A Contextualised Risk-Based Framework for Maven Dependency Updates

David Rodrigues Almeida
Dept. of Informatics Engineering
School of Engineering of the Technical University of Porto
Porto, Portugal
1181210@isep.ipp.pt

Isabel Azevedo
Dept. of Informatics Engineering
School of Engineering of the Technical University of Porto
Porto, Portugal
ifp@isep.ipp.pt

***Abstract*—Modern software systems rely heavily on dependencies distributed through Maven Central, and vulnerabilities frequently reside in transitive rather than direct dependencies. Tools such as OWASP Dependency-Check are effective at flagging these known vulnerabilities, yet a large share of the flagged code is never actually executed by the depending application, leaving developers with a binary choice whenever a vulnerability is reported: upgrade immediately and risk breaking the build, or postpone the update and accept the security exposure. This paper presents a Contextualised Risk-Based Framework (CRBF) that supports Maven dependency-update decisions by combining exploitability, reachability, ecosystem stability and compatibility signals into a single risk score, and that selects an optimal remediation plan under a configurable effort budget. CRBF is implemented as a Maven plugin with a hexagonal architecture: known vulnerabilities are retrieved from the OSV database and enriched with EPSS exploitation scores; a static call graph built with WALA determines whether the vulnerable code is reachable; binary compatibility is assessed with JApiCmp; and ecosystem adoption and maintenance indicators are retrieved from the Goblin Weaver dependency graph. These signals are aggregated into a contextual risk score, and a Z3 SMT solver selects the subset of upgrades that maximises risk reduction within the effort budget. The framework has been implemented and exercised on individual Maven projects; a comparative evaluation against OSV-Scanner, with and without call-graph analysis, across a broader sample of open-source projects is under way. By turning severity, exploitability, reachability, stability and compatibility into a single, effort-bounded recommendation, CRBF offers developers an alternative to the current binary security-versus-stability decision.**

Keywords—Maven, Dependency Management, Software Supply Chain Security, Reachability Analysis, Risk-Based Framework

I. Introduction

Modern Java applications rely extensively on third-party libraries distributed through Maven Central. While the use of external dependencies simplifies software development, it can also propagate vulnerabilities through dependency chains, particularly through transitive dependencies [1]. Conventional dependency scanners can identify known vulnerable versions, but the presence of a vulnerable dependency alone does not establish whether the vulnerability is relevant to a particular application. Many library methods are never invoked by their dependents [2], highlighting the importance of project-specific usage in vulnerability assessment.

Remediation, however, is not risk-free. Updating a vulnerable dependency may introduce breaking changes, even when versioning conventions suggest compatibility [3], while postponing remediation leaves the known vulnerability unaddressed. Developers must therefore balance the security benefit of an update against its potential impact on application stability and the effort required to adopt it. Existing approaches address individual aspects of this decision, including reachability, compatibility, ecosystem stability, and update cost [4], [5], [6], [7]. However, the reviewed approaches integrate only subsets of these dimensions rather than combining them into a single contextualised dependency-update decision process.

To address this gap, this paper presents CRBF, designed to support dependency-update decisions in Maven projects by combining exploitability, static reachability, ecosystem stability, and compatibility information. CRBF produces a contextual risk assessment for each vulnerability and estimates the effort associated with candidate remediations. It then formulates remediation as a constrained optimisation problem, selecting the set of dependency updates that maximises risk reduction within a configurable effort budget. This work addresses two Research Questions.

RQ1: *How can a novel risk-based framework be designed and implemented to automate dependency update decisions for Maven projects by integrating vulnerability data, ecosystem stability metrics, and project-specific usage context?* RQ1 examines how the different dimensions identified in the literature can be integrated into a single decision process for Maven dependency updates. We addressed RQ1 by designing and implementing CRBF as a Maven plugin that combines exploitability, reachability, ecosystem stability, and compatibility information.

RQ2: *To what extent does the proposed framework's context-aware guidance improve the effectiveness of dependency management compared to non-contextual vulnerability reports?* RQ2 evaluates the effect of adding contextual information to vulnerability reports. We investigate RQ2 by comparing OSV-Scanner without call-graph analysis, OSV-Scanner with call-graph analysis, and CRBF, allowing us to examine the effects of reachability and CRBF's additional contextual signals separately.

The remainder of this paper reviews related work, presents the design of CRBF, including its risk-scoring and remediation models, describes its implementation, outlines the evaluation methodology and current experimental setup, and concludes with the main contributions and directions for future work.

II. Related Work

Existing approaches to dependency vulnerability management broadly focus either on identifying relevant vulnerabilities or on supporting update decisions through compatibility, ecosystem, or optimisation information.

Conventional Software Composition Analysis (SCA) tools identify vulnerable dependencies by matching resolved package versions against vulnerability databases [8].

However, dependency-level detection does not establish whether vulnerable code is actually relevant to the analysed application. Keshani et al. [2] found that 87% of publicly documented methods in Maven libraries were never invoked by their dependents, showing that package-level exposure alone is an insufficient basis for application-specific risk assessment. Reachability-aware approaches address this limitation by determining whether execution paths from the application can reach vulnerable code. Zhao et al. [9], for example, reported that 105 of 145 detected vulnerabilities were false positives after incorporating reachability information, corresponding to a 72% precision improvement over unfiltered package-level scanning. OSV-Scanner v2 [10] extends conventional dependency scanning with Java call analysis, adding reachability information to the vulnerabilities it reports.

Vulnerability severity provides limited information about real-world exploitation risk. Jacobs et al. [11] showed that exploitation likelihood offers a distinct prioritisation signal, motivating the use of the Exploit Prediction Scoring System (EPSS) alongside severity-based metrics.

Reachability alone does not determine whether an update should be applied. The target version may introduce incompatibilities, and version numbers provide limited assurance of upgrade safety. Raemaekers et al. [3] analysed over 100,000 JAR files and found that 35.7% of minor releases and 23.8% of patch releases introduced binary incompatibilities, despite Semantic Versioning expectations. Compatibility information can therefore help estimate the effort and risk associated with a candidate update. Ecosystem-level information provides another perspective: the Goblin framework models the historical Maven dependency graph, allowing adoption and stability metrics to be derived from how library versions are used across the ecosystem [4]. Together, these signals complement reachability by considering not only whether vulnerable code is exposed, but also the potential impact of replacing the affected dependency.

Beyond assessing the risk of an update, previous work has also considered its adoption cost and the selection of suitable remediation actions. Huang et al. [7] proposed LIBSECURIFY, which combines reachability information with an estimate of update effort based on API changes. In their evaluation, 24 of the 33 projects that received contextualised alerts subsequently updated their vulnerable dependencies. Other approaches formulate dependency management as an optimisation problem. Depex [5] encodes dependency and version constraints as a Satisfiability Modulo Theories (SMT) problem to identify compatible dependency configurations, while GoblinUpdater [6] uses multi-objective optimisation to balance freshness, popularity, vulnerability information, and adaptation cost. However, GoblinUpdater evaluates vulnerabilities at the package level and does not consider whether the vulnerable code is reachable from the analysed application [6].

Existing approaches therefore address different parts of the dependency-update problem. Reachability-aware tools improve the relevance of vulnerability reports [9], [10], while other approaches incorporate compatibility, ecosystem information, update effort, or optimisation into the decision problem [4], [7]. However, these dimensions are typically considered separately or only in partial combinations. CRBF integrates exploitation likelihood, project-specific reachability, ecosystem stability, compatibility-based update effort, and constrained remediation within a single framework for Maven dependency-update decisions. The following section describes how these elements are organised within the proposed framework.

III. Framework Design

CRBF adopts a Hexagonal Architecture to separate the dependency-risk analysis from external tools and services [12]. The Domain and Application layers form the core of the framework, while interactions with external systems are defined through ports and implemented by adapters. The Maven plugin acts as the entry point of the framework, invoking the dependency-risk analysis through the application layer. Outbound ports provide access to vulnerability and exploitation data, reachability analysis, ecosystem metrics, compatibility analysis, remediation optimisation, and report generation. The separation keeps the core independent of specific external technologies and allows individual integrations to be replaced without modifying the core logic. Fig. 1 presents the main components and their interactions.

A. Risk Scoring Model

CRBF assigns a contextual risk score to each vulnerable dependency using Simple Additive Weighting (SAW), followed by project-specific multiplicative factors. To ensure that heterogeneous signals are combined consistently, all inputs are normalised to the interval [0,1]. Metrics with a bounded scale are normalised against their upper bound, while metrics without a natural upper bound use a saturation threshold.

Staleness quantifies how outdated the current dependency version is, based on the time elapsed since its release and the number of subsequent versions available. It is computed as:

$$Staleness = 0.60\ TimeLag_{\text{norm}} + 0.40\ VersionLag_{\text{norm}} \quad (1)$$

Here, $TimeLag_{\text{norm}} = min(lagDays / 365, 1)$ represents the time elapsed since the current version was released, normalised against a one-year threshold, and $VersionLag_{\text{norm}} = min(versionLag / 10, 1)$ represents the number of subsequent releases, normalised against a threshold of ten releases. Time lag receives a higher weight because elapsed time provides a more consistent indication of accumulated exposure, whereas version lag depends on each project's release frequency.

The base risk then combines vulnerability severity, exploitation likelihood and staleness:

$$R_b = 0.50\ CVSS_{norm} + 0.30\ EPSS + 0.20\ Staleness \quad (2)$$

where $CVSS_{norm} = max_{cvss} / 10$ represents the severity of the most severe vulnerability affecting the dependency, while *EPSS* is already defined in [0,1]. CVSS is assigned the highest weight because it represents the intrinsic severity of the vulnerability and remains the primary severity signal. EPSS complements severity with the probability of real-world exploitation, while staleness receives the smallest weight because it is an indirect indicator of accumulated risk rather than evidence of a specific threat. When EPSS is unavailable for a vulnerability, the missing value is not treated as zero. Instead, the CVSS and staleness weights are renormalised to preserve their relative contribution and sum to 1.0. These weights are initial design values and require further empirical calibration.

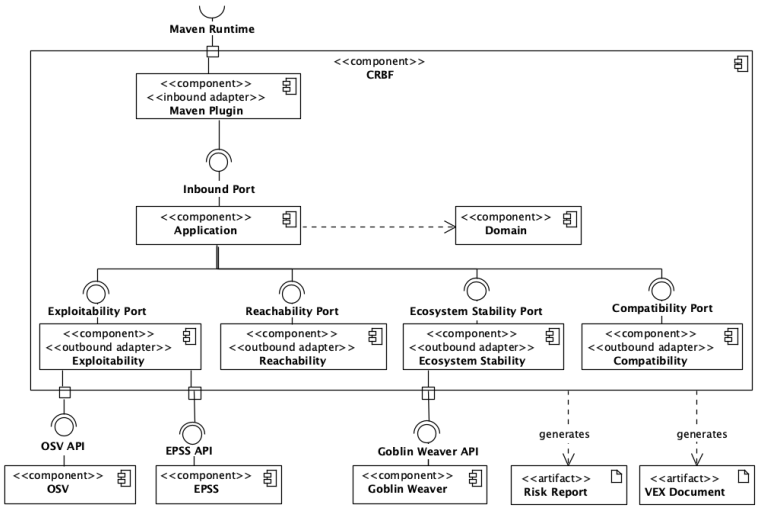


Fig. 1. CRBF architectural overview

The base risk score is then adjusted using contextual factors specific to the analysed project and the proposed remediation:

$$R_c = R_b \times W_r \times F_p \tag{3}$$

Here R_c is the contextual risk, R_b the base risk, W_r the reachability weight, and F_p the path-quality factor.

The reachability weight reflects whether the vulnerable code is reachable from the analysed application. Reachable and probably reachable vulnerabilities retain a weight of 1.0, while unknown and unreachable cases receive weights of 0.5 and 0.1, respectively. The non-zero weight assigned to unreachable cases accounts for the limitations of static call-graph analysis, which cannot guarantee complete coverage of all execution paths.

The path-quality factor captures how the proposed update changes transitive vulnerability risk. It ranges from 0.3 to 1.0, with higher values assigned to updates that reduce transitive risk and lower values to those that introduce additional vulnerable dependencies. The lower bound of 0.3 ensures that resolving a known vulnerability retains some value even when the proposed update introduces additional transitive risk.

For each candidate fix, CRBF estimates ecosystem stability based on the target version's adoption rate, maintenance activity and adoption lifespan;

$$StabilityScore = 0.55\ AdoptionRate + 0.30\ Maintenance_{norm} + 0.15\ Lifespan_{norm} \tag{4}$$

Here, *AdoptionRate* represents the proportion of dependent projects using the candidate fix version; $Maintenance_{norm}$ the normalised release frequency; and $Lifespan_{norm}$ how long the candidate version has remained in active use, normalised against a two-year threshold.

Adoption rate receives the highest weight because it provides direct evidence of ecosystem use, while maintenance activity and adoption lifespan provide complementary indications of continued support and temporal stability.

B. Remediation Optimisation

CRBF formulates remediation as a constrained optimisation problem in which each candidate update is characterised by its contextual risk score and estimated upgrade effort. The effort cost is derived from the compatibility assessment using a relative scale: compatible updates are assigned 1 effort unit, unknown compatibility 2, source-incompatible updates 3, and binary-incompatible updates 4. Candidates without an available fix are excluded from the optimisation.

Given a configurable effort budget, the framework uses the Z3 SMT optimiser to select the subset of candidate updates that maximises cumulative risk reduction without exceeding the available effort. For each candidate i, a binary decision variable x_i indicates whether the update is selected, while R_i represents its contextual risk score and C_i its effort cost. The optimisation objective is:

$$Maximise \sum_{i=1}^{n} x_i R_i \tag{5}$$

The objective is subject to an effort-budget constraint, which ensures that the cumulative cost of the selected updates does not exceed the configured budget:

$$\sum_{i=1}^{n} x_i C_i \leq B \tag{6}$$

Here, B is the configured effort budget. Two additional constraints are applied: a candidate without an available fix cannot be selected, while mandatory updates must be included in the remediation plan. A candidate is considered mandatory when the vulnerability is critical, reachable, and has an available fix.

IV. Implementation

CRBF is implemented as a Maven plugin, allowing analysis to run directly from a Maven project. The plugin resolves the project's dependency graph and coordinates the analysis pipeline required by the framework. OSV and EPSS provide vulnerability and exploitation data, WALA performs static reachability analysis, JApiCmp assesses compatibility between the current and candidate versions, Goblin Weaver provides ecosystem metrics and dependency-graph information, and Z3 solves the remediation optimisation problem. Fig. 2 summarises the main stages of the CRBF analysis pipeline.

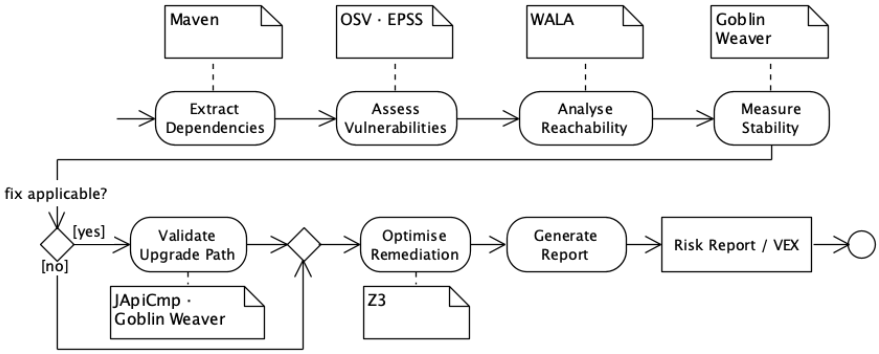


Fig. 2. CRBF analysis pipeline

After resolving the dependency graph and identifying vulnerable dependencies, CRBF enriches each vulnerability with exploitation and reachability information and retrieves the corresponding ecosystem metrics. For vulnerabilities with an available fix, the framework evaluates the candidate version through compatibility analysis and a transitive dependency comparison to determine whether the update introduces additional vulnerable dependencies. The resulting remediation candidates are scored according to the contextual risk model and then evaluated by the optimiser to determine the final update plan. The analysis produces two outputs: a contextualised risk report, exported in JSON and HTML formats, and a VEX document that provides a standardised, machine-readable representation of the exploitability status of each identified vulnerability.

V. Evaluation

To evaluate RQ2, a comparative experiment was designed using OSV-Scanner with and without call analysis and CRBF. The evaluation is still ongoing and currently includes three open-source Maven projects, with additional projects to be included. This section describes the experimental setup, dataset selection, and execution procedure.

A. Experimental Setup

The experiment compares three configurations. C1 uses OSV-Scanner v2 without call analysis and serves as the non-contextual baseline, reporting vulnerabilities associated with the dependencies present in the project. C2 uses the same scanner with Java call analysis enabled, adding reachability information to the vulnerabilities reported in C1. C3 uses CRBF, which incorporates reachability together with exploitation likelihood, ecosystem stability, binary compatibility, and remediation optimisation. The comparison isolates the effect of reachability (C1-C2) and the additional context introduced by CRBF (C2-C3).

B. Dataset and Execution

The evaluation dataset consists of public open-source Maven projects hosted on GitHub and screened according to a set of technical criteria. Each project must have a valid Maven configuration, build successfully without source-code modifications, produce an executable JAR, and contain known vulnerable dependencies. To remain within the evaluation scope defined for the framework, projects are also limited to at most 100 resolved direct and transitive dependencies. To increase dataset diversity, the selection aims to include projects with varying dependency complexity within this limit and different levels of project maturity. A fixed public commit is used for each project to ensure that the source code and dependency graph remain unchanged across configurations.

The evaluation currently includes three projects satisfying these criteria and will be extended as the evaluation progresses. Project identifiers, selected revisions, experimental configurations, and generated outputs are available in the repository.

VI. Conclusion

This paper presented CRBF, a contextualised risk-based framework for supporting dependency update decisions in Maven projects. CRBF combines security, usage, ecosystem, and compatibility information to support dependency-update decisions while accounting for the effort required to apply each remediation. This addresses RQ1 by demonstrating how these heterogeneous sources of information can be combined within the proposed framework to produce contextualised update guidance.

RQ2 is currently being evaluated through a comparative experiment against OSV-Scanner with and without call graph analysis. The evaluation is still ongoing, and the current dataset is being extended before the collected outputs are analysed. At this stage, we cannot yet determine to what extent CRBF improves the effectiveness of dependency management.

Future work will focus on completing the experimental dataset and analysing the outputs produced by the three configurations. This analysis will address RQ2 by examining the effect of reachability and the additional context introduced by CRBF on dependency-update guidance.

Code and Data Availability

The source code of CRBF, together with the experimental configurations and generated outputs, is publicly available at [13].

References

[1] C. Nachuma, M. M. Hossan, A. K. Turzo, and M. F. Zibran, "Decoding Dependency Risks: A Quantitative Study of Vulnerabilities in the Maven Ecosystem," in *2025 IEEE/ACM 22nd International Conference on Mining Software Repositories (MSR)*, IEEE, Apr. 2025, pp. 270–274. doi: 10.1109/MSR66628.2025.00048.

[2] M. Keshani, S. Vos, and S. Proksch, "On the relation of method popularity to breaking changes in the Maven ecosystem," *Journal of Systems and Software*, vol. 203, p. 111738, Sep. 2023, doi: 10.1016/j.jss.2023.111738.

[3] S. Raemaekers, A. van Deursen, and J. Visser, "Semantic versioning and impact of breaking changes in the Maven

repository," *Journal of Systems and Software*, vol. 129, pp. 140–158, Jul. 2017, doi: 10.1016/j.jss.2016.04.008.

[4] D. Jaime, J. El Haddad, and P. Poizat, "Goblin: A Framework for Enriching and Querying the Maven Central Dependency Graph," in *2024 IEEE/ACM 21st International Conference on Mining Software Repositories (MSR)*, 2024, pp. 37–41.

[5] A. G. Márquez, Á. J. Varela-Vaca, M. T. Gómez López, J. A. Galindo, and D. Benavides, "Depex: A software for analysing and reasoning about vulnerabilities in software projects dependencies," *SoftwareX*, vol. 30, p. 102152, May 2025, doi: 10.1016/j.softx.2025.102152.

[6] D. Jaime, P. Poizat, J. El Haddad, and T. Degueule, "Balancing the Quality and Cost of Updating Dependencies," in *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*, in ASE '24. New York, NY, USA: Association for Computing Machinery, 2024, pp. 1834–1845. doi: 10.1145/3691620.3695595.

[7] K. Huang *et al.*, "Characterizing usages, updates and risks of third-party libraries in Java projects," *Empirical Softw. Engg.*, vol. 27, no. 4, Jul. 2022, doi: 10.1007/s10664-022-10131-8.

[8] L. Zhao *et al.*, "Software Composition Analysis for Vulnerability Detection: An Empirical Study on Java Projects," in *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, New York, NY, USA: ACM, Nov. 2023, pp. 960–972. doi: 10.1145/3611643.3616299.

[9] Y. Zhao, Y. Zhang, D. Chacko, and J. Cappos, "CovSBOM: Enhancing Software Bill of Materials with Integrated Code Coverage Analysis," *Proceedings - International Symposium on Software Reliability Engineering, ISSRE*, pp. 228–237, 2024, doi: 10.1109/ISSRE62328.2024.00031.

[10] Google, "OSV-Scanner Changelog." Accessed: Sep. 27, 2026. [Online]. Available: https://github.com/google/osv-scanner/blob/main/CHANGELOG.md

[11] J. Jacobs, S. Romanosky, O. Suciu, B. Edwards, and A. Sarabi, "Enhancing Vulnerability Prioritization: Data-Driven Exploit Predictions with Community-Driven Insights," in *2023 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, IEEE, Jul. 2023, pp. 194–206. doi: 10.1109/EuroSPW59978.2023.00027.

[12] A. Cockburn, "Hexagonal Architecture," 2005. [Online]. Available: https://alistair.cockburn.us/hexagonal-architecture/

[13] D. Almeida, "CRBF-Project repositories." Accessed: Jul. 25, 2026. [Online]. Available: https://github.com/orgs/CRBF-Project/repositories