UpCy: Safely Updating Outdated Dependencies (Summary)

Andreas Dann [1], Ben Hermann [2], and Eric Bodden [3]

Keywords: Security maintenance, Open-Source Software, Security Vulnerabilities

Introduction The use of open-source software (OSS) in Java applications is prevalent [Ku18; Wa20]. However, vulnerabilities like Log4Shell have emphasized the importance of updating OSS quickly. Manually finding compatible updates is challenging, especially for transitive dependencies. However, developers are hesitant to update and skeptical of tools like Dependabot since they fear introducing incompatibilities that break their application [BKH15; HG22; Ku18; MP17; Wa20]. To address this concern, UpCy[4] proposes updates with minimal incompatibilities by analyzing how the application uses a dependency. It utilizes the min-(s,t)-cut algorithm and a graph database of Maven Central.

Compatible Updates When updating dependencies, developers must be cautious to avoid breaking existing functionality. The dependency's location in the application's dependency tree determines the options for updating. For every update, developers must ensure that the app still compiles, links with other dependencies, and executes successfully. Additionally, dependencies belonging to one framework must be updated consistently.

UpCy's Design We created UpCy to help find OSS dependency updates for Java apps with minimum incompatibilities. It computes a min-(s,t)-cut on the app's dependency graph and runs queries against a full Maven Central dependency graph. As an output, developers get a set of OSS to update to the target version. UpCy executes the following steps:

First, UpCy builds the app's *unified dependency graph* to identify which parts of the OSS's API the app uses. The unified dependency graph combines the apps's dependency graph with its call graph, created with the static analysis framework Soot [Va10].

Second, UpCy checks if a naïve update (simply increasing the version number of the OSS—as applied by state-of-the-art tools) leads to source-code or binary incompatibilities.

Third, if the naïve update yields incompatibilities, UpCy tries to identify updates of a (set of) dependencies with fewer incompatibilities. To do this, UpCy computes the minimal number of compatibilities an update has to fulfill by computing min-(s,t)-cuts on the undirected unified dependency graph with the app $s \in S$ and the vulnerable OSS $t \in T$. Since each edge

1 CodeShield GmbH, Paderborn, Germany,
andreas.dann@uni-paderborn.de, https://orcid.org/0000-0002-6587-7431
2 Technical University Dortmund, Dortmund, Germany,
ben.hermann@cs.tu-dortmund.de, https://orcid.org/0000-0001-9848-2017
3 Heinz Nixdorf Institute & Fraunhofer IEM, Paderborn, Germany,
eric.bodden@uni-paderborn.de, https://orcid.org/0000-0003-3470-3647
4 The full paper is available online: 10.1109/ICSE48619.2023.00031

doi:10.18420/sw2024_36

in the graph represents a compatibility requirement, the min-(s,t)-cut represents the number of compatibilities that may be violated if dependencies in T are updated. For each found min-cut, UpCy queries our Neo4j graph database of Maven Central to find compatible and non-vulnerable versions of the dependencies in T by mapping the compatibility requirements to Neo4j's query language Cypher.

Fourth, if the graph database returns updates, UpCy assesses the changes in the app's dependency tree and computes which API calls in the unified dependency graph may suffer from incompatibilities if the update is applied.

Evaluation A thorough evaluation of 29,698 updates across 380 projects shows that UpCy consistently generated fewer incompatibilities than other state-of-the-art tools. Impressively, 70.1% of the updates generated by UpCy resulted in zero incompatibilities.

Data Availability UpCy and our evaluation are available on GitHub `https://github.com/secure-software-engineering/upcy` and Zenodo `https://doi.org/10.5281/zenodo.7037673`.

References

[BKH15] Bogart, C.; Kästner, C.; Herbsleb, J.: When It Breaks, It Breaks: How Ecosystem Developers Reason about the Stability of Dependencies. In: 30th IEEE/ACM International Conference on Automated Software Engineering Workshop (ASEW). Pp. 86–89, 2015, URL: https://doi.org/10.1109/ASEW.2015.21.

[HG22] Hejderup, J.; Gousios, G.: Can we trust tests to automate dependency updates? A case study of Java Projects. Journal of Systems and Software 183/, 2022, URL: https://doi.org/10.1016/j.jss.2021.111097.

[Ku18] Kula, R. G.; German, D. M.; Ouni, A.; Ishio, T.; Inoue, K.: Do developers update their library dependencies? Empirical Software Engineering 23/1, pp. 384–417, 2018, URL: https://doi.org/10.1007/s10664-017-9521-5.

[MP17] Mirhosseini, S.; Parnin, C.: Can automated pull requests encourage software developers to upgrade out-of-date dependencies? 32nd IEEE/ACM International Conference on Automated Software Engineering/, pp. 84–94, 2017, URL: https://dl.acm.org/citation.cfm?id=3155577.

[Va10] Vallée-Rai, R.; Co, P.; Gagnon, E.; Hendren, L.; Lam, P.; Sundaresan, V.: Soot: A Java Bytecode Optimization Framework. In: CASCON First Decade High Impact Papers. CASCON '10, IBM Corp., Toronto, Ontario, Canada, pp. 214–224, 2010, URL: https://doi.org/10.1145/1925805.1925818.

[Wa20] Wang, Y.; Chen, B.; Huang, K.; Shi, B.; Xu, C.; Peng, X.; Wu, Y.; Liu, Y.: An Empirical Study of Usages, Updates and Risks of Third-Party Libraries in Java Projects. 2020 IEEE International Conference on Software Maintenance and Evolution, ICSME 2020/, pp. 35–45, 2020, URL: https://doi.org/10.1109/ICSME46990.2020.00014.