Innovative Journal of Computing, Security and Digital Technologies
Volume 1, Issue 1, Jan – Apr 2026
Page no : 29-40

AI-Driven Vulnerability Prioritization Using Exploitability and Software Dependency Data

Shirazi. H, Maneriker.P , Fujita.H

University of Toronto, Toronto, Ontario, Canada.

Abstract

Software systems increasingly depend on open-source and third-party components. A vulnerability in one dependency may affect many applications, yet security teams often have limited time to investigate and remediate every finding. Prioritization based only on technical severity can overlook whether a vulnerability is likely to be exploited, whether the affected component is actually used, and whether it is reachable in the deployed application.

This paper proposes an AI-driven vulnerability prioritization framework that combines exploitability indicators with software dependency data. The framework integrates vulnerability records, exploit prediction scores, known exploitation evidence, dependency relationships, and deployment context. A dependency graph represents relationships among applications, packages, and vulnerable components. A machine-learning model estimates exploitation-related risk, while graph-based context identifies vulnerabilities that affect important or reachable components. The resulting priority score supports remediation decisions by considering both threat likelihood and organizational exposure.

The proposed approach is designed to improve the ordering of vulnerability remediation tasks rather than replace security analysts or established scoring systems. Evaluation should compare the framework with severity-only and exploitability-only baselines using a temporally separated dataset. Metrics should include precision and recall among the highest-ranked vulnerabilities, average precision, ranking quality, and remediation workload. The paper also discusses data quality, model explainability, dependency uncertainty, and operational limitations.

Keywords: Vulnerability prioritization, artificial intelligence, software dependencies, exploitability, dependency graph, risk-based remediation, cybersecurity.

Publication Date : 30.01.2026

1. INTRODUCTION

Modern software is assembled from libraries, frameworks, packages, containers, and other reusable components. This dependency model improves development speed and maintainability, but it also expands the security exposure of applications. A vulnerable package may be included directly by a project or indirectly through another dependency. As a result, an organization may receive vulnerability findings that are numerous, duplicated, or difficult to connect to actual deployment risk.

The challenge is not simply to identify vulnerabilities. It is to decide which findings should be addressed first. A vulnerability's technical severity describes important characteristics of the flaw, but does not by itself establish whether the vulnerable component is present in a particular environment, reachable by an attacker, or likely to be exploited. Exploit prediction and confirmed exploitation data add threat-related evidence. Dependency data adds information about where the vulnerable component is used and how it connects to applications and services.

The Exploit Prediction Scoring System (EPSS) estimates the probability that a publicly disclosed vulnerability will be exploited in the wild within the next 30 days. EPSS is not a complete organizational risk score: it does not determine whether a vulnerable component is present or exposed in a specific environment. That context must be supplied by the organization.

first.org

This paper proposes a framework that combines exploitability information with software dependency context. It treats vulnerability prioritization as a ranking problem: given a set of known findings, rank them so that the most actionable risks appear first. The framework is intended to support security teams operating under limited remediation capacity.

1.1 Research Problem

Organizations may have difficulty distinguishing between vulnerabilities that are severe in principle and vulnerabilities that represent urgent risk in their own software environment. A high-severity vulnerability in an unused or unreachable component may require a different response from a moderate-severity vulnerability in a widely deployed, externally accessible service.

The research problem is:

How can exploitability indicators and software dependency data be combined to rank vulnerabilities according to their likely security impact in a specific software environment?

1.2 Research Objectives

The objectives of this study are to:

1. Integrate vulnerability records, exploitability signals, and software dependency data.
2. Represent software dependencies as a graph that links applications to affected components.
3. Develop a model that estimates vulnerability priority using threat and dependency-context features.
4. Produce interpretable rankings that support remediation decisions.
5. Define an evaluation method that compares the proposed framework with conventional prioritization baselines.

1.3 Research Questions

- RQ1: Does combining exploitability information and dependency context improve vulnerability ranking compared with severity-only prioritization?

- RQ2: Which dependency-context features contribute most to the resulting priority ranking?
- RQ3: How does the framework perform when evaluated on later vulnerability disclosures rather than randomly selected records?
- RQ4: Can the model provide useful explanations for why a vulnerability is ranked as urgent?

2. BACKGROUND AND RELATED WORK

2.1 Vulnerability Severity and Exploitability

Vulnerability scoring systems provide structured descriptions of vulnerability characteristics. However, severity and exploitation likelihood represent different dimensions. A severity score may indicate the potential technical impact of a vulnerability, while exploitability data provides evidence or estimates about attacker activity.

EPSS provides a daily probability estimate for exploitation in the wild over the next 30 days. Its documentation emphasizes that the score does not account for an organization's particular assets, compensating controls, or deployment context. Therefore, EPSS is best treated as an input to a broader risk assessment rather than a final decision by itself.

first.org

The Known Exploited Vulnerabilities (KEV) catalog supplies evidence that vulnerabilities have been exploited. Such evidence is useful for identifying threats that have moved beyond theoretical possibility. However, a confirmed exploitation indicator does not alone describe the vulnerability's relevance to a particular organization. A component may be absent, unreachable, or protected by controls in one environment and exposed in another.

A recent integrated prioritization study illustrates the value of combining severity, exploit prediction, and confirmed exploitation evidence. Its results should be treated as evidence for the general value of multi-source prioritization, not as performance results for the framework proposed in this paper.

DOI.org

2.2 Software Dependency Data

Software dependency data describes how applications rely on packages and components. A dependency may be direct, meaning it is explicitly included by the application, or transitive, meaning it is introduced through another dependency.

A dependency inventory can help answer:

- Which applications contain the affected package?
- Is the package directly or transitively included?
- Which versions are installed?
- Is the vulnerable component used by a deployed service?
- Does the affected application support a business-critical function?
- Is the component exposed to untrusted input or external networks?

Dependency context is important because the same vulnerability can have different operational implications across deployments. A vulnerability in a component used by multiple critical applications may deserve more attention than one affecting a test-only component.

2.3 AI-Based Vulnerability Prioritization

Machine-learning approaches can combine multiple indicators and learn relationships that may not be captured by a single score. For example, exploitability signals, vulnerability characteristics, package popularity, and deployment context can be used as input features for a ranking or classification model.

However, predictive performance depends on the quality and timing of the data. If a model uses information that became available only after the prioritization date, it may appear more accurate than it would be in real operation. For this reason, a temporal evaluation design is more appropriate than a random split when the intended use is to rank newly disclosed vulnerabilities.

2.4 Research Gap

The proposed study focuses on the combination of two complementary data domains:

1. Exploitability data, which estimates or records the likelihood of exploitation.
2. Software dependency data, which connects a vulnerability to the software assets and applications that may be affected.

The intended contribution is a transparent ranking framework that uses both domains and produces actionable explanations. Its effectiveness must be established empirically; the framework is not assumed to outperform existing approaches before testing.

3. Proposed Framework

The framework contains six main stages:

1. Vulnerability and exploitability data collection
2. Software dependency inventory and graph construction
3. Feature extraction and data alignment
4. AI-based risk estimation
5. Context-aware priority ranking
6. Remediation recommendation and feedback

3.1 Data Sources

The framework can use the following categories of data:

Data category	Example fields	Purpose
Vulnerability records	CVE identifier, affected versions, severity, publication date	Identify and describe the vulnerability
Exploitability indicators	EPSS score, known exploitation status, public exploit evidence	Estimate or establish threat activity
Dependency	Package name, version, direct/transitive	Identify affected software

inventory	status	components
Dependency graph	Parent package, child package, application relationship	Trace the vulnerability's reach through software
Deployment context	Production status, external exposure, asset criticality	Estimate relevance to the organization
Remediation data	Fixed version, patch availability, mitigation	Support an actionable response

The data should be time-aligned. Each model input should reflect what was known at the prioritization date. Current values must not be used retrospectively when evaluating earlier decisions.

3.2 Dependency Graph Construction

The framework represents software relationships as a directed graph:

$$G=(V,E)$$

where V contains applications, packages, and component versions, and E represents dependency relationships. An edge from application a to package p indicates that the application depends on that package, directly or through another component.

A vulnerability is linked to the package versions it affects. The graph can then identify the applications that depend on those versions and estimate the breadth of potential impact.

A simple graph may include:

- Application nodes
- Package and version nodes
- Dependency edges
- Vulnerability-to-component links
- Optional deployment and asset nodes

The graph should preserve package version information. A package name alone is insufficient because only particular versions may be affected.

3.3 Feature Extraction

The framework extracts three groups of features.

Vulnerability and exploitability features

- Technical severity score
- Exploit prediction probability
- Known exploitation indicator
- Public exploit availability, where reliable and time-stamped
- Vulnerability age at the decision date

Dependency features

- Number of affected applications
- Number of dependency paths to affected applications
- Direct or transitive dependency status
- Number of affected deployed instances
- Dependency depth
- Package centrality within the dependency graph

Deployment-context features

- Production or non-production status
- External exposure
- Asset criticality
- Availability of a fixed version
- Existing mitigation or compensating control

These features should be normalized and checked for missing values. Missing data must not automatically be interpreted as evidence of low risk.

3.4 AI-Based Risk Estimation

The proposed model estimates a risk-related score for each vulnerability finding. Depending on the available labelled data, the model may be implemented as a classifier, a ranking model, or a calibrated probability estimator.

3.5 Explainability

For each ranked finding, the system should provide a concise explanation, such as:

- The vulnerability has a high exploitation estimate.
- The affected package is present in a production application.
- The package is used by several applications.
- The application is externally exposed.
- A fixed version is available.

Explanations should reflect actual input data and model behavior. They should not invent attack paths or claim confirmed exploitability when the available data only indicates a prediction.

4. SYSTEM WORKFLOW

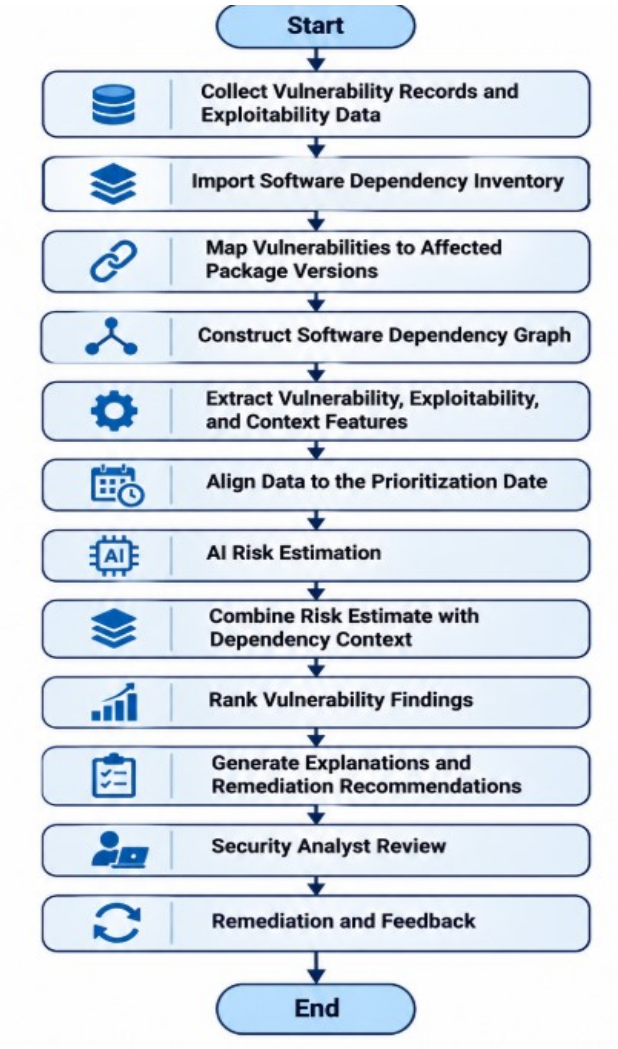


Figure 1. AI-Driven Vulnerability Prioritization Workflow

5. RESEARCH METHODOLOGY

5.1 Research Design

The study should use a quantitative experimental design. The proposed model is compared with baseline prioritization methods on a dataset of vulnerability findings. The unit of analysis should be explicitly defined. For example, it may be a unique CVE, a CVE-package-version pair, or a vulnerability finding in a particular application. Mixing these units can distort the results because one vulnerability may affect many packages or applications.

5.2 Dataset Construction

A reproducible dataset should include:

- Vulnerability identifiers and publication dates
- Severity and exploitability values available at the decision date
- Affected package names and version ranges

- Dependency relationships
- Deployment or asset context, where available
- A defined outcome label and observation window

The study should document data sources, collection dates, filtering rules, missing-value handling, and deduplication. If dependency or deployment context is unavailable, the paper should state that limitation rather than infer the missing information.

5.3 Training, Validation, and Test Sets

The data should be divided chronologically:

- Training set: earlier records used to fit the model.
- Validation set: later records used to select model settings and ranking thresholds.
- Test set: the latest held-out records used only for final evaluation.

This design better reflects the intended task of prioritizing future vulnerabilities. All features must be limited to information available at the relevant decision date. A random split may place related or near-duplicate findings across sets and can produce an overly optimistic estimate of performance.

5.4 Baseline Methods

The proposed framework should be compared with at least these baselines:

1. Severity-only ranking
2. Exploitability-only ranking
3. Severity combined with exploitability
4. Dependency-context ranking without the AI model
5. The complete proposed framework

An ablation study should remove one feature group at a time to determine whether exploitability, dependency graph features, and deployment context contribute measurable value.

5.5 Evaluation Metrics

The study should emphasize ranking quality and practical remediation usefulness.

Metric	Purpose
Precision@K	Fraction of the top-ranked \(K\) findings that meet the defined outcome
Recall@K	Fraction of all positive findings captured within the top \(K\)
Average precision	Evaluates ranking quality across thresholds
ROC-AUC	Measures discrimination across classification thresholds
PR-AUC	Useful when positive outcomes are rare
Remediation	Number of findings requiring review or action to capture a chosen share

workload	of positives
Calibration	Checks whether predicted probabilities correspond to observed frequencies

The positive outcome must be defined before selecting metrics. For example, predicting future exploitation is different from predicting whether a vulnerability is relevant to a specific deployed application.

6. PROPOSED MODEL AND ALGORITHM

The framework can use a tree-based model, such as gradient boosting, when the feature set is primarily tabular. A graph-based model may be considered if the dependency graph itself is used as a learned representation. The study should choose the model based on data availability, reproducibility, and performance against baselines—not on the assumption that a more complex model is necessarily better.

6.1 Training Objective

If the target is binary, the model can estimate:

$$\hat{p}_i = P(y_i = 1 \mid X_i)$$

where y_i=1 indicates that the defined outcome occurred within the study's observation window.

If the objective is to order findings rather than estimate probabilities, a learning-to-rank objective may be more directly aligned with remediation prioritization. The paper should state which approach is used and why.

6.2 Priority Categories

For operational use, a ranked list may be mapped into priority categories. The boundaries should be selected using validation results and organizational remediation capacity.

Category	Suggested interpretation
Priority 1	Strong exploitation evidence or high estimated risk, with relevant deployment exposure
Priority 2	Meaningful exploitation risk or broad dependency impact requiring planned remediation
Priority 3	Lower immediate risk, but should be tracked and handled through routine remediation
Review	Insufficient or conflicting information; requires analyst verification

These categories are proposed operational labels, not universal thresholds. A vulnerability with confirmed exploitation may still require immediate review even when the model score is low.

7. EXPECTED CONTRIBUTIONS

The proposed study is expected to make the following contributions:

1. Integrated data representation: a common structure for vulnerability, exploitability, and dependency information.
2. Dependency-aware ranking: a method for considering which applications and components are affected.
3. Explainable prioritization: ranked findings accompanied by evidence-based explanations.
4. Evaluation protocol: a temporal test design and baseline comparisons for measuring prioritization quality.
5. Operational guidance: a workflow connecting ranking results to analyst review and remediation.

These are intended contributions. Their value must be confirmed by the completed evaluation.

8. DISCUSSION

8.1 Why Dependency Context Matters

A vulnerability's importance depends partly on where the affected component is used. A component may appear in a software inventory but not be deployed. It may be deployed but unreachable from an attacker-controlled input. It may also be shared across multiple production applications. Dependency relationships help distinguish these cases.

However, dependency count should not be treated as a direct measure of risk. A component used by many applications may be low-impact if those applications are isolated or not exposed. Conversely, a component used by one externally accessible critical service may deserve urgent attention. Dependency breadth should therefore be interpreted alongside deployment and exposure context.

8.2 Why Exploitability and Severity Should Remain Distinct

Exploitability and severity answer different questions. Exploitability estimates whether attackers may exploit a vulnerability; severity characterizes technical consequences under specified conditions. A robust prioritization process should preserve both dimensions rather than assume one replaces the other.

EPSS documentation explicitly cautions that its score does not account for organization-specific context or impact. This supports the need to combine exploitation estimates with local asset and dependency information.

first.org

8.3 Explainability and Human Review

A model-generated ranking should assist, not replace, security analysts. The system should show the factors that influenced each finding's position and allow analysts to inspect the underlying records. This is especially important when inputs are incomplete, conflicting, or stale.

A review process should also allow analysts to record why a finding was escalated or deprioritized. Such feedback can improve future evaluations, but it must be handled carefully:

analyst decisions may reflect resource constraints or policy requirements rather than the underlying probability of exploitation.

8.4 Operational Integration

The framework can be integrated into a vulnerability management pipeline that receives scanner results, matches them to dependency inventory, enriches findings with exploitability data, and produces a ranked remediation queue. Integration should preserve data provenance and timestamp each input.

The model should not automatically close findings merely because they receive a low priority score. Low-ranked findings should remain visible, and policy-based exceptions should be documented.

9. LIMITATIONS AND ETHICAL CONSIDERATIONS

9.1 Data Completeness

Vulnerability records, dependency inventories, and exploitation indicators may be incomplete or delayed. Missing dependency information can cause the framework to underestimate exposure. The system should distinguish “not present” from “unknown.”

9.2 Label Quality

Confirmed exploitation is not always fully observable. A dataset based on reported exploitation may underrepresent attacks that were not detected or publicly disclosed. The study should describe how labels are created and avoid treating absence of evidence as definitive evidence of safety.

9.3 Temporal Drift

Exploit patterns, package ecosystems, and attacker behavior change over time. A model trained on older vulnerabilities may not generalize to new classes of software or emerging attack methods. Periodic evaluation and retraining may be necessary.

9.4 Bias Toward Well-Monitored Software

Popular software may have more public reporting and better dependency metadata than less common or internally developed software. The model may therefore rank well-documented components more confidently than poorly documented ones. The evaluation should examine performance across software categories and data-availability levels.

9.5 Misuse and Overreliance

A priority score is a decision-support signal, not a guarantee that a vulnerability will or will not be exploited. The system should avoid presenting uncertain predictions as certainties and should preserve analyst review for high-impact decisions.

10. CONCLUSION

This paper proposed an AI-driven vulnerability prioritization framework that combines exploitability indicators with software dependency data. The framework maps vulnerabilities to affected package versions, constructs a dependency graph, extracts threat and deployment-context features, estimates risk, and ranks findings for remediation review.

The central premise is that vulnerability priority depends on more than technical severity alone. Exploitability indicators provide threat-related information, while dependency and deployment data help establish whether a vulnerability is relevant to the organization's software environment. A temporal evaluation using clear outcome labels, meaningful baselines, and ranking metrics is required to determine whether the proposed combination improves prioritization.

The framework is intended to support transparent, context-aware vulnerability management. It should be used alongside established security processes, reliable data inventories, and human review rather than as an autonomous replacement for remediation decisions.

REFERENCES

1. Jacobs, J., Romanosky, S., Edwards, B., Adjerid, I., & Roytman, M. (2021). Exploit Prediction Scoring System (EPSS). Digital Threats: Research and Practice, 2(3), Article 20, 1–17. https://doi.org/10.1145/3436242
2. Jacobs, J., Romanosky, S., Adjerid, I., & Baker, W. (2020). Improving vulnerability remediation through better exploit prediction. Journal of Cybersecurity, 6(1), tyaa015. https://doi.org/10.1093/cybsec/tyaa015
3. Bozorgi, M., Saul, L. K., Savage, S., & Voelker, G. M. (2010). Beyond heuristics: Learning to classify vulnerabilities and predict exploits. In Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (pp. 105–114). https://doi.org/10.1145/1835804.1835820
4. Allodi, L., & Massacci, F. (2014). Comparing vulnerability severity and exploits using case-control studies. ACM Transactions on Information and System Security, 17(1), Article 1. https://doi.org/10.1145/2630069
5. FIRST. (2023). Common Vulnerability Scoring System version 4.0: Specification document. https://www.first.org/cvss/v4.0/specification-document
6. FIRST. (2023). Common Vulnerability Scoring System version 4.0: User guide. https://www.first.org/cvss/v4.0/user-guide
7. Kausar, F., Batiste, L., Muallem, A., & Hussain, S. (2026). Integrated risk scoring and exploit prediction for cyber-physical power system vulnerabilities. Energy Informatics, 9, Article 32. https://doi.org/10.1186/s42162-026-00640-x
8. Parente, F. R., Rodrigues, E. B., & Mattos, C. L. C. (2025). FRAPE: A framework for risk assessment, prioritization and explainability of vulnerabilities in cybersecurity. Journal of Information Security and Applications, 89, Article 103971. https://doi.org/10.1016/j.jisa.2025.103971
9. Márquez, A. G., Varela-Vaca, Á. J., & Gómez López, M. T. (2024). Vulnerability impact analysis in software project dependencies based on satisfiability modulo theories (SMT). Computers & Security, 139, Article 103669. https://doi.org/10.1016/j.cose.2023.103669
10. Enhancing vulnerability prioritization: Data-driven exploit predictions with community-driven insights. (2023). In 2023 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW). https://doi.org/10.1109/EuroSPW59978.2023.00027