, 2.17 and 2.17.1. Customers therefore had to upgrade to the new versions four times in order to secure the attack surface in their code. Unfortunately, co-evolution remains a manual, tedious and error-prone task, which discourages clients from co-evolving to new versions of the libraries or leads to erroneous co-evolution. It is therefore crucial to facilitate the automation of this co-evolution between libraries and their clients. Furthermore, after the initial co-evolution has been applied, it may be necessary to refactor the code, i.e. restructure the code without changing or adding any behaviour or functionality. The aim is to keep the code efficient and maintainable over time. Ideally, in this work, refactoring should not only take into account the language paradigm and constructs, but also "hidden" aspects of the code such as annotations, domain, or other external properties and known security anti-patterns [4, 6, 7]. Therefore, developers should be helped by proposing refactorings that take cybersecurity into account.

AI algorithms (in particular machine learning) will provide the capacity for generalisation in order to obtain an adaptation capacity and thus a robustness with respect to the examined code, which entails both variations in the form of the vulnerability to be detected and fixed and variations in the countermeasure to be applied. In fact, due to the local context, there may be variations between the implementation of the vulnerability (and, consequently, its correction) and its description in the catalogues.

These variations may include not only local syntactic variations, but also variations in depth within the code and libraries under consideration, necessitating co-evolution of the code during the countermeasure propagation phase.

While refactoring support is applied at the code level, it is also possible to reason about it at the model level prior to application. For instance, the introduction of a front-end design pattern to decouple a particular application from a specific version of the library is an improvement to security. As it affects the architecture, it must be performed at a more abstract level.

Automating the co-evolution of clients as a result of library evolution due to vulnerability fixes will facilitate the propagation of security fixes into client code at the appropriate moment. Similarly, the application of refactoring that is cognizant of cybersecurity will improve code quality and security levels.

This work will rely as much as possible on public databases and existing developments, for example, MITRE CVE (Common Vulnerabilities and Exposures) [14], Software Heritage [15].

The expected outcome of this thesis is a system for modelling and detecting vulnerabilities and their variants in private or public code bases for civil and/or military applications.

4. PhD programme

First, we intend to automate the co-evolution of libraries and their client code in the Maven ecosystem. Particular attention will be paid to co-evolutions associated with the subtleties of vulnerability fixes in libraries, i.e. genuine fine-grained code modifications. For instance, co-evolution as a result of the replacement of a vulnerable function with a newer, more secure function, or the inclusion of security parameters that must be configured in client calls, etc. In order to accomplish this, we will rely on patches or countermeasures that will serve as components for the co-evolution of client code. We will investigate various client-level co-evolution strategies, such as using artificial intelligence algorithms based on machine learning from existing co-evolutions in response to similar fixes in other libraries, both for vulnerabilities and non-vulnerabilities.

A second step in this work will be to consider vulnerabilities that may not correspond exactly to those described in the catalogues (e.g., a vulnerability constructed from several others) [10], thus making the associated countermeasures sub-optimal. The artificial intelligence algorithms implemented will therefore have to be able to compose known solutions, or extrapolate from them, in order to best respond to these vulnerabilities and thus allow relevant and adapted refactoring.

In addition, we propose designing cybersecurity-aware refactorings that account for both code modifications that may introduce vulnerabilities and known Java security antipatterns [4, 6, 7]. Refactorings should also consider the presence of additional concerns, such as calls to specific secure functions, storage in a given registry, the use of Secure Sockets Layer (SSL) and Transport Layer Security (TLS) protocols, etc. Therefore, cybersecurity-aware refactorings should not introduce inherent security vulnerabilities by ignoring the aforementioned considerations. Instead, they should adhere to the established design rules that we will define as part of our cybersecurity-aware refactoring methodology, such as avoiding unprotected access to global variables and utilising secure protocols whenever feasible. In addition to the possible use of verification tools in a later phase, the tools we will provide will ensure compliance with these rules. Not only will these rules apply at the level of the code, but they will also include architecture rules, such as design patterns and antipatterns.
This thesis intends to build upon a set of existing tools, including GumTree [3], RefactoringMiner [9], and the co-evolution techniques developed at DiverSE [2, 5].

This thesis will take place within the DiverSE team of INRIA/IRISA/University of Rennes, under the responsibility of Olivier Barais (Professor at the University of Rennes, head of the DiverSE team), Paul Temple (Lecturer at the University of Rennes), and Olivier Zendra (Inria Research Fellow).

It will be carried out in connection with the team's other projects. In particular, the synergies identified with the Software Heritage Security - SWHSec project (in which DiverSE is co-leader), funded by the Cyber Campus [16], are as follows:

1. Reuse, in the context of this thesis, of the work done by an engineer in SWHSec to interface with existing catalogues of vulnerabilities and their patches. This provides more input data for this thesis, without duplicating development that is already planned in SWHSec.

2. Use, in the context of this thesis, of Software Heritage as one of the (large) source code bases we can explore, benefiting additionally from the engineering work to facilitate access to SWH that is already planned in SWHSec WP1. Again, this provides more input for this thesis, without duplicating the development that is already planned in SWHSec.

3. Some of the research work done in SWHSec can be compared with the work done in this thesis. The latter is clearly more AI oriented than what is planned in SWHSec, and therefore complementary.

5. Applications

Applicants must have a Masters degree or equivalent, and originate from UE, UK or Switzerland.
To apply, please contact the advisors:
Olivier.Barais@irisa.fr, Paul.Temple@irisa.fr, Olivier.Zendra@inria.fr
(mailto:Olivier.Barais@irisa.fr,Paul.Temple@irisa.fr,Olivier.Zendra@inria.fr)

6. References

[1] Amine Benelallam, Nicolas Harrand, César Soto-Valero, Benoit Baudry, and Olivier Barais. “The maven dependency graph: a temporal graph-based representation of maven central”. In: 2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR). IEEE. 2019, pp. 344–348.

[2] Quentin Le Dilavrec, Djamel Eddine Khelladi, Arnaud Blouin, and Jean-Marc Jézéquel. “Untangling Spaghetti of Evolutions in Software Histories to Identify Code and Test Co-evolutions”. In: IEEE International Conference on Software Maintenance and Evolution, ICSME 2021, Luxembourg, September 27 - October 1, 2021. IEEE, 2021, pp. 206–216. doI: 10.1109/ICSME52107.2021.00025. URl: https://doi.org/10.1109/ICSME52107.2021. 00025.

[3] Jean-Rémy Falleri, Floréal Morandat, Xavier Blanc, Matias Martinez, and Martin Mon- perrus. “Fine-grained and accurate source code differencing”. In: ACM/IEEE Interna- tional Conference on Automated Software Engineering, ASE ’14, Vasteras, Sweden - September 15 - 19, 2014. Ed. by Ivica Crnkovic, Marsha Chechik, and Paul Grünbacher. ACM, 2014, pp. 313–324. doI: 10.1145/2642937.2642982. URl: https://doi.org/10.1145/2642937. 2642982.

[4] Mazharul Islam, Sazzadur Rahaman, Na Meng, et al. “Coding practices and recommen- dations of spring security for enterprise applications”. In: 2020 IEEE Secure Development (SecDev). IEEE. 2020, pp. 49–57.

[5] Djamel Eddine Khelladi, Benoît Combemale, Mathieu Acher, Olivier Barais, and Jean- Marc Jézéquel. “Co-evolving code with evolving metamodels”. In: ICSE ’20: 42nd In- ternational Conference on Software Engineering, Seoul, South Korea, 27 June - 19 July, 2020. Ed. by Gregg Rothermel and Doo-Hwan Bae.

ACM, 2020, pp. 1496–1508. doI: 10.1145/ 3377811.3380324. URl: https://doi.org/10.1145/3377811.3380324.

[6] Akond Rahman. “Anti-patterns in infrastructure as code”. In: 2018 IEEE 11th Interna- tional Conference on Software Testing, Verification and Validation (ICST). IEEE. 2018, pp. 434– 435.

[7] Marc Schönefeld. “Anti-patterns in JDK security and refactorings”. In: Detection of in- trusions and malware & vulnerability assessment, GI SIG SIDAR workshop, DIMVA 2004. Gesellschaft für Informatik eV. 2004.

[8] César Soto-Valero, Amine Benelallam, Nicolas Harrand, Olivier Barais, and Benoit Baudry. “The emergence of software diversity in maven central”. In: 2019 IEEE/ACM 16th Inter- national Conference on Mining Software Repositories (MSR). IEEE. 2019, pp. 333–343.

[9] Nikolaos Tsantalis, Ameya Ketkar, and Danny Dig. “RefactoringMiner 2.0”. In: IEEE Transactions on Software Engineering (2020). doI: 10.1109/TSE.2020.3007722.

[10] Fabio Pierrazzi, Feargus Pendlebury, Jacopo Cortellazzi, and Lorenzo Cavallaro. “Intriguing properties of adversarial ml attacks in the problem space”. In : 2020 IEEE symposium on security and privacy (SP). IEEE, 2020. p. 1332-1349.

[11] MITRE CVE: https://cve.mitre.org/index.html

[12] Software Heritage: https://www.softwareheritage.org/

[13] Campus Cyber: https://campuscyber.fr/

Last update: 14/04/2023 14:05