Investigating Next Steps in Static API-Misuse Detection

Sven Amann[1] Hoan Anh Nguyen[2] Sarah Nadi[3] Tien N. Nguyen[4] Mira Mezini[5]

Keywords: API Misuse, Constraint Mining, Machine Learning, Bug Detection

Incorrect usages of an Application Programming Interface (API), or *API misuses*, are violations of (implicit) *usage constraints* of the API. An example of a usage constraint is having to check that `hasNext()` returns `true` before calling `next()` on an `Iterator`, in order to avoid a `NoSuchElementException` at runtime. API misuse is a prevalent cause of software bugs, crashes, and vulnerabilities [Na16, Am16].

To mitigate API misuse, researchers have proposed several *API-misuse detectors* [WZ11, MM13, Ng15]. These detectors analyze *API usages*, i.e., code snippets that use a given API. The detectors commonly mine *usage patterns*, i.e., equivalent API usages that occur frequently, and then report deviations from these patterns as potential misuses. Unfortunately, the reported precision of such detectors is typically low and a recent study [Am18] showed that their recall is also very low. Thus, we need better detectors to address the still-prevalent problem of API misuse [Le16, Ac16].

Previous work identified individual as well as common strengths and weaknesses of existing detectors [Am18] in an empirical study using the open-source benchmark MuBench [MU17]. In this paper, we investigate whether addressing the reported weaknesses indeed leads to better performance in practice. Therefore, we design a new misuse detector, MuDetect. MuDetect encodes API usages as API-Usage Graphs (AUGs), a comprehensive usage representation that captures different types of API misuses. MuDetect employs a greedy, frequent-subgraph-mining algorithm to mine patterns and a specialized graph-matching strategy to identify pattern violations. Both components consider code semantics to improve the overall detection capabilities. On top, MuDetect uses an empirically optimized ranking strategy to effectively rank true positives. While previous detectors mostly target a per-project setting [Am18], MuDetect also works in a cross-project setting, where it mines thousands of usage examples from third-party projects.

We assess the precision and recall of MuDetect and show that it outperforms the four state-of-the-art detectors evaluated in prior work [Am18]. In our evaluation, we extended MuBench

[1] CQSE GmbH, Centa-Hafenbrädl-Straße 59, 81249 München, Germany, amann@cqse.eu
[2] Amazon Research, hoan@iastate.edu
[3] University of Alberta, 4-41 Athabasca Hall, Canada, nadi@ualberta.ca
[4] The University of Texas at Dallas, 800 W. Campbell Road, ECSS 4.229 Richardson, TX 75080-3021, USA, tien.n.nguyen@utdallas.edu
[5] Technische Universität Darmstadt, Hochschulstraße 10, 64289 Darmstadt, mezini@cs.tu-darmstadt.de

doi:10.18420/SE2020_32

by 107 real-world misuses identified in a recent study on run-time verification [Le16]—more than doubling its size—to ensure that our design decisions generalize. We show that, in a setting with perfect training data, MuDetect achieves a recall of 72.5%, which is 20.3% higher than the next best detector and over 50% higher than the other detectors. In the typical per-project setting, MuDetect achieves recall of 20.9%, which is 10.2% better than the second-best detector, and precision of 21.9%, which is 13.1% better than the second-best detector. In a cross-project setting, MuDetect's recall and precision again improve significantly to 42.2% and 33.0%, respectively. Throughout the experiments, MuDetect identified 27 previously unknown misuses, which we reported in eight pull requests (PRs). To date, three of the PRs got accepted, demonstrating that MuDetect identifies actual issues in current software projects.

We publish our MuBench extension, MuDetect's implementation, and all experiment data, tooling, and results [Ar19].

Bibliography

[Ac16] Acar, Yasemin; Backes, Michael; Fahl, Sascha; Kim, Doowon; Mazurek, Michelle L.; Stransky, Christian: You Get Where You're Looking For. The Impact of Information Sources on Code Security. In: Proceedings of the 37th IEEE Symposium on Security and Privacy. IEEE Computer Society Press, 2016.

[Am16] Amann, Sven; Nadi, Sarah; Nguyen, Hoan A.; Nguyen, Tien N.; Mezini, Mira: MUBench: A Benchmark for API-Misuse Detectors. In: Proceedings of the 13th Working Conference on Mining Software Repositories. MSR '16. ACM Press, 2016.

[Am18] Amann, Sven; Nguyen, Hoan A.; Nadi, Sarah; Nguyen, Tien N.; Mezini, Mira: A Systematic Evaluation of Static API-Misuse Detectors. IEEE Transactions on Software Engineering, 2018.

[Ar19] Artifacts: , http://www.st.informatik.tu-darmstadt.de/artifacts/mudetect/, 2019.

[Le16] Legunsen, Owolabi; Hassan, Wajih Ul; Xu, Xinyue; Roşu, Grigore; Marinov, Darko: How Good Are the Specs? A Study of the Bug-finding Effectiveness of Existing Java API Specifications. In: Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering. ASE '16. ACM Press, pp. 602–613, 2016.

[MM13] Monperrus, Martin; Mezini, Mira: Detecting Missing Method Calls as Violations of the Majority Rule. ACM Transactions on Software Engineering and Methodology, 22(1):1–25, 2013.

[MU17] MUBench: , https://github.com/stg-tud/MUBench/, 2017.

[Na16] Nadi, Sarah; Krüger, Stefan; Mezini, Mira; Bodden, Eric: "Jumping Through Hoops": Why do Developers Struggle with Cryptography APIs? In: Proceedings of the 38th International Conference on Software Engineering. ICSE'16. ACM Press, 2016.

[Ng15] Nguyen, T. T.; Pham, H. V.; Vu, P. M.; Nguyen, T. T.: Recommending API Usages for Mobile Apps with Hidden Markov Model. In: Proceedings of the 30th ACM/IEEE International Conference on Automated Software Engineering. ASE '15. IEEE Computer Society Press, pp. 795–800, 2015.

[WZ11] Wasylkowski, Andrzej; Zeller, Andreas: Mining Temporal Specifications from Object Usage. Automated Software Engineering, 18(3-4):263–292, 2011.