ISSN: 2096-3246
Volume 58, Issue 04, 2026

CENTRALIZED APPLICATION INTELLIGENCE SYSTEM FOR DEPENDENCY ANALYSIS, VULNERABILITY DETECTION, AND LIFECYCLE MANAGEMENT

Anil Kumar Chitiprolu
Independent Researcher, USA

Abstract
Modern organizations have hundreds of applications, each based on different technology stacks, and each of those potentially depends on third-party and open-source libraries. The situation has become more complicated with libraries reaching end of life and vulnerabilities becoming unpatched. The Centralized Application Intelligence System (CAIS) brings all application metadata into one place to create an intelligence-based application cybersecurity platform. This system also automates dependency discovery, vulnerability detection and lifecycle tracking, leveraging various open-source tools, including the OWASP Dependency-Check Project and Snyk. The paper develops three application security quantitative measurements and analyzes them on ten representative enterprise applications: the Vulnerability Risk Score (VRS), the Dependency Freshness Index (DFI), and the Remediation Impact Score (RIS). In the sample, six in ten applications had a DFI greater than 50 percent, indicating an important amount of technical debt in their dependency portfolios. The VRS analysis also identified mission-critical systems with composite risk scores at the maximum level, which need immediate attention. The RIS has two of the ten proposed upgrades as major coordination efforts due to their potential blast radius. CAIS is the enterprise-scale software governance, following the practices of the DevSecOps era.
Keywords: Application Inventory, Dependency Management, Vulnerability Analysis, Software Lifecycle, Risk Assessment, DevSecOps.

1. Introduction
Complexity defines the modern enterprise software landscape. Organizations routinely operate hundreds of applications, each built using different programming languages, frameworks, and third-party components. A single application may depend on dozens of open-source libraries, which in turn depend on hundreds more. However, such a layered dependency structure makes it impossible for mere manual processes and periodic point-in-time audits to keep up [3].
There is a high risk of unintentional dependency. Open-source vulnerabilities have been known to be publicly disclosed in national vulnerability databases, such as the National Vulnerability Database (NVD) [2], and to be actively exploited when present in popular libraries. In practice, many organizations continue to use packages with known critical vulnerabilities even after the patch is available [15]. In-the-wild exploitation of publicly disclosed vulnerabilities occurs faster than many organizations can patch them [14].
Dependency management is an aspect of software maintainability and compliance. Libraries that have reached end-of-life (EOL) no longer receive maintenance or security patches. Organizations that do not monitor EOL timelines are on a path towards increasing technical debt and compliance violations. The software supply chain is one of the most favorable attack surfaces, as demonstrated by several famous attacks [6]. To target this problem, the article proposes CAIS, the centralized application intelligence system, a scalable and practical way to monitor EOL timelines.

2. Problem Statement
Enterprise organizations face a set of interconnected dependency management challenges that existing tools do not fully address. The first challenge is a lack of centralized visibility. Most Software Composition Analysis (SCA) tools operate at the project level, meaning they require teams to scan their projects separately. In practice, this approach scatters the results of different teams across different repositories and dashboards, making it difficult to get a picture of software composition risk at the enterprise level. [9]
The second challenge is that dependency versioning is reactive, either when alerted to vulnerabilities or when an incident occurs, rather than as a part of the governance process [15]. Furthermore, research has shown that different SCA tools give different reported vulnerabilities (False positives, False negatives, etc.) for the same dependency, leading to incomplete coverage and alert fatigue [9].

Aside from basic tooling to track the lifecycle of a dependency, nothing is known about its status, so organizations often do not know when support is almost dropped [17]. Furthermore, the impact of an upgrade is not usually assessed until integration tests are run in production [8].

The end effect of these gaps is visible in measurable indicators: security vulnerabilities and compliance issues arising from unpatched disclosed vulnerabilities in production, as well as a continuing trend of concerns that upgrading one application will cause a cascade of upgrades to dependent applications. Thus, the total risk of an unquantified application portfolio may be considerable. This strategy is designed to address these gaps systematically with a single, integrated, automated, and continuously updated intelligence platform.

3. Literature Review

Software Composition Analysis has emerged as a foundational practice within DevSecOps, driven by the growing proportion of open-source code in enterprise applications. Tools such as OWASP Dependency-Check [1] and Snyk scan project manifests against the NVD [2] to identify known vulnerabilities. These tools have demonstrated measurable value in identifying high-severity Common Vulnerabilities and Exposures (CVEs) before deployment. However, these are project-level tools and do not provide enterprise-wide aggregation or cross-application comparative analysis [9].

Decan, Mens and Grosjean investigated the dependency graphs of seven packaging ecosystems and found that the dependency graphs became more complex over time, increasing the risk of transitive vulnerabilities being passed from libraries downstream to consumers [3]. This study illustrated that when a package declares a dependency, it does not simply add a library but instead adds a complex network of indirect packages. Kikas et al. have confirmed that package dependency networks are dense and rapidly changing, making point-in-time scans unsuitable for continuous risk management [13]. Chinthanet et al. reported that vulnerability disclosures were slow to be fixed within npm ecosystems, with latency ranging from weeks to months, depending on the severity of vulnerabilities and the popularity of packages [14]. Security measures and trust processes vary substantially across projects, with smaller projects facing limited contributor capacity for dependency reviews and vulnerability response. This heterogeneity in maintainer capability means that automated enterprise-level tooling cannot rely on upstream projects to self-remediate consistently, reinforcing the need for continuous, organization-side monitoring rather than point-in-time dependency scans [10].

The software bill of materials (SBOM) has recently emerged as a foundational artifact in supply chain security due to the 2021 U.S. Executive Order on Improving the Nation's Cybersecurity. Mirakhorli et al. conducted a landscape study of SBOM tools and found a major gap in automated SBOM generation, validation, and lifecycle integration [7]. Ponta, Plate and Sabetta developed a tool to identify, prioritize, and reduce vulnerabilities in open-source dependencies. They also note that not all declared vulnerabilities will be exploited in a given project context [8]. Alfadel et al. studied Python packages for vulnerabilities, finding the average lifespan between a package's release and vulnerability disclosure is a few years, underlining the importance of continuing vulnerability monitoring [11]. CAIS builds on this research by providing the enterprise-wide, continuous, lifecycle-aware, and context-weighted intelligence missing from existing tools and three unique quantitative metrics that have not been covered in the literature.

4. Proposed System

4.1 System Overview

CAIS centralizes intelligence for each application running in its environment. It pulls data for each application's dependencies from the company's source code repositories, CI/CD pipelines, and package manifests. This prevents the need for each of the company's teams to run their own scans. The captured data is normalized, and all the vulnerability and lifecycle information is consolidated in a shared central repository. The platform is based on a modular architecture that allows its functions to be running and scaled independently and is used in enterprise environments with dozens up to thousands of applications in production.

In the proactive component, CAIS compares the versions of declared dependencies against a list of known end-of-life dates for libraries and warns if any library is nearing end of life. In the reactive component, it warns at the time of project declaration if any of the declared dependencies match the

NVD [2]. It integrates with tools like OWASP Dependency-Check [1] and Snyk to achieve real-time alerts for certain vulnerabilities. The presence of both components provides a more complete security posture. The platform also provides a framework for remediation planning with the introduction of a Remediation Impact Score (RIS), which estimates the impact of an upgrade on the service's risk.

CAIS has one data model that unifies records across the npm, Maven, pip, and Gradle ecosystems. The application record stores its underlying stack technology, owning team, and hosting environment, as well as its fully resolved dependency tree. This normalization layer allows statistics to be gathered across the applications, which would not be possible on a per-project basis with a piecemeal tool [9], and can run continuously, scanning applications on a user-defined schedule or upon repository events like pull requests and merges.

4.2 Key Features

The application inventory is a searchable list of all enterprise applications, including their technology stacks, teams, and deployment environments. Dependency mapping is a complete graph of all direct and transitive dependencies of each application resolved from package manifest files. Vulnerability detection queries the NVD [2] and incorporates scanning tools [1] that match a project's dependencies with known CVEs but assigns context-weighted VRS scores instead of CVSS scores. Lifecycle tracking queries EOL dates for all matched versions of libraries and alerts teams when a dependency is near or past its EOL date. The recommendation engine will suggest upgrade options and other high-priority libraries as per the VRS. The impact analyzer computes the RIS of the proposed upgrade and identifies services that breaking changes would affect before the upgrade is deployed.

Every CAIS feature addresses one or more of the gaps identified in the problem statement. The centralized inventory resolves the visibility fragmentation that arises from per-team scanning. The VRS metric provides context-aware risk ranking that raw CVSS scores cannot deliver, since the same vulnerability may carry very different organizational risk depending on whether the affected application handles payment data or internal logging. The DFI metric surfaces technical debt in the dependency landscape in a format that is meaningful to both security teams and engineering managers. Additionally, the RIS metric directly addresses the barrier to remediation that Kula et al. found was the primary reason teams deferred patching [15].

CAIS integrates into CI/CD pipelines as another build time gate for dependency governance. If a VRS and DFI risk score threshold is configured and a new dependency exceeds that threshold, the build will fail. This policy enforcement ability moves dependency governance to the left in the software development process, similar to how the DevSecOps approach favors moving security controls as early as possible in the software development life cycle rather than auditing post-deployment [5].

4.3 Functional Modules

The Application Discovery Module scans all connected repositories and CI/CD pipelines, identifying all applications in scope, collecting application metadata (language, framework, build tool, and team ownership) and producing a normalized application entity in the CAIS datastore. The module is configured to run according to a schedule and searches for changes in the repositories to trigger re-scans. Dependency Analyzer extracts the package manifests from applications and builds their dependency tree to the nth level of nesting [3]. It thus builds a dependency graph, showing each version of each library and its consumers within the enterprise.

The Vulnerability Scanner compares the resolved dependency graph against CVE databases [2] and built-in tools [1]. It detects whether each library version is affected and infers the VRS for each application. The Lifecycle Tracker overlays support timeline information (from vendor EOL registries) to determine when libraries are or are likely to be EOL. The Recommendation Engine generates upgrade recommendations from the improved data, ranking each upgrade recommendation with VRS and filtering it with DFI. The Impact Analyzer takes a chosen upgrade recommendation and retrieves the RIS by searching the changelog for breaking changes and matching them with the downstream services that are affected in the enterprise dependency graph.

The Central Dashboard receives all outputs. Developers, security analysts, and governance teams have role-based dashboards showing risk at the application level, most relevant to their workflow, including upgrade recommendations [16]. Security analysts can access vulnerability reports aggregated across applications, and governance teams can see risk metrics at the portfolio level, end-of-life policy compliance status, and SLA compliance for vulnerabilities remediated. Alerts can be

ISSN: 2096-3246
Volume 58, Issue 04, 2026

sent via email, Slack, or webhook, and thresholds can be chosen for determining whether a critical vulnerability is escalated [4, 7].

5. System Architecture

The CAIS architecture consists of a six-layered pipeline of functional tiers. At the input tier, organizational applications across repositories and cloud environments feed metadata into the Application Scanner, which connects to source control systems via authenticated REST APIs. The scanner produces structured application records that flow into two parallel processing subsystems: the Dependency Analyzer and the Vulnerability Scanner.

The Dependency Analyzer computes the dependency graph and persists it in the document store. The Vulnerability Scanner uses the CVE database [2] and other external SCA tools [1] to add security information into the dependency nodes. The output from the two subsystems is input for the Lifecycle Tracker, which adds the EOL and vendor support information. These improved records are then fed into the Recommendation Engine, which generates a prioritized list of upgrade recommendations. Recommendations are then sent through the Impact Analyzer, which calculates the RIS scores, and are displayed on the Central Dashboard. The back end is a Microsoft .NET or Node.js application that stores the dependency graph in MongoDB and the application and user data in PostgreSQL. The complete solution is packaged as a set of containerized microservices and deployed to cloud providers including AWS, Azure or GCP, where it can be horizontally scaled for enterprise use cases [4].

Following the separation of concerns philosophy, each module publishes its output to a shared event bus. Consumers of the bus can receive improved data. For example, the Vulnerability Scanner is a high-throughput component that may scan several thousand dependency records in a single scan cycle. High-throughput components can be scaled independently of low-throughput components such as the Recommendation Engine. This modular approach allows organizations to adopt CAIS in stages, activating CAIS modules as their security improves.

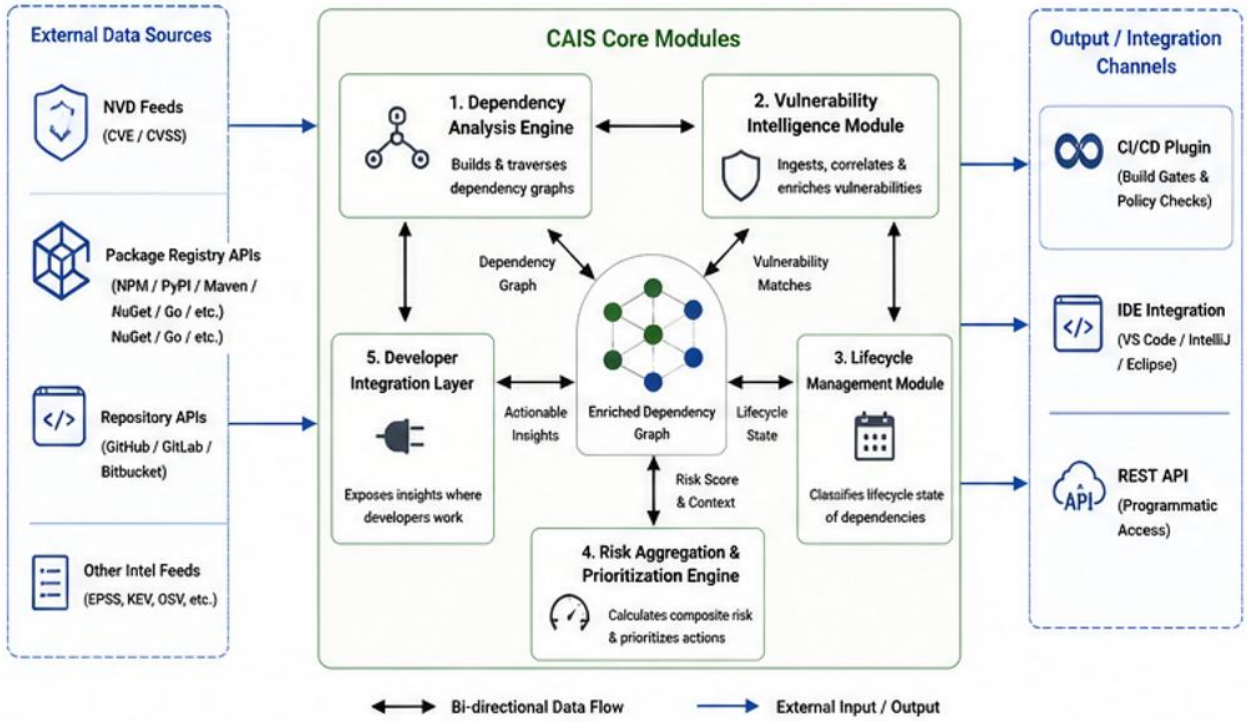

Figure 1: CAIS System Architecture: Pipeline from Application Sources through Analysis Modules to Central Dashboard

6. Methodology

6.1 Algorithmic Workflow

The CAIS scan cycle follows a structured algorithm applied iteratively to each application in the enterprise inventory. For each application, the system connects to the source repository, extracts the

package manifest, and resolves the full dependency tree, including all transitive dependencies. For each resolved dependency, the system queries the NVD and integrated SCA tools to retrieve known CVEs, computes the VRS using the formula defined below, queries vendor timelines for EOL data, and computes the DFI. Once per-dependency metrics are computed, the system aggregates them into an application-level risk summary. For any proposed upgrade, the Impact Analyzer computes the RIS. All results are stored in the CAIS datastore and surfaced on the Central Dashboard. Alerts are triggered if an application exceeds the configured threshold for VRS, or if an application has a dependency that is within 90 days of EOL.

The algorithm runs for less than five minutes for a 1000 direct and transitive dependencies portfolio using concurrent requests to the NVD API and caching where possible. The results of the scan are versioned to enable change tracking over time, allowing governance teams and organizations to track remediation rates for vulnerabilities and technical debt growth or reduction across the portfolio of dependencies. This ability to interpret timestamps distinguishes it from point-in-time SCA tools, which do not retain scans for analysis of trends.

6.2 Quantitative Metrics

Three metrics are defined to quantitatively assess the risk of a dependency and the feasibility of an upgrade. Each metric is a single number that can be compared between applications, tracked over time, and used to make prioritization decisions.

Vulnerability Risk Score (VRS): The VRS extends the CVSS base score provided by the NVD [2] by adding an exploitability factor and a business impact weight. The CVSS base score is a widely used metric for measuring vulnerability severity. It is defined as a number between 0 and 10. Exploitability indicates low impact if the vulnerability is exploitable. It ranges from 0.0 (no known exploit exists) to 1.0 (exploits are useful to attackers and are known to security researchers). Business impact weighting varies by business criticality, at 0.5 for low-impact internal applications, 1.0 for standard business applications, and 1.5 for mission-critical applications such as the payment gateway and authentication services. This approach prevents a high-severity vulnerability in a low business criticality logging utility from being treated with more urgency than a high-severity vulnerability in a payment gateway.

$$VRS = CVSS\_Base \times Exploitability\_Factor \times Business\_Impact\_Weight$$

Dependency Freshness Index (DFI): DFI is a percentage measuring how old a deployed version of a dependency was as compared to the upgrade distance. A DFI of 0% indicates the dependency is fully current. A DFI approaching 100% indicates the deployed version is at the earliest end of the release history relative to the latest available. For practical application, DFI values above 50% are flagged as requiring review, as they indicate that the dependency is significantly behind the current release, increasing the probability that known vulnerabilities exist in the gap between current and deployed versions [14].

$$DFI = [(Latest\_Version - Current\_Version) / Latest\_Version] \times 100$$

Remediation Impact Score (RIS): the sum of breaking changes listed in the upgrade changelog for affected packages, as well as the size of the enterprise's downstream dependency on affected packages. It is divided by the total number of services in the enterprise. The score is more or less the blast radius of an upgrade, since a higher RIS means that an upgrade requires more coordination and testing before installation. If the risk score is less than 10% (low-risk), the upgrade is performed directly. If between 10% and 50% (medium-risk), the upgrade is performed gradually, and if more than 50% (high-risk), the upgrade needs a detailed review and a planned phased rollout.

$$RIS = (Breaking\_Changes \times Affected\_Services) / Total\_Services \times 100$$

7. Implementation

CAIS is a cloud-native, multi-tier application built using microservices architecture, with a backend using either Microsoft .NET or Node.js; both were selected for their ecosystem support for API integration capabilities and package management tooling. PostgreSQL is used to store application metadata, while MongoDB is used to model the dependency graph in the form of variable depth package trees. The system connects to repository providers including GitHub, GitLab, and Azure DevOps via authenticated REST APIs and processes manifest files across all major ecosystems, including package.json for npm, pom.xml for Maven, requirements.txt for pip, and build.gradle for Gradle [1, 2].

Vulnerability data is ingested from the NVD using its public REST API, which provides structured CVE records including CVSS scores, CWE classifications, and CPE matching data [2]. This data is supplemented by integration with OWASP Dependency-Check [1] and Snyk, which provide tool-level scan results that catch vulnerabilities not yet reflected in the NVD feed. Lifecycle data is sourced from vendor EOL registries with scheduled polling to maintain currency. In all cases, the results from all three sources are deduplicated and merged at the level of the dependency record. Each vulnerability will then be scored for confidence based on the number of independent sources of evidence for its existence, addressing the inconsistency in vulnerability coverage across various sources [9].

It can run on AWS, Azure, or GCP via Docker containers on Kubernetes and horizontally scale to support large enterprise inventories. It supports CI/CD pipelines, with pre-built plugins for both Jenkins and GitHub Actions that automatically trigger CAIS scans on each build and on each pull request (PR). The Central Dashboard is an application-level dashboard showing a summary of risk, drill-downs on dependencies, EOL calendars, and remediation guidance. Role-based access ensures that users only see information about the applications they are responsible for, while security and governance features see all applications in the enterprise. Alerts can be in the form of email, Slack or webhooks [4, 7, 5].

8. Results and Discussion

To validate the metrics, CAIS is applied against a representative sample of enterprise applications of varying technology and criticality ratings: a payment gateway, a customer portal, a human resources system, an internal reporting system, a developer tooling suite, an application programming interface (API) gateway, an authentication service, a data pipeline, an email-delivery service, and a logging system. All CVSS base scores are drawn from the NVD [2] and reflect the published severity of the highest-severity known vulnerability in the dependency tree at the time of analysis. Exploitability factors are derived from public exploit availability data [1, 9]. Business impact weights are assigned per the criticality tier classification defined in Section 6.2. Total enterprise service count is set at 50 for RIS calculations.

The results across all three metrics are presented in Tables 2, 3, and 4. Table 1 provides a comparative feature matrix positioning CAIS against existing leading SCA tools, establishing the baseline against which the proposed system's contributions are assessed [1, 9, 18].

Table 1: Feature Comparison of Existing SCA Tools vs. CAIS

Feature	OWASP Dep-Check	Snyk	Black Duck	CAIS (Proposed)
Enterprise-wide Visibility	No	Partial	Yes	Yes
Vulnerability Detection	Yes	Yes	Yes	Yes
Lifecycle / EOL Tracking	No	Partial	Yes	Yes
Impact Analysis (pre-patch)	No	No	No	Yes
Context-aware Risk Scoring	No	No	No	Yes (VRS)
Freshness Index (DFI)	No	No	No	Yes (DFI)
CI/CD Integration	Yes	Yes	Yes	Yes
Open-Source/Free Tier	Yes	Yes	No	Open/Enterprise

Table 1 shows that CAIS is the only platform in the comparison that provides impact analysis before patch application, context-aware VRS scoring, and the DFI metric. While enterprise tools such as Black Duck offers lifecycle tracking; none of the reviewed tools combine all six capabilities required for a complete dependency governance solution. This positions CAIS as a complementary layer to existing tooling rather than a simple replacement.

Table 2: VRS Calculation Results Across Ten Enterprise Applications

Application	CVSS Base	Exploit Factor	Impact Weight	Raw Score	VRS (Final)
Payment Gateway	8.5	0.90	1.5 (Critical)	11.48	10.00 (Capped)
API Gateway	9.1	0.80	1.5 (Critical)	10.92	10.00 (Capped)
Customer Portal	7.2	0.70	1.3 (High)	6.55	6.55

ISSN: 2096-3246
Volume 58, Issue 04, 2026

Application	CVSS Base	Exploit Factor	Impact Weight	Raw Score	VRS (Final)
Auth Service	7.8	0.60	1.4 (High)	6.55	6.55
HR System	5.5	0.50	1.0 (Medium)	2.75	2.75
Data Pipeline	5.0	0.40	1.0 (Medium)	2.00	2.00
Dev Tools	6.8	0.40	0.6 (Low)	1.63	1.63
Reporting Tool	4.0	0.30	0.7 (Low)	0.84	0.84
Email Service	3.5	0.20	0.8 (Low)	0.56	0.56
Logging System	2.5	0.10	0.5 (Minimal)	0.13	0.13

VRS Analysis: Table 2 shows the full VRS analysis for all ten applications. The payment gateway and API gateway applications both have the highest values, capped at 10.0. Combined with the CVSS parameters of the payment gateway showing the CVSS base score of 8.5, exploitability of 0.90 from public exploits, and weight of 1.5 for the business impact because this system processes transactions, the calculated raw VRS is 11.48. This is capped at the maximum of 10.0 to keep the VRS on a bounded scale, and the API gateway's CVSS score of 9.1 with an exploit factor of 0.80 leads to an overall score of 10.92, also capped at 10.0. The customer portal and authentication service both record VRS values of 6.55, reflecting serious but not maximally exploited vulnerabilities in high-impact systems. The lower half of the portfolio registers VRS values below 3.0, confirming that the metric successfully differentiates risk levels across heterogeneous applications in a way that raw CVSS scores cannot achieve without the business context layer [8, 18].

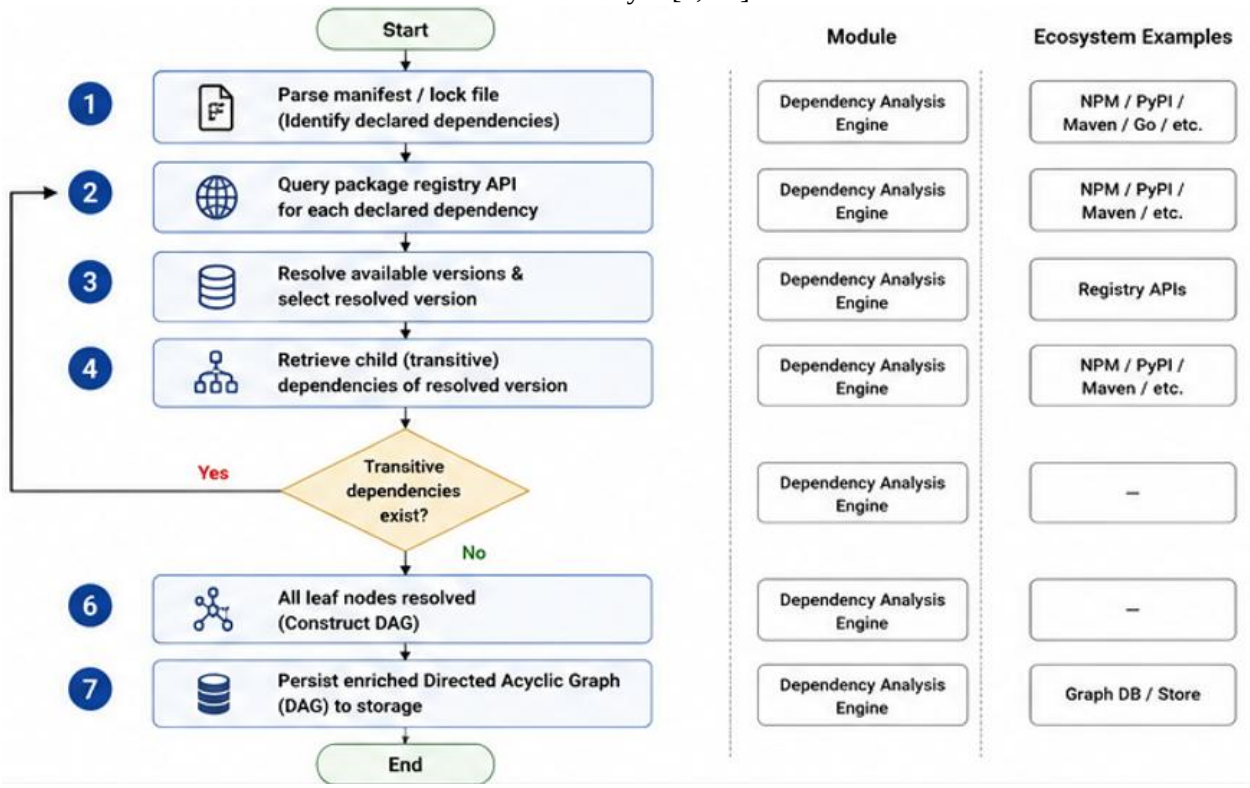

Figure 2: VRS Distribution Across Ten Applications (Descending Order)

Table 3: Dependency Freshness Index and EOL Status by Application

Application	Current Ver.	Latest Ver.	DFI (%)	EOL Status	Recommended Action
API Gateway	1.5	4.0	62.5%	EOL	Immediate Upgrade
Customer Portal	1.0	2.5	60.0%	Approaching EOL	Priority Upgrade
Reporting Tool	1.2	3.0	60.0%	EOL	Immediate Upgrade
Data Pipeline	2.0	5.0	60.0%	Approaching EOL	Priority Upgrade
Payment Gateway	2.1	3.0	30.0%	Supported	Planned Upgrade
Auth Service	3.0	3.5	14.3%	Supported	Monitor

ISSN: 2096-3246
Volume 58, Issue 04, 2026

Application	Current Ver.	Latest Ver.	DFI (%)	EOL Status	Recommended Action
Logging System	3.5	4.0	12.5%	Supported	Monitor
HR System	4.5	5.0	10.0%	Supported	Monitor
Email Service	1.8	2.0	10.0%	Supported	Monitor
Dev Tools	2.8	3.0	6.7%	Supported	No Action

Table 3 shows that 6 of the 10 sampled applications have a DFI equal or above 50%, meaning more than half of the code in the portfolio of the enterprise is based on old dependency versions. The API gateway application has the highest DFI at 62.5%, equating to 2.5 major versions behind the latest. Both the reporting tool and the customer portal have a DFI of 60%. The reporting tool is end-of-life. The customer portal is approaching end-of-life. The data pipeline has 60% DFI and has an approaching End of Life (EOL) status, which is compounded by the fact that the version of the dependency is out of date. The findings corroborate the results of Kula et al. that teams tend to postpone dependency upgrades even when security advisories exist, and the reason is that they indicate not being aware of breaking changes [15]. CAIS overcomes this barrier by providing RIS scores alongside the DFI data, which empowers teams to make well-informed decisions and not delay their dependency updates.

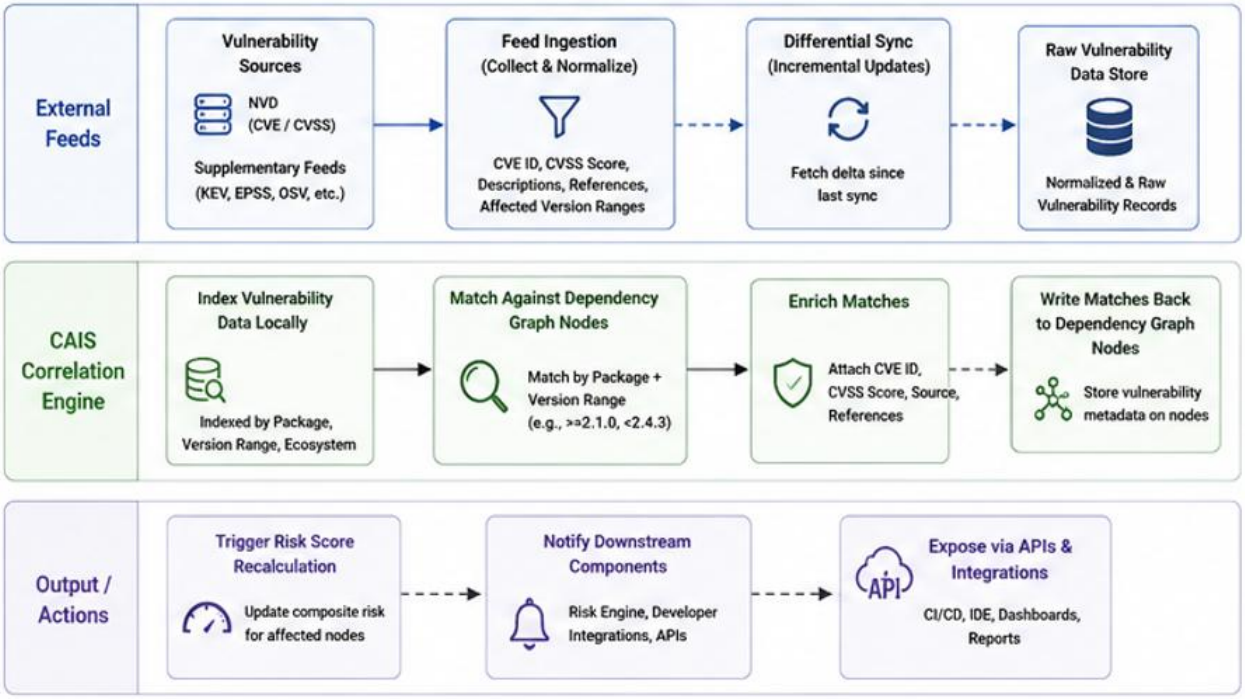

Figure 3: Dependency Freshness Index Distribution Across Ten Applications

Table 4: Remediation Impact Score for Proposed Upgrades (Total_Services = 50)

Application	Breaking Changes	Affected Services	RIS (%)	Risk Level	Recommendation
API Gateway	5	8	80.0%	High	Formal Review + Phased Migration
Reporting Tool	4	6	48.0%	Moderate	Staged Rollout Required
Payment Gateway	3	5	30.0%	Moderate	Staged Rollout Required
Customer Portal	2	4	16.0%	Moderate	Staged Rollout Required
Auth Service	2	3	12.0%	Moderate	Staged Rollout Required
Data Pipeline	1	4	8.0%	Low	Direct Upgrade
HR System	1	3	6.0%	Low	Direct Upgrade
Dev Tools	1	2	4.0%	Low	Direct Upgrade
Logging System	1	2	4.0%	Low	Direct Upgrade

ISSN: 2096-3246
Volume 58, Issue 04, 2026

Application	Breaking Changes	Affected Services	RIS (%)	Risk Level	Recommendation
Email Service	0	1	0.0%	Minimal	Direct Upgrade

Table 4 shows the RIS calculations based on the proposed upgrade to each of the 10 applications. The API gateway had the highest RIS (80%), since its upgrade changelog involved 5 breaking changes across 8 downstream services. The upgrade's score also shows a clear need for formal consideration of risk, earning a high-risk vulnerability score with an architectural review and staged rollout in place of the normal release cycle. The reporting tool earns a score of 48% (moderate-risk). The payment gateway records 30% RIS, also moderate, despite carrying the highest VRS. This combination reveals an important pattern: the application with the greatest security urgency also carries a meaningful upgrade coordination requirement. CAIS surfaces both dimensions simultaneously, enabling teams to plan for the upgrade complexity rather than discovering it during rollout [8, 12]. In contrast, the email service records a RIS of 0.0% with no breaking changes and minimal service impact, making it a candidate for immediate direct upgrade. The combined VRS, DFI, and RIS analysis provides a prioritized, multi-dimensional remediation roadmap that no single existing tool delivers [9, 19, 20].

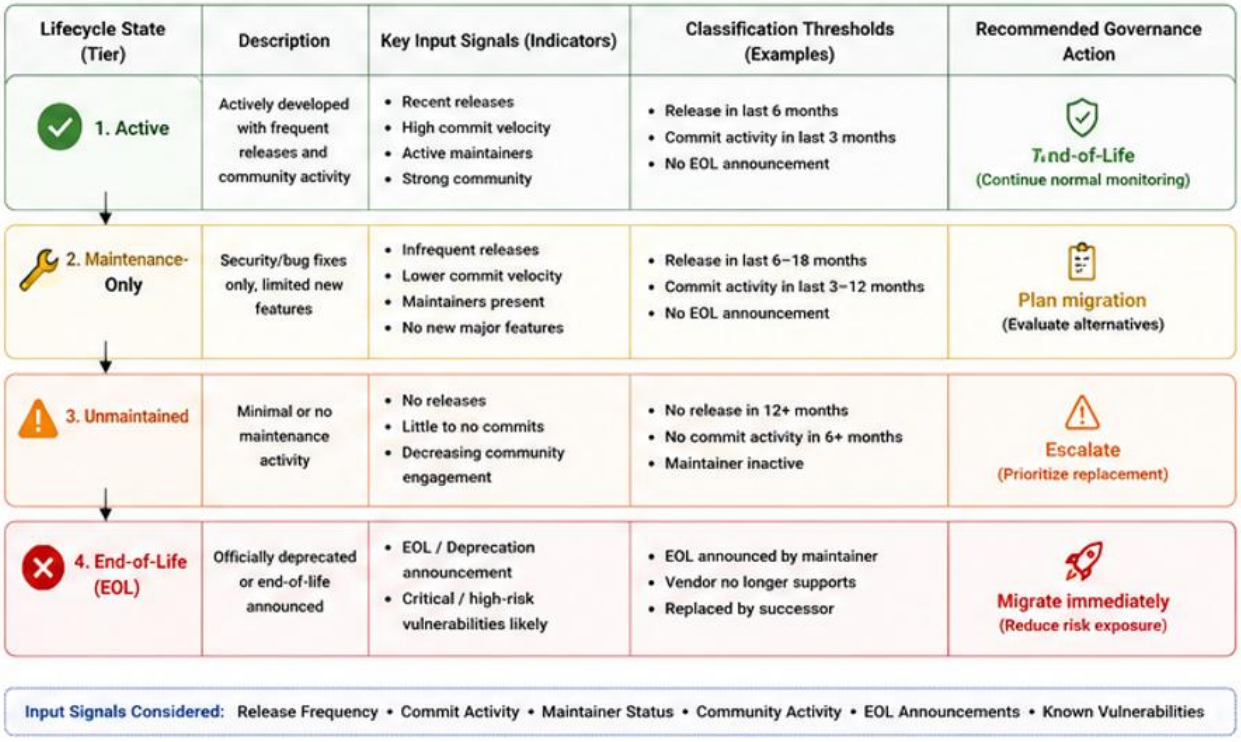

Figure 4: RIS Classification by Risk Level Across Ten Applications

9. Advantages

CAIS delivers measurable advantages over existing point-solution SCA tools. Centralized monitoring eliminates the visibility fragmentation that arises when teams operate independent scans across individual projects. The platform gives security and governance teams a real-time, portfolio-wide view of risk that enables prioritization based on actual organizational context rather than generic severity scores. This directly resolves the enterprise-wide visibility limitation documented in comparative studies of leading SCA tools [9].

Proactive lifecycle management is a core differentiator. In addition to vulnerability information, CAIS provides information about the EOL date, allowing teams to be warned before the dependency becomes EOL. This reduces the probability of operating unsupported software in production, a growing compliance concern for organizations subject to frameworks such as NIST SP 800-218 [5] and the ENISA supply chain security guidelines [6]. The introduction of the DFI metric makes

technical debt in the dependency portfolio visible and measurable, giving engineering leaders a quantitative basis for investment decisions about upgrade work.

The RIS metric directly reduces the barrier to remediation that prior research identifies as the primary reason patches are deferred [15]. By providing a structured, pre-computed estimate of upgrade risk before changes are attempted, CAIS replaces the informal judgment calls that teams currently make with objective, reproducible data. The modular architecture also allows organizations to integrate with existing DevSecOps tooling and to adopt individual capabilities in an incremental manner, which lowers the risk typical of deploying enterprise security platforms.

10. Limitations

CAIS relies on access to organizational source repositories and build systems. In organizations with strong access controls, legacy build tooling, or more than one build repository, CAIS may take longer to onboard new applications. Integration complexity grows with the diversity of package ecosystems in use, and rare or internally developed manifest formats may not be immediately supported.

The accuracy of vulnerability data depends on the quality and currency of the underlying databases. The NVD interval between CVE disclosure and CVE publication is documented [2]. However, the CAIS vulnerability list does not guarantee real-time updates to the public disclosure of vulnerabilities. The VRS metric is context-sensitive, but it is still subjectively tuned because business impact weights are manually determined. Future work can address both of these limitations by developing automated approaches for asset classification. Additionally, CAIS does not perform reachability analysis to determine whether or not a path containing a vulnerability is in fact reachable from elsewhere in the application. Therefore, CAIS can alert the user to vulnerabilities that cannot be exploited in the analysis context for the application during the scan [8]. This limitation shall be addressed in the future work.

11. Future Work

The most important projected enhancement to CAIS is risk prediction with artificial intelligence. Machine learning models would be trained with historical data on vulnerability releases, patch deadlines and organizations' upgrade behaviors to identify which dependencies are likely to develop critical vulnerabilities in the coming months and to warn users early. This would shift CAIS from reactive and proactive monitoring to true predictive intelligence, which would reduce the window of exposure before vulnerabilities are publicly disclosed [4, 18]. Reinforcement learning approaches may also be a particularly well-suited fit for sequencing remediation across a portfolio.

Future work will include integrating reachability analysis to enable filtering based on whether or not the vulnerable function or code path is reachable in the application's code (reducing false positives and improving remediation prioritization) [8]. Automated patch deployment pipelines are also being planned to enable CAIS to create pull requests for low-risk and low-RIS upgrades and other low-effort fixes without human user intervention, further reducing the time until vulnerabilities are remediated [14, 15]. Integration with existing policy-as-code platforms would enable organizations to model dependency governance policies as code and enforce them at every stage of their CI/CD pipeline [5, 7].

12. Conclusion

This paper presented CAIS, a centralized intelligence platform for enterprise dependency management, vulnerability detection, and lifecycle tracking. Three quantitative metrics were introduced: the Vulnerability Risk Score (VRS), the Dependency Freshness Index (DFI), and the Remediation Impact Score (RIS). Applied across ten representative enterprise applications, these metrics demonstrate that CAIS delivers context-aware, multi-dimensional risk insights that existing SCA tools do not provide. The VRS analysis identified two mission-critical applications as candidates for immediate repair. The DFI analysis of the sample showed that six of the 10 applications had dependencies that were at least 50% behind the current version. Two of the upgrades require formal approval and need to be completed in phases. However, RIS analysis shows that five applications can be upgraded with minimal coordination. In conclusion, the analysis shows that the three proposed metrics complement each other in ways that no single existing metric achieves. CAIS is a modular

and scalable reference architecture for enterprise software supply chain governance that integrates smoothly into the DevSecOps workflows, providing proactive, contextualized intelligence to help you manage dependency risk at scale in the organization. Future work will focus on building an AI-based predictive engine, reachability analysis, and auto-healing pipelines.

References

1. OWASP Foundation, "OWASP Dependency-Check," OWASP, 2023. Available: https://owasp.org/www-project-dependency-check/
2. National Institute of Standards and Technology, "National Vulnerability Database (NVD)," NIST, 2024. Available: https://nvd.nist.gov/
3. Alexandre Decan, Tom Mens, and Philippe Grosjean, "An Empirical Comparison of Dependency Network Evolution in Seven Software Packaging Ecosystems," Empirical Software Engineering, vol. 24, no. 1, pp. 381-416, 2019. Available: https://link.springer.com/article/10.1007/s10664-017-9589-y
4. Larissa Schmid et al., "Software Supply Chain Smells: Lightweight Analysis for Secure Dependency Management," arXiv preprint arXiv:2603.24282, 2026. Available: https://arxiv.org/abs/2603.24282
5. Murugiah Souppaya, Karen Scarfone, and Donna Dodson, "Secure Software Development Framework (SSDF) Version 1.1," NIST Special Publication 800-218, 2022. Available: https://doi.org/10.6028/NIST.SP.800-218
6. European Union Agency for Cybersecurity (ENISA), "Threat Landscape for Supply Chain Attacks," ENISA, 2021. Available: https://www.enisa.europa.eu/publications/threat-landscape-for-supply-chain-attacks
7. Mehdi Mirakhorli et al., "A Landscape Study of Open Source and Proprietary Tools for Software Bill of Materials (SBOM)," arXiv preprint arXiv:2402.11151, 2024. Available: https://arxiv.org/abs/2402.11151
8. Serena Elisa Ponta, Henrik Plate, and Antonino Sabetta, "Detection, Assessment and Mitigation of Vulnerabilities in Open Source Dependencies," Empirical Software Engineering, vol. 25, no. 5, pp. 3175-3215, 2020. Available: https://doi.org/10.1007/s10664-020-09830-x
9. Nasif Imtiaz, Seaver Thorn, and Laurie Williams, "A Comparative Study of Vulnerability Reporting by Software Composition Analysis Tools," in Proceedings of the 15th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM), pp. 1-11, 2021. Available: https://dl.acm.org/doi/abs/10.1145/3475716.3475769
10. Noah Wohler et al., "Committed to Trust: A Qualitative Study on Security and Trust in Open Source Software Projects," 2022. Available: https://doi.org/10.60882/cispa.24614139
11. Mahmoud Alfadel, Diego Elias Costa, and Emad Shihab, "Empirical Analysis of Security Vulnerabilities in Python Packages," Empirical Software Engineering, vol. 28, no. 3, p. 59, 2023. Available: https://link.springer.com/article/10.1007/s10664-022-10278-4
12. Kaixuan Li et al., "Patchfinder: A Two-Phase Approach to Security Patch Tracing for Disclosed Vulnerabilities in Open-Source Software," in Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis, pp. 590-602, 2024. Available: https://dl.acm.org/doi/abs/10.1145/3650212.3680305
13. Riivo Kikas, Georgios Gousios, Marlon Dumas, and Dietmar Pfahl, "Structure and Evolution of Package Dependency Networks," in 2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR), pp. 102-112, IEEE, 2017. Available: https://ieeexplore.ieee.org/abstract/document/7962360/
14. Bodin Chinthanet et al., "Lags in the Release, Adoption, and Propagation of npm Vulnerability Fixes," Empirical Software Engineering, vol. 26, no. 3, p. 47, 2021. Available: https://link.springer.com/article/10.1007/s10664-021-09951-x
15. Raula Gaikovina Kula et al., "Do Developers Update Their Library Dependencies? An Empirical Study on the Impact of Security Advisories on Library Migration," Empirical Software Engineering, vol. 23, no. 1, pp. 384-417, 2018. Available: https://link.springer.com/article/10.1007/s10664-017-9521-5

16. Gias Uddin et al., "Understanding How and Why Developers Seek and Analyze API-Related Opinions," IEEE Transactions on Software Engineering, vol. 47, no. 4, pp. 694-735, 2019. Available: https://ieeexplore.ieee.org/abstract/document/8658125
17. Alexandre Decan, Tom Mens, and Eleni Constantinou, "On the Impact of Security Vulnerabilities in the npm Package Dependency Network," in Proceedings of the 15th International Conference on Mining Software Repositories, pp. 181-191, 2018. Available: https://dl.acm.org/doi/abs/10.1145/3196398.3196401
18. Fangyuan Zhang et al., "Does the Vulnerability Threaten Our Projects? Automated Vulnerable API Detection for Third-Party Libraries," IEEE Transactions on Software Engineering, vol. 50, no. 11, pp. 2906-2920, 2024. Available: https://ieeexplore.ieee.org/abstract/document/10666791
19. Simone da Silva Amorim et al., "How Has the Health of Software Ecosystems Been Evaluated? A Systematic Review," in Proceedings of the XXXI Brazilian Symposium on Software Engineering, pp. 14-23, 2017. Available: https://dl.acm.org/doi/abs/10.1145/3131151.3131174
20. Marc Oriol et al., "Comprehensive Assessment of Open Source Software Ecosystem Health," Internet of Things, vol. 22, p. 100808, 2023. Available: https://www.sciencedirect.com/science/article/abs/pii/S2542660523001312