2026 Volume: 02 Issue: 11 www.ijrpa.com ISSN 2456-9995 Research Article

International Journal Research Publication Analysis

Page: 01-14

SOFTWARE VERSION COMPLIANCE AND RELEASE APPROVAL USING SERVICENOW

***Shashi Prabhat Ranjan, V. Yasaswini, K. Naveen Sai, K. Sagar, B. Varun**

Department of CSE (Artificial Intelligence & Data Science) GMRIT-Deemed to be University.

Article Received: 25 September 2026
Article Revised: 15 October 2026
Published on: 04 November 2026

***Corresponding Author: Shashi Prabhat Ranjan**
Department of CSE (Artificial Intelligence & Data Science) GMRIT-Deemed to be University.

DOI: https://doi-doi.org/101555/ijrpa.9409

ABSTRACT

Before a software release goes live, someone has to check a lot of things: which versions of Java, Spring Boot, the database and the frontend framework it uses, whether those versions work together, whether any of the libraries have known security holes, and whether the tests actually passed. Doing this by hand is slow, and things get missed. In this work we built ReleaseGuard AI, a release approval system on ServiceNow that does the first round of these checks automatically. A developer raises a release request and either types in the component details or uploads the pom.xml and package.json files. A Spring Boot service reads those files and pulls out the versions. The system then compares them with the organization's version rules, checks compatibility, looks for known vulnerabilities (CVEs) and reviews the test and build evidence. From this it gives the release a risk level and one of three recommendations: Approved, Needs Review or Blocked. The recommendation goes into a ServiceNow approval workflow, where a release manager makes the final call and every step is logged. We tried the prototype on simulated release records and sample Java and React configurations. A release that used a Log4j version affected by CVE-2021-44228 was blocked automatically, and a low-risk release with a readiness score of 95 was marked eligible for approval. The system is meant to help reviewers, not replace them. We have not yet measured how it compares with manual checking, and we leave that for future work.

KEYWORDS: ServiceNow, Version Compliance, Release Management, Risk Assessment, Security Compliance.

INTRODUCTION

Almost every organization today runs on software, and almost every piece of software is put together from smaller parts: a programming language, a framework such as Spring Boot, a database, an operating system, a frontend library and a long list of third-party packages. Each of these parts comes in many versions. Before a new release goes to production, the team needs to be sure that the versions in use are approved, that they work with each other, that they are free of known security problems and that the application has been tested properly. If any of this is skipped, the result can be a crash, a security incident or a long night for the support team.

In most places these checks are still done by hand. Someone opens the build files, looks up each version, reads a security report, asks QA whether the tests passed and then chases approvals by email. It works, but it is slow and it depends on people not making mistakes. The information also ends up scattered across spreadsheets, documents, chat messages and ticket systems, so nobody has a single view of where a release stands. A small oversight is enough to let trouble through. An old Java library with a publicly known vulnerability is a good example, and so is a framework version that does not work with the rest of the stack.

ServiceNow is a workflow platform that many companies already use to handle requests, approvals, notifications and records. It is a natural place to run release approvals, because it can send each request to the right reviewers, remember every decision and keep a history that can be audited later. What it does not do by itself is check the technical content of a release. That is the gap this project tries to fill.

We built ReleaseGuard AI, a system that works inside ServiceNow and combines both sides. The project, **Software Version Compliance and Release Approval Using ServiceNow**, collects the release details, reads the component versions from the project files, checks them against compliance rules, and combines the security and quality findings into a recommendation. A Spring Boot service reads pom.xml and package.json files and extracts the versions of Java, Spring Boot, databases, operating systems, frontend frameworks and third-party dependencies. The recommendation then goes to the ServiceNow approval workflow, where reviewers can see the details, spot unresolved problems and make a decision.

We should be clear about what the system can and cannot do. Its results depend on the quality of the configuration files, how accurate the compliance rules are, how reliable the vulnerability data is and how complete the test evidence is. It supports the people who

approve releases. It does not guarantee that a release is secure or free of bugs, and any performance figures we report must come from real experiments.

What motivated this work was a simple question: how much of the routine checking can be handed to automation, while keeping a person in charge of the final decision? By putting configuration parsing, compliance checking, security and quality review, risk scoring and approval into one workflow, we wanted to show one practical answer.

LITERATURE SURVEY

A few earlier studies deal with parts of this problem. Marquez et al. [1] model the links between software components and their versions as a dependency network and use Satisfiability Modulo Theories to work out how a vulnerability spreads through direct and indirect dependencies. Their method depends only on the National Vulnerability Database (NVD), which can be incomplete or out of date, and it stops at reporting. It does not feed into any release approval process.

Other work looks at testing. Haas et al. [2] studied automated and manual testing in large industrial projects and found that Pareto testing and test-case prioritization can shorten test runs while still catching most faults. They also point out that manual testing is hard to optimize and that what works depends a lot on the organization. Natarajan [3] uses AI and natural language processing to generate test cases and keep them working through self-healing scripts, but admits that biased data and poor interpretability are still open problems.

Alsarraj et al. [4] generate test paths from software design models and judge them by coverage. The paths are only as good as the models they come from. Srinivas and Nadimpalli [5] look at data-driven and model-based test automation and find that it improves defect detection and efficiency, but that keeping automated scripts up to date takes real effort.

Taken together, these studies show that dependency analysis and test automation both cut manual work and bring risks to light earlier. But they treat the problems separately. A dependency study tells you what is vulnerable and leaves the decision to someone else. A testing study tells you about quality but does not connect it to version policy or security findings. Almost none of it ties technical checks to a formal approval and audit process.

Table 1 sets the surveyed studies next to our system. Each of them covers one piece of the problem, and none connects its findings to an approval workflow.

Table 1: Comparison of the surveyed studies with the proposed system.

Ref	Focus	Technique	Limitation noted	Approval workflow
[1]	Vulnerability impact in software dependencies	Dependency-configuration analysis with Satisfiability Modulo Theories	Relies only on the NVD, which may be incomplete or outdated	No
[2]	Automated and manual testing in industry	Pareto testing and test-case prioritization	Manual testing is hard to optimize; results depend on organization	No
[3]	AI-generated test automation	NLP-based test generation, self-healing tests	Data bias and limited interpretability	No
[4]	Test generation from design models	Model-Based Testing with UML rule flow models	Depends on accuracy of input models	No
[5]	Automation in quality assurance	Data-driven and model-based testing	Script maintenance overhead	No
Proposed	Version compliance and release approval	Configuration parsing, rule-based compliance, CVE check, risk scoring	Depends on rule accuracy and vulnerability data quality	Yes (ServiceNow)

ReleaseGuard AI tries to close these gaps by:

- Reading component and dependency versions straight from pom.xml and package.json files.
- Checking those versions against the organization's compliance and compatibility rules.
- Turning vulnerability, version and quality evidence into one risk level and an Approved, Needs Review or Blocked recommendation.
- Sending the recommendation through a ServiceNow approval workflow that keeps a full audit trail.
- Leaving the final decision with a human release manager.

NOVELTY

- **One Combined Check**

Version compliance, compatibility, security and quality evidence are evaluated together in a single workflow, instead of in separate tools and documents.

- **A Clear Recommendation**

The findings are turned into a risk level and one of three outcomes (Approved, Needs Review or Blocked), so reviewers know where to start.

- **Automatic Reading of Project Files**

A Spring Boot service reads pom.xml and package.json, so developers do not have to type in every dependency.

- **Rules the Organization Controls**

Administrators set rules such as the minimum supported versions of Java, Spring Boot, React and MySQL and a code-coverage threshold. Each rule has a weight and is either blocking or advisory.

- **Approval with a Person in Charge**

Automated checks feed the ServiceNow approval workflow, where a release manager confirms the decision and every action is logged.

WORKING

ReleaseGuard AI runs on ServiceNow, with a Spring Boot integration service that handles configuration parsing. In short, a developer submits a release request, the project files are parsed, four modules check the extracted data, a risk module pulls their results together, and the ServiceNow workflow takes the release through review and approval to production. Fig-1 shows the architecture, and Table 2 lists the main components.

Table 2: Main components of ReleaseGuard AI.

Component	Role in the system
ServiceNow platform	Stores release requests, runs the approval workflow, sends notifications and keeps the audit trail
Spring Boot integration service	Parses pom.xml and package.json and extracts component and dependency versions
pom.xml / package.json	Source files for Maven-based Java and Node.js applications
Compliance rules	Minimum supported versions and coverage threshold; each rule has a weight and is blocking or advisory
Vulnerability information (CVE)	Identifies dependencies with known security weaknesses
Role-based portals	Separate views for developers, administrators, release managers and security reviewers

1. Release Request and Developer Input

The process starts when a developer raises a release request in ServiceNow. They enter the application name, release version, description, target environment, deployment date and technology stack, and attach the supporting files. Each request gets a unique ID so it can be followed from submission to closure.

2. Configuration File Parsing

A Spring Boot integration service reads pom.xml for Maven-based Java applications and package.json for Node.js applications. From these it extracts package names, dependency versions and other metadata, and hands them to the validation modules. Developers can also type the component details in by hand. Before the analysis runs, the system checks that the information is complete.

3. Version Compliance Check

Here the extracted versions are compared with the organization's approved version lists and policies, for example the minimum supported versions of Java, Spring Boot, React and MySQL. Anything outdated, unsupported or not on the approved list is flagged and recorded so it can be corrected.

4. Security and Vulnerability Check

Each dependency is checked against vulnerability information such as the Common Vulnerabilities and Exposures (CVE) database. If a library has a known critical vulnerability, the module reports it so that the developers can update or replace that dependency.

5. Compatibility Check

The compatibility rules look for version conflicts, unsupported combinations and mismatched dependencies. A simple example is a library that needs a newer Java version than the one the application uses. Catching this early avoids runtime errors and integration trouble after deployment.

6. Quality Evidence Evaluation

The system looks at the test results, build status, QA approval and code coverage. If the build has failed or a mandatory test has not passed, the release is flagged for investigation before it can reach the approval stage.

7. Risk Assessment and Decision

The results of the four checks are combined using predefined rules and weights, and every rule is either blocking or advisory. The module then assigns a risk level and one of three recommendations. A release with a critical vulnerability is Blocked, a release with a minor

policy exception goes to manual review, and a release that meets all the blocking rules is eligible for approval. Table 3 summarizes how this works.

8. ServiceNow Approval and Audit

The recommendation starts the ServiceNow approval workflow. The right reviewers are notified, the approval status is tracked, and each action and decision is written to an audit trail. If a release fails the checks or is rejected, it goes back to the developer to fix and resubmit.

9. Release Manager Decision and Production Deployment

The release manager looks over the compliance, security, compatibility and quality findings, adds comments if needed and confirms the final decision. Only a release with the required approval goes on to production. Blocked releases, and those still waiting for a mandatory approval, are held back. Once deployment is done, stakeholders are notified and the request is closed in ServiceNow.

10. Output Screens and Results

The prototype has separate portals for developers, administrators, release managers and security reviewers (Figs. 2 to 8). In our demonstration, Payment Service v2.4.0 scored 95/100 for readiness with low risk and was eligible for approval. Inventory Portal v3.1.0 scored 73/100 with high risk and needed changes. Legacy Billing v1.9.0 scored 50/100 with critical risk and was blocked because its Log4j dependency has a critical vulnerability (CVE-2021-44228). The approval gate for that release stays locked until the vulnerability is fixed and the security checks are done. Administrators use their portal to manage users, compliance rules and security advisories.

11. Testing and Performance Evaluation

We ran the prototype on simulated enterprise release records and sample Java and React configurations, and the dashboards shown here were filled with sample data. Performance should be judged on detection accuracy, processing time and how well high-risk releases are caught, compared with manual verification. Any numbers reported must come from real measurements, not estimates. We have not yet carried out this evaluation.

12. Overall System Workflow

From start to finish, a release moves from the developer’s request through parsing, the four checks, risk assessment and the ServiceNow approval workflow, and ends with production deployment and closing the request. The idea is to let automation do the repetitive checking while people keep control of the decision.

Table 3: Risk-based recommendation outcomes.

Outcome	Typical condition	Effect in the workflow
Approved	All blocking rules are met, no critical vulnerabilities, and quality evidence is satisfactory	Release is eligible for approval and, once the release manager confirms, may proceed to production
Needs Review	Minor policy exception, advisory rule violation or elevated risk	Routed to the security or release team for manual review, or returned for changes
Blocked	Critical security vulnerability or a violated blocking rule	Approval gate is locked and the release returns to the developer for correction

Overall, the system is a ServiceNow-based framework for checking version compliance and approving releases. How reliable it is depends on accurate compliance rules, trustworthy vulnerability data and complete test evidence, so it should be seen as a decision-support tool for reviewers. It cannot promise that a release is secure or error-free.



Fig-1: Proposed system architecture of ReleaseGuard AI.

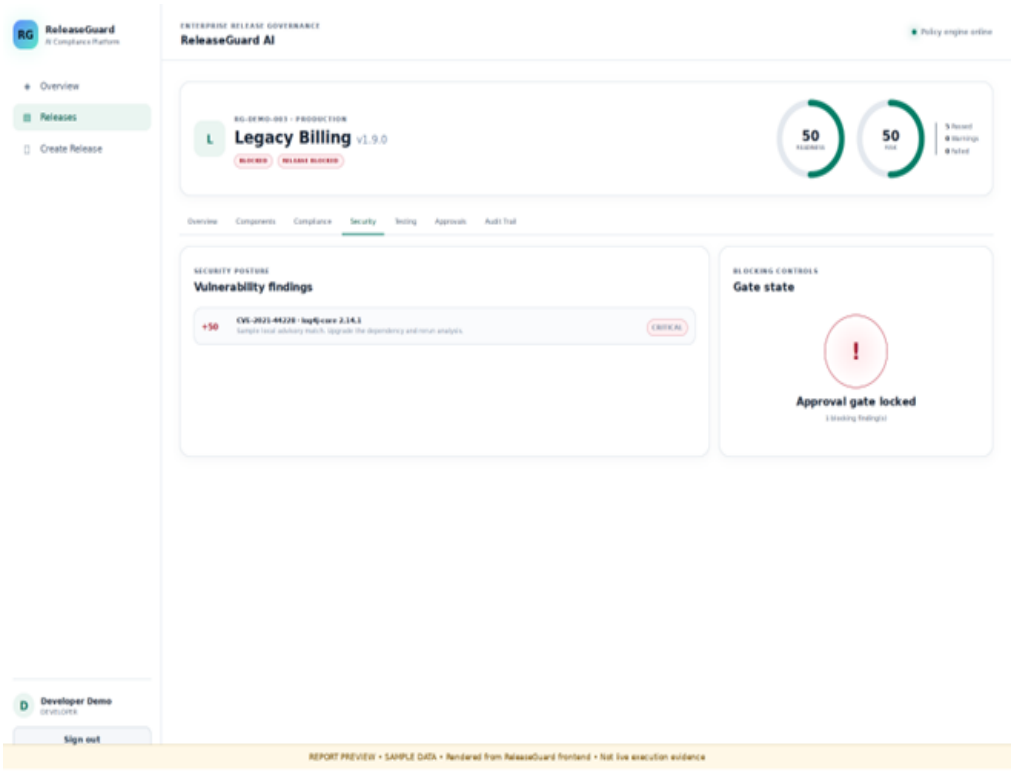


Fig-2: Developer portal – release blocked because of a critical vulnerability. (CVE-2021-44228)

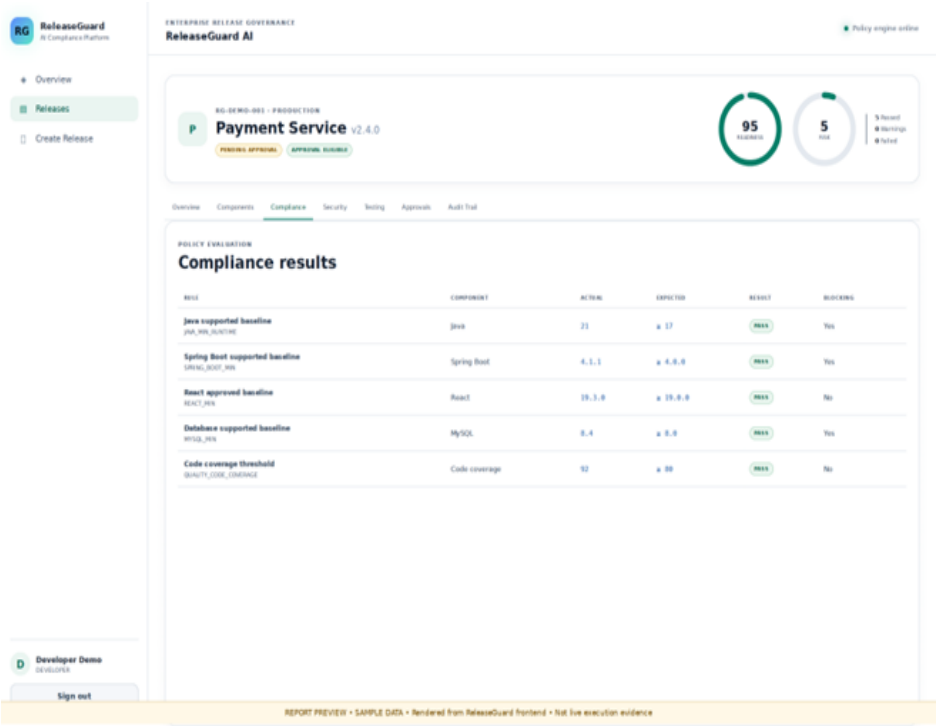


Fig-3: Developer portal – compliance results of the Payment Service release.

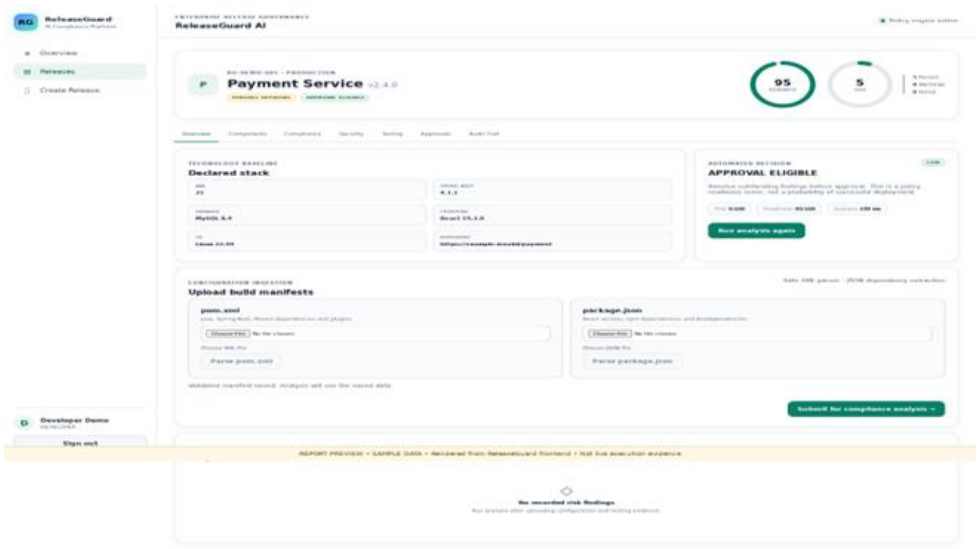


Fig-4: Developer portal – uploading pom.xml and package. json files.

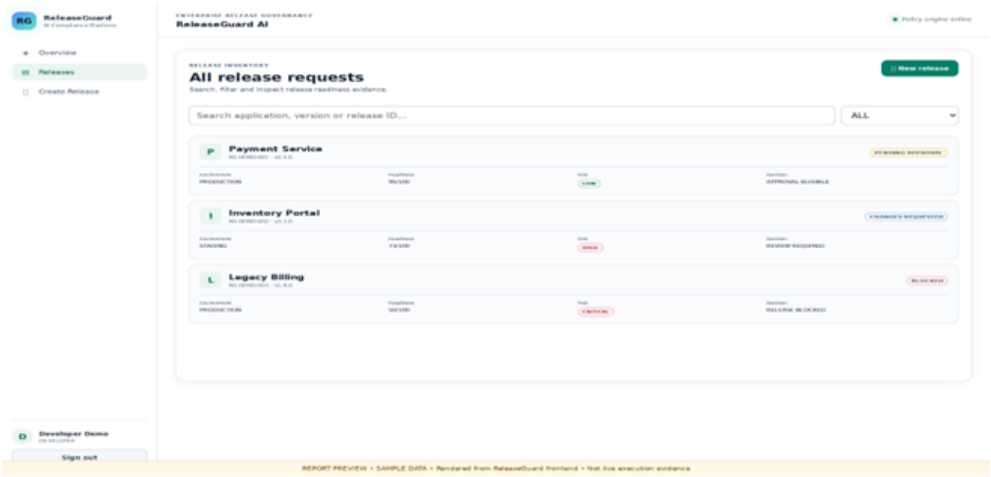


Fig-5: Developer portal – tracking release requests with readiness score, risk level and decision.

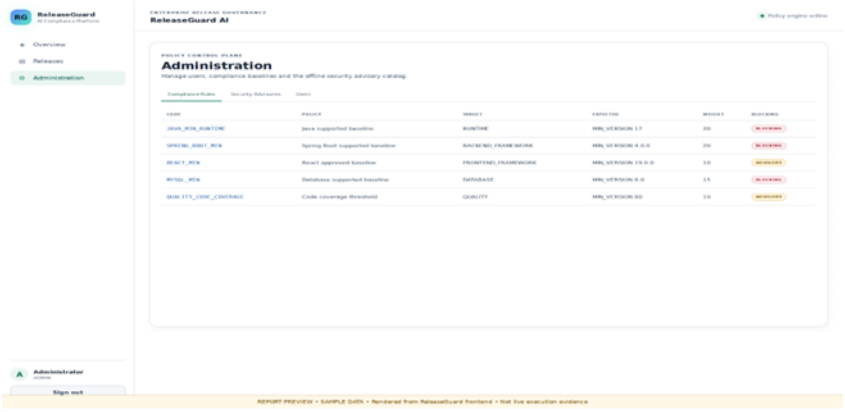


Fig-6: Administration portal – compliance rules with weights and blocking or advisory classification.

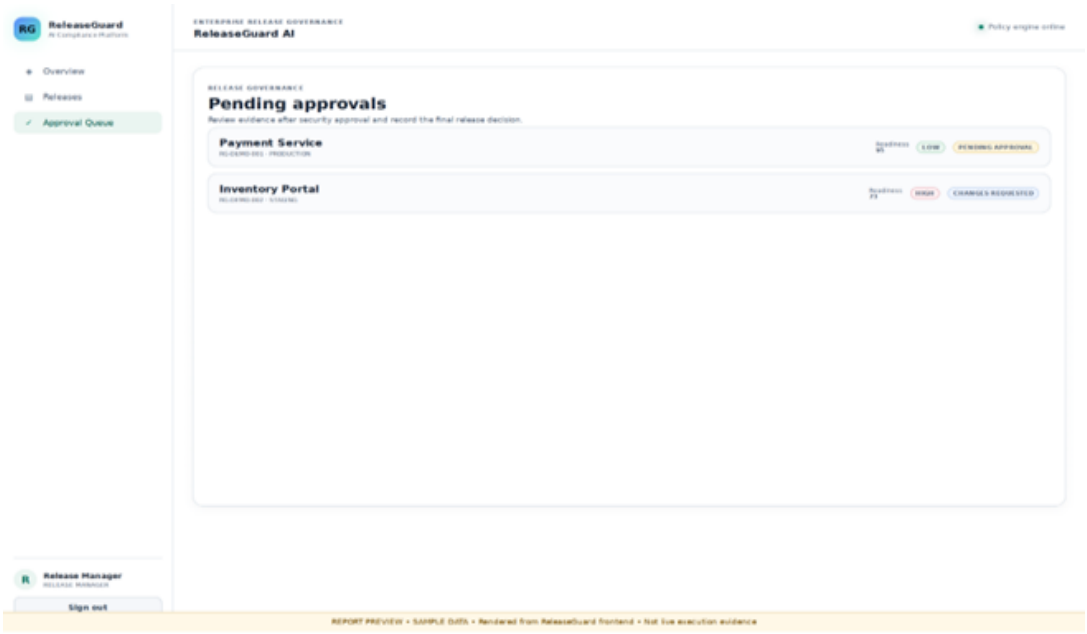


Fig-7: Release management portal – pending approvals.

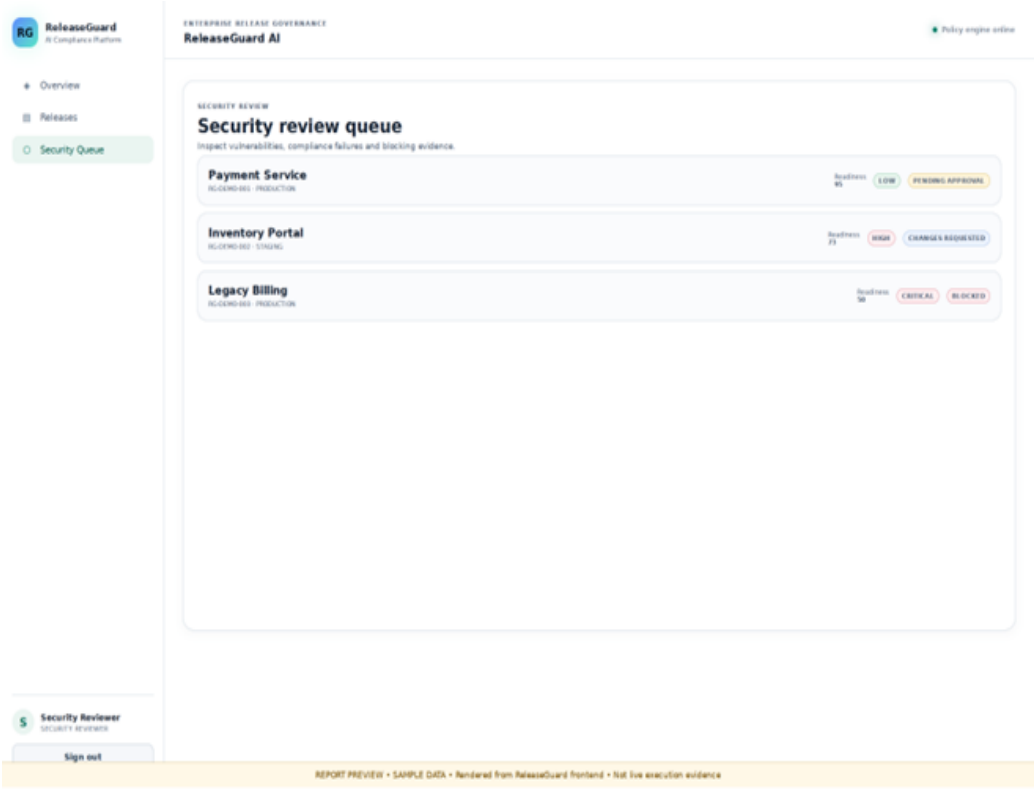


Fig-8: Security portal – security review queue.

FLOW CHART

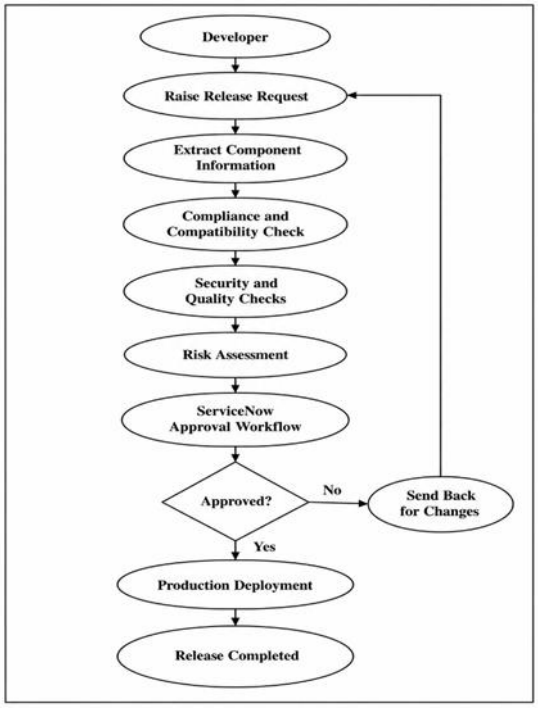


Fig-9: Workflow diagram of the software version compliance and release approval system.

ADVANTAGES AND LIMITATIONS:

Advantages. The biggest gain is that people no longer have to open build files and look up versions one by one, because the system reads them directly. The same rules are applied to every release, so the checks are consistent, and outdated components, incompatible versions and known vulnerabilities show up before deployment instead of after it. Reviewers get one place to see a release's status, the decisions made and the deployment progress, and the audit trail shows who decided what and when. Blocking rules stop a release with a critical vulnerability from reaching production, while advisory rules let minor issues be flagged without holding up the work.

Limitations. The system is only as good as what it is given. It depends on the quality of the configuration files, on compliance rules that are accurate and kept up to date, on reliable vulnerability data and on complete test evidence. The ServiceNow workflows have to be set up correctly and connected to whatever security and quality scanners the organization uses. We demonstrated the prototype on simulated release records and sample configurations, so we do not yet know how accurate or how fast it is on real release data compared with manual verification. Its recommendations help reviewers decide. They do not by themselves guarantee that a release is secure or error-free.

FUTURE SCOPE

The system gives a starting point for controlled, automated release governance, and there is plenty of room to build on it. The most useful next step is a proper evaluation against manual verification, measuring detection accuracy, processing time and how well high-risk releases are blocked. Without that, we cannot say how much the approach really helps.

Machine learning could be added to predict release failures, spot unusual deployment patterns and estimate risk from historical release data. Connecting the system to vulnerability scanners and software composition analysis tools would allow automatic detection of known vulnerabilities, outdated dependencies and licence problems.

It could also be tied into CI/CD pipelines, so that compliance checks and approval gates run throughout development and not only at release time. Automated rollback to the last stable version when a deployment fails, and real-time dashboards and notifications for compliance violations, approval status and deployment progress, would make it more useful day to day. Finally, the system could be extended to handle many applications, distributed teams and cloud environments.

CONCLUSION

In this project we built ReleaseGuard AI, a ServiceNow-based system that checks software version compliance and manages release approval. It reads component information from configuration files, checks version compliance and compatibility, reviews security and quality evidence, scores the risk and passes the result through an approval workflow, so that a release meets the organization’s requirements before it reaches production. By catching outdated components, incompatible versions and known vulnerabilities early, it helps lower deployment risk.

The ServiceNow workflow lets authorized people review requests, approve eligible releases and send rejected ones back to developers. Central tracking and audit records make the process more transparent and easier to trace. The demonstration shows that automation can take over much of the repetitive checking and support more consistent, controlled deployments.

The system still has limits. Its reliability depends on accurate rules, reliable vulnerability data and complete test evidence, and its benefits over manual verification remain to be measured. It is meant to assist release reviewers, not to replace their judgement or to guarantee that a release is secure or error-free.

REFERENCES

1. A. German Marquez, A. J. Varela-Vaca, M. T. Gómez López, J. A. Galindo and D. Benavides, “Vulnerability impact analysis in software project dependencies based on Satisfiability Modulo Theories (SMT),” Computers & Security, vol. 139, Art. no. 103669, Apr. 2024.
2. R. Haas, R. Nömmer, E. Jürgens and S. Apel, “Optimization of Automated and Manual Software Tests in Industrial Practice: A Survey and Historical Analysis,” IEEE Transactions on Software Engineering, vol. 50, no. 8, pp. 2005–2020, Aug. 2024.
3. D. R. Natarajan, “AI-Generated Test Automation for Autonomous Software Verification: Enhancing Quality Assurance Through AI-Driven Testing,” Journal of Cognitive Computing and Neural Intelligence, vol. 1, no. 1, 2026.
4. R. Gh. Alsarraj, A. M. Altaie and E. Z. Majeed, “Developing an Automated Model-Based Software Testing Tool From the Design Phase,” IEEE Access, vol. 13, pp. 58549–58557, Mar. 2025.
5. N. Srinivas and S. V. Nadimpalli, “Leveraging Automation in Software Quality Assurance: Enhancing Defect Detection and Improving Efficiency,” International Journal of Acta Informatica, pp. 112–121, 2024.