Université de Rennes

Direction de la Recherche et de l'Innovation

Appel à projets - Année 2025

Contrats doctoraux «Ordinaires»

ECOLE DOCTORALE : MATISSE **POLE DE RECHERCHE :** Université Bretagne Sud

Unité de recherche d'accueil

Unité de Recherche : UMR 6074 IRISA

Equipe : SecReizh, DiverSE

Encadrement

Directeur de thèse : Salah Sadou **Unité de recherche** : UMR 6074 IRISA

Employeur/Établissement d'origine :

HDR : ☒oui ☐ non **Nb de thèses en cours** : 4

Mail : salah.sadou@irisa.fr **Tél.** :

Co-directeur de thèse : Djamel Eddine Khelladi **Unité de recherche** : UMR 6074 IRISA

Employeur/Établissement d'origine :

HDR : ☐ oui ☒non **Nb de thèses en cours** : 4

Mail : djamel-eddine.khelladi@irisa.fr **Tél.** :

Sujet

Titre : Multi-Objective Debloating : from automatic vulnerability fixing to enhancing energy consumption, build and testing performances

Mots clés : Multi-Objective, Debloating

Cotutelle internationale (oui/non) : non **Si oui, préciser l'établissement et le pays de rattachement** :

Descriptif (15-30 lignes) :
Contexte socioéconomique et scientifique ; Les hypothèses et questions posées ; Les grandes étapes de la thèse et démarche ; Approches méthodologiques et techniques envisagées

Résumé du projet

When vulnerabilities are identified, applying their fixes may take time. However, for many real-world clients, it is not acceptable to simply keep using the vulnerable code, with its attack surface. We propose to analyze the usage of vulnerable code to help developers to reduce such usage, a process that we refer to as debloating. The principle of debloating is to remove portions of code with the goal of narrowing the attack surface to limit the effects of a vulnerability [1-4]. Debloating can take different forms and granularity: from removing a specific function in a single file to removing a high-level functionality spanning different concerns and files. Debloating can have a direct effect here by removing the vulnerable code.

Another targeted scenario of debloating is when the vulnerability is not precisely mitigated and understood. For example, vulnerabilities are not yet fixed in a given API. In this case, debloating an API would provide a sub-API that still meets the customers' other functional needs, but with a reduced code and surface attack.

Debloating can complicate the task of an attacker and reduce the damage incurred by vulnerable code for two reasons: i) debloating produces a new code variant and attackers have to design specific strategies, ideally for each debloated

variant of the code; ii) the attack over the vulnerable code is possible if and only if other parts of the code and entry points can be leveraged: if debloated, some attacks are simply not applicable.

Whereas extensive works exist on debloating in the security research field e.g., [22, 26, 4, 2, 1, 31, 24]), little work explores its benefits in the software engineering research field. Our hypothesis is that software debloating has also a potential in benefiting several software engineering activities, such as in a better modularization and splitting of large open-source projects, reducing their ecological footprint, and enhancing their build and testing in the continuous integration, etc. Thus, debloating becomes a multi-objective problem to address. However, to the best of our knowledge, no existing works have investigated this novel research perspective.

Description des objectifs et originalité de la thèse

Beyond the surface attack, one must also ensure that the original functional specification is not impacted entirely, or at least control and identify the impacts so that developers can perform informative decisions on how to use the debloated software. Moreover, the security benefit of debloating can also be coupled with performance benefit. State of the art does not investigate the benefit of debloating on non-functional properties, such as on build and tests execution in the continuous integration (CI) or the energy consumption of the debloated software. Hence, debloating becomes a multi-objective problem to solve, which to the best of our knowledge has never been addressed.

Therefore, the goal of MODeb is to not only aim to fix software vulnerabilities but also to explore the benefits of debloating in software engineering with a series of implementations and experiments.

Description des principaux verrous et techniques envisagées

We will realize automatic debloating tools operating over source code. The tools will produce debloated versions of applications and libraries that other projects depend on. We plan to leverage databases of vulnerabilities as well as associated patches and tests (based on existing vulnerability databases) to i) guide the debloating strategy (i.e., what to remove in the code); ii) validate and characterize to what extent the system is less vulnerable. Moreover, we will investigate the effect of debloating on build and test execution in the CI and on the energy consumption of the debloated software variants. We can orient our effort towards 1) libraries and Web applications (e.g., written in JavaScript) and 2) libraries and clients in Java projects on Maven central. We will leverage on state of the art multi-objective algorithms (e.g., NSGA III or others) to implement automatic solutions and we will evaluate them with state of the art methods and tools. We will rely on the known use cases of vulnerabilities and their fixes to compare to our debloating solution.

Approche méthodologique et critères de qualité des résultats obtenus

We will focus on debloating large open source projects in GitHub based on two scenarios, namely based on: 1) their commit history and contributors, 2) their clients' usages, i.e., what functionalities other projects in GitHub uses from the open source projects. We will reuse know datasets of GitHub project with detected vulnerabilities and CVEs.

As measuring and understanding the energy consumption of software is a growing concern for Green IT in terms of CPU usage, frequency (GHz), temperature (ºC), and power (W) [19, 21, 8, 12, 7]. The goal is then to set up different strategies to reduce and optimize the energy consumption of software (e.g., [10, 15, 5, 20]). In this task, we will explore this concern from a novel perspective, that is the impact of debloating on the energy consumption of the debloated software projects. To do so, as a first step, we will leverage RAPL [13], PowerAPI/ Jouleit and Intel Power Gadget/ PowerLog tools, as they have been successfully used to provide precise measurements of energy consumption. As a second step, we will extend and vary the techniques to measure the energy consumption to have a better confidence in our results. We will leverage other existing known tools, such as Powerstat, PowerTOP, Perf, Likwid.

Moreover, as continuous integration has been reported to be costly in terms of building new versions of code, and running a large set of unit and acceptance tests [27, 23].

Herein, we will investigate the impact of the debloated software on the performance of the continuous integration in terms of build and testing. In particular, we will measure several metrics, namely build results, build duration, time to run the tests, test pass rate, time to fix tests, numbers of failed deployments, etc.

Our evaluation methodology will be to perform the measurements for the original software and the debloated software (i.e., sub-projects) to compare their energy consumption and their build and testing performances with the fixing of vulnerabilities. After that, we will also debloat the same software over different versions (e.g., releases) to monitor the progression of vulnerabilities, the energy consumption and their build and testing performances over time.

[1] Aatira Anum Ahmad, Abdul Rafae Noor, Hashim Sharif, Usama Hameed, Shoaib Asif, Mubashir Anwar, Ashish Gehani, Junaid Haroon Siddiqui, and Fareed M Zaffar. Trimmer : An automated system for configuration-based software debloating. IEEE Transactions on Software Engineering, 2021.

[2] Babak Amin Azad, Pierre Laperdrix, and Nick Nikiforakis. Less is more : quantifying the security benefits of debloating web applications. In 28th USENIX Security Symposium (USENIX Security 19), pages 1697–1714, 2019.

[3] João Ferreira Filho Bosco, Mathieu Acher, and Olivier Barais. Software Unbundling : Challenges and Perspectives. In S. Chiba, M. Südholt, P. Eugster, L. Ziarek, and G.T. Leavens, editors, Transactions on Modularity and Composition I. Springer, May 2016.

[4] Michael D Brown and Santosh Pande. Is less really more ? towards better metrics for measuring security improvements realized through software debloating. In 12th USENIX Workshop on Cyber Security Experimentation and Test (CSET 19), 2019.

[5] Bobby R Bruce, Justyna Petke, and Mark Harman. Reducing energy consumption using genetic improvement. In Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation, pages 1327–1334, 2015.

[6] Bobby R Bruce, Tianyi Zhang, Jaspreet Arora, Guoqing Harry Xu, and Miryung Kim. Jshrink : In-depth investigation into debloating modern java applications. In Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, pages 135–146, 2020.

[7] Tiago Carçao. Measuring and visualizing energy consumption within software code. In 2014 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC), pages 181–182. IEEE, 2014.

[8] Shaiful Alam Chowdhury and Abram Hindle. Greenoracle : Estimating software energy consumption with energy measurement corpora. In 2016 IEEE/ACM 13th Working Conference on Mining Software Repositories (MSR), pages 49–60. IEEE, 2016.

[9] Jonas Gamalielsson and Björn Lundell. Sustainability of open source software communities beyond a fork: How and why has the libreoffice project evolved ? Journal of Systems and Software, 89 :128–145, 2014.

[10] Édouard Guégain, Clément Quinton, and Romain Rouvoy. On reducing the energy consumption of software product lines. In Proceedings of the 25th ACM International Systems and Software Product Line Conference-Volume A, pages 89–99, 2021.

[11] Mariam Guizani, Amreeta Chatterjee, Bianca Trinkenreich, Mary Evelyn May, Geraldine J Noa-Guevara, Liam James Russell, Griselda G Cuevas Zambrano, Daniel Izquierdo-Cortazar, Igor Steinmacher, Marco A Gerosa, et al. The long road ahead : Ongoing challenges in contributing to large oss organizations and what to do. Proceedings of the ACM on Human-Computer Interaction, 5(CSCW2) :1–30, 2021.

[12] Syed Islam, Adel Noureddine, and Rabih Bashroush. Measuring energy footprint of software features. In 2016 IEEE 24th International Conference on Program Comprehension (ICPC), pages 1–4. IEEE, 2016.

[13] Kashif Nizam Khan, Mikael Hirki, Tapio Niemi, Jukka K Nurminen, and Zhonghong Ou. Rapl in action : Experiences in using rapl for power measurements. ACM Transactions on Modeling and Performance Evaluation of Computing Systems (TOMPECS), 3(2) :1–26, 2018.

[14] Hyungjoon Koo, Seyedhamed Ghavamnia, and Michalis Polychronakis. Configuration-driven software debloating. In Proceedings of the 12th European Workshop on Systems Security, pages 1–6, 2019.

[15] Young-Woo Kwon and Eli Tilevich. Reducing the energy consumption of mobile applications behind the scenes. In 2013 IEEE International Conference on Software Maintenance, pages 170–179. IEEE, 2013.

[16] Konner Macias, Mihir Mathur, Bobby R Bruce, Tianyi Zhang, and Miryung Kim. Webjshrink : a web service for debloating java bytecode. In Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, pages 1665–1669, 2020.

[17] Anderson S Matos, Joao B Ferreira Filho, and Lincoln S Rocha. Splitting apis : An exploratory study of software unbundling. In 2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR), pages 360–370. IEEE, 2019.

[18] Mathias Meyer. Continuous integration and its tools. IEEE software, 31(3) :14–16, 2014.

[19] Adel Noureddine, Romain Rouvoy, and Lionel Seinturier. Unit testing of energy consumption of software libraries. In Proceedings of the 29th Annual ACM Symposium on Applied Computing, pages 1200–1205, 2014.

[20] Zakaria Ournani, Romain Rouvoy, Pierre Rust, and Joel Penhoat. On reducing the energy consumption of software : From hurdles to requirements. In Proceedings of the 14th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM), pages 1–12, 2020.

[21] Candy Pang, Abram Hindle, Bram Adams, and Ahmed E Hassan. What do programmers know about software energy consumption ? IEEE Software, 33(3) :83–89, 2015.

[22] Pardis Pashakhanloo, Aravind Machiry, Hyonyoung Choi, Anthony Canino, Kihong Heo, Insup Lee, and Mayur Naik. Pacjam : Securing dependencies continuously via package-oriented debloating. 2022.

[23] Gustavo Pinto, Fernando Castor, Rodrigo Bonifacio, and Marcel Rebouças. Work practices and challenges in continuous integration : A survey with travis ci users. Software : Practice and Experience, 48(12) :2223–2236, 2018.

[24] Serena Elisa Ponta, Wolfram Fischer, Henrik Plate, and Antonino Sabetta. The used, the bloated, and the vulnerable : Reducing the attack surface of an industrial application. In 2021 IEEE International Conference on Software Maintenance and Evolution (ICSME), pages 555–558. IEEE, 2021.

[25] Chris Porter, Girish Mururu, Prithayan Barua, and Santosh Pande. Blankit library debloating : Getting what you want instead of cutting what you don't. In Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation, pages 164–180, 2020.

[26] Chenxiong Qian, Hyungjoon Koo, ChangSeok Oh, Taesoo Kim, and Wenke Lee. Slimium : Debloating the chromium browser with feature subsetting. In Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security, pages 461–476, 2020.

[27] Mojtaba Shahin, Muhammad Ali Babar, and Liming Zhu. Continuous integration, delivery and deployment : a systematic review on approaches, tools, challenges and practices. IEEE Access, 5 :3909–3943, 2017.

[28] César Soto-Valero, Thomas Durieux, and Benoit Baudry. A longitudinal analysis of bloated java dependencies. In Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, pages 1021–1031, 2021.

[29] César Soto-Valero, Nicolas Harrand, Martin Monperrus, and Benoit Baudry. A comprehensive study of bloated dependencies in the maven ecosystem. Empirical Software Engineering, 26(3) :1–44, 2021.

[30] Yutian Tang, Hao Zhou, Xiapu Luo, Ting Chen, Haoyu Wang, Zhou Xu, and Yan Cai. Xdebloat : Towards automated feature-oriented app debloating. IEEE Transactions on Software Engineering, 2021.

[31] Renjun Ye, Liang Liu, Simin Hu, Fangzhou Zhu, Jingxiu Yang, and Feng Wang. Jslim : Reducing the known vulnerabilities of javascript application by debloating. In International Symposium on Emerging Information Security and Applications, pages 128–143. Springer, 2021.

[32] Yunwen Ye and Kouichi Kishida. Toward an understanding of the motivation of open source software developers. In 25th International Conference on Software Engineering, 2003. Proceedings., pages 419–429. IEEE, 2003.