NextGen Scientific Publication
American International Journal of Computer Science and Technology

Volume 4 Issue 5, Pg. No. 61-68, AIJCST-V4I5P107, 2022
https://doi.org/10.63282/3117-5481/AIJCST-V4I5P107

Original Article

Vulnerability Prioritization and Patch Scheduling Under Cloud Deployment and Validation Constraints

***Surya Kiran Yelagam**
Independent Researcher, McKinney, Texas, USA.

Abstract:

Cloud teams must decide both which security update is urgent and when it can be deployed safely. This paper proposes a scheduling record that keeps urgency separate from deployment readiness. The record combines exposure and service criticality with dependency checks, validation capacity, maintenance-window availability, and rollback readiness. An illustrative maintenance-window decision shows how to document a temporary deferral, its owner, interim mitigation, and reassessment time. The proposal has not been evaluated against operational outcomes.

Keywords:

Cloud Security, Vulnerability Prioritization, Patch Scheduling, Deployment Constraints, Service Dependencies, Validation Effort, Rollback Feasibility.

Article History:

Received: 06.08.2022

Revised: 21.08.2022

Accepted: 06.09.2022

Published: 19.09.2022

1. Introduction

Software vulnerabilities may arise from outdated packages, insecure configurations, unpatched operating systems, third-party libraries, and application components. Patching reduces known exposure, but it also changes production systems. Teams therefore need to consider testing, validation, resource availability, service coordination, and maintenance windows before deployment. Severity and exploitability provide a useful starting point for prioritization, but they do not by themselves establish that a patch can be deployed safely in a particular cloud service.

Prior work supports prioritization that goes beyond a severity score. SmartPatch [1] provides a structured approach to patch prioritization, and Olswang et al. [2] show that network relationships can affect the order in which vulnerabilities should be addressed. For cloud services, dependencies among applications, infrastructure components, and shared platforms can also affect both deployment order and the scope of validation.

Patch planning also draws on context-specific assessment. Jiang and Atif [4] examine vulnerability severity in cyber–physical systems. Their setting differs from a cloud maintenance window, where compatibility testing and recovery capacity can determine when an update can be deployed.

Patil and Modi [5] examine vulnerability assessment and patching in a virtualized cloud environment. The scheduling question here is narrower: how should the service owner record an urgent update when a required validation step is incomplete?
The framework records two groups of inputs. Five inform urgency: vulnerability severity, exploitability, exposure, service criticality, and service dependencies. Four inform readiness: validation status, available capacity, maintenance-window feasibility, and rollback readiness.

2. Related Work

Research on vulnerability management covers categorization, risk assessment, patch selection, and remediation planning. Context-aware approaches are particularly relevant to cloud environments because service dependencies, maintenance constraints,

Copyright by the Author(s). This work is licensed under a Creative Commons Attribution-ShareAlike 4.0 International (CC BY-NC-ND 4.0) License (https://creativecommons.org/licenses/by-sa/4.0/)

and operational impact can change the appropriate remediation sequence. This paper builds on that literature by proposing a decision record for separating security urgency from deployment readiness.

Yadav et al. [1] propose SmartPatch for systematic patch prioritization. Olswang et al. [2] examine prioritization in large networks and show that network topology matters when selecting remediation order. Together, these works support considering the environment in which a vulnerability exists instead of relying on a vulnerability score alone.

Shah et al. [3] analyze factors that influence vulnerability selection for remediation. Jiang and Atif [4] address automated discovery and assessment of vulnerability severity in cyber-physical systems. These studies inform the risk-assessment inputs used here, although the proposed framework addresses cloud maintenance decisions rather than a measured remediation system. Angelini et al. [6] present Vulnus, a visual analytics approach to vulnerability analysis for network security.

Lee et al. [7] present a package-based vulnerability evaluation and patch framework. Al-Ayed et al. [8] describe automated support for security-vulnerability management. Their work motivates a structured record for identifying affected components, dependencies, validation requirements, and planned actions.

Ahmadi et al. [9, 10] study normalization for context-aware vulnerability risk management, including cloud settings. Zhang and Li [11] add a scheduling perspective for changing security risk. These studies provide a bridge from prioritizing vulnerabilities to deciding when a patch can be deployed.

Fruhwirth and Mannisto [12] argue that CVSS should be supplemented with contextual information. Zhang et al. [13] and Wang et al. [14] explore machine-learning approaches to remediation analysis and patch classification. Walkowski et al. [15] further examine context-aware CVSS-based prioritization. The proposed framework does not implement these automated methods. It uses a human-owned decision record that can later be extended with validated automation.

Practitioner guidance and public data sources address the same separation. The Stakeholder-Specific Vulnerability Categorization replaces a single score with a decision tree whose outcomes are response actions [16], following the argument that a CVSS severity score alone is a poor basis for prioritization [17]. The Exploit Prediction Scoring System estimates the probability that a vulnerability will be exploited in the wild [18], and the CISA Known Exploited Vulnerabilities catalog lists vulnerabilities with confirmed exploitation [19]. NIST guidance on enterprise patch management treats patching as preventive maintenance that must be planned around testing, deployment, and verification [20]. These sources inform the urgency inputs used here. The proposed record adds the readiness side: why an urgent item was or was not deployed in a given window.

3. Constraint-Aware Patch Scheduling Framework

This paper proposes the Constraint-Aware Patch Scheduling Framework (CAPSF). The framework is a decision-record model for patch management that combines security urgency with operational readiness. It does not calculate a universal priority score or automate deployment. Instead, it requires the service owner to document exposure, dependencies, validation status, maintenance-window feasibility, and rollback readiness before scheduling a patch. Figure 1 shows the relationship among these proposed decision inputs.

Patch management is a coordinated decision process. Security urgency reflects vulnerability severity, exploitability, exposure, and service criticality. Deployment readiness reflects dependency analysis, validation planning, available capacity, maintenance-window feasibility, and rollback preparation. The framework helps a reviewer record why a patch is scheduled, deferred, or escalated. It is a conceptual model that could be adapted for public, private, hybrid, or multi-cloud environments.

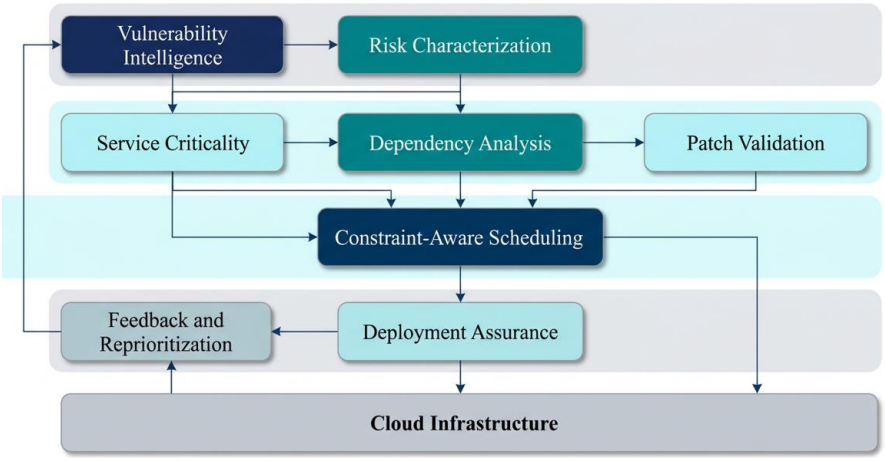

Figure 1. Overall Structure of the Constraint-Aware Patch Scheduling Framework

3.1. Cloud Vulnerability Intelligence and Risk Characterization

The first layer records the known vulnerability and the available remediation. Inputs may include security scanners, cloud security posture-management platforms, software-composition analysis, container-image scanners, operating-system feeds, and vendor advisories. For each item, the record identifies affected versions, patch availability, severity, exploitation indicators, exposure, and the recommended remediation.

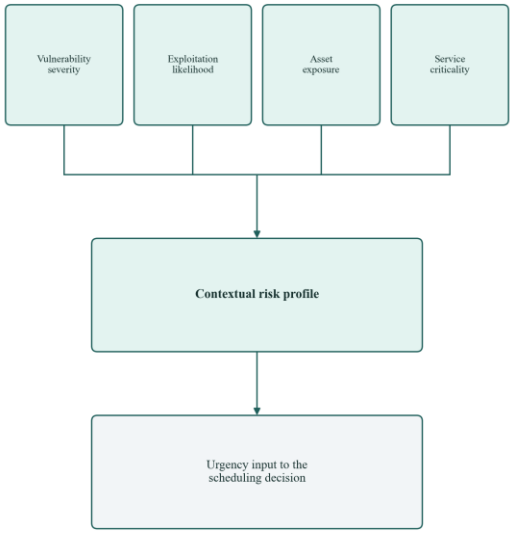

Figure 2. Multi-Dimensional Vulnerability Prioritization Model

Figure 2 shows how severity, exploitation likelihood, asset exposure, and service criticality contribute to contextual prioritization. The framework records these inputs for each vulnerability and preserves the reasoning behind the priority decision.

The framework does not treat severity as the sole priority signal. It uses the resulting contextual risk profile as an input to service-criticality assessment and scheduling. A high-urgency item can still be unready for deployment when required validation or rollback preparation is incomplete.

3.2. Service-Criticality and Dependency-Aware Assessment

The second layer assesses the criticality of the affected cloud resource. Cloud applications commonly depend on virtual machines, containers, databases, API gateways, authentication services, storage systems, and network services. Patching one

component can affect dependent services or temporarily interrupt a business function. The record therefore identifies both dependency relationships and service criticality.

Service criticality is determined by business importance, user impact, data sensitivity, availability needs, and the component's role in the overall service design. This layer distinguishes standalone, interconnected, and business-critical resources. It records whether deployment should be isolated, sequenced, or coordinated with related components. That service context is carried forward to the scheduling decision.

3.3. Patch Compatibility and Validation-Effort Estimation

The third layer assesses technical readiness for deployment. A patch can address a security issue while also changing configuration, compatibility, or performance characteristics. Validation is therefore required after urgency has been assessed and before production deployment. The decision record identifies the tests needed to establish readiness.

Validation requirements depend on the component, patch characteristics, dependent services, and expected operational effect. They may include compatibility checks, configuration review, functional testing, integration testing, security verification, and service-health assessment. Shared databases, authentication systems, and runtime platforms generally require broader validation than isolated components because a change can affect multiple services. The framework records the effort, capacity, and time required before the patch is considered ready.

3.4. Constraint-Aware Patch Prioritization and Scheduling

The framework separates two questions. First, how urgent is remediation given severity, exploitability, exposure, service criticality, and dependencies? Second, is deployment ready given validation status, available capacity, maintenance-window feasibility, and rollback readiness? Both questions inform the schedule, but neither replaces the service owner's documented decision.

The scheduling record applies four principles. High exposure or credible exploitation increases urgency. Business-critical and highly dependent services require coordinated planning. Patches needing substantial validation require sufficient time before production rollout. Deployment should occur in a maintenance window that can accommodate the change, verification, and any required rollback. Figure 3 illustrates the dependency information considered in this decision.

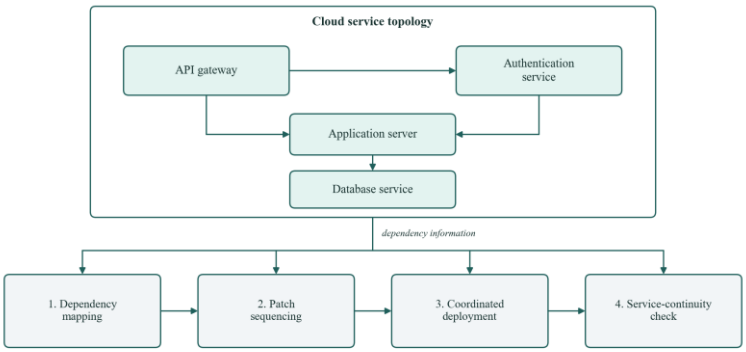

Figure 3. Dependency-Aware Cloud Patch Scheduling Workflow

Testing capacity can limit the number of patches validated in a deployment cycle. When capacity is insufficient, the record compares urgency, validation effort, and the resources required. The highest-risk items can be escalated for additional capacity or an emergency window. Lower-readiness items remain deferred with an owner, interim mitigation, and a specific reassessment time.

Maintenance-window feasibility is a separate constraint. The record estimates the time needed for deployment, verification, and rollback. If the available window is insufficient, the owner can select a later window, split a coordinated rollout when technically safe, or request an emergency window.

Dependencies also affect deployment order. When a component depends on another service, the record identifies compatibility and readiness requirements and avoids concurrent changes that would obscure the source of a failure.

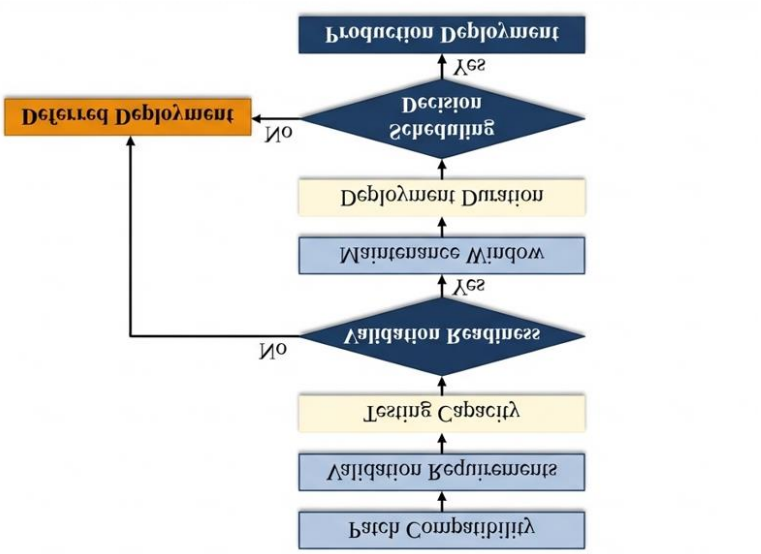

Figure 4. Validation and Deployment Feasibility Decision Model

Figure 4 shows how testing capacity, maintenance-window availability, and patch readiness influence the proposed production-scheduling decision.

3.5. Rollback Readiness and Deployment Assurance

The fifth layer addresses rollback readiness for potentially disruptive deployments. Readiness is not established merely because a rollback is theoretically possible. The service owner must record the restoration path, responsible operator, recovery time constraint, and verification needed to confirm service recovery.

Rollback readiness can include current backups or snapshots, configuration recovery, a prior software version, database-recovery procedures, and a documented rollback plan. The record also identifies dependent services and the time needed to restore acceptable operation.

Deployment assurance includes pre-deployment checks, a controlled rollout, service-health monitoring, and post-deployment verification. Depending on the architecture, a team may use staged deployment, canary release, rolling update, or blue-green deployment. The selected approach and its success criteria should be recorded before the change begins.

If observed service behavior does not match the documented success criteria, the responsible operator pauses or halts the rollout and follows the recorded recovery plan. The decision record retains deployment status, validation evidence, mitigation steps, and the outcome of any rollback.

3.6. Feedback-Driven Reprioritization and Continuous Coordination

The final layer updates the decision record as conditions change. Exploit activity, external exposure, infrastructure configuration, patch availability, and business-service requirements can all alter the appropriate schedule.

The owner periodically reviews vulnerability intelligence and service context. Material changes update the contextual risk profile and can trigger an earlier reassessment of a deferred patch.

The feedback process does not assume that all patches can be deployed immediately. It keeps deferred vulnerabilities visible and makes the reason, owner, interim mitigation, and next review date auditable.

3.7. Integrated Framework Operation

The proposed process is: identify the vulnerability; record contextual risk; assess service criticality and dependencies; define validation; assess capacity, window, and rollback readiness; deploy in a controlled manner when ready; and reassess as conditions change. Each stage records evidence needed by the next decision. The process is a decision aid, not a measured operational result.

4. Illustrative Patch Scheduling Decision

4.1. Dependency-Aware Scheduling and Service Continuity

Consider an illustrative service whose urgent security update must be validated with a downstream client. The security owner records high urgency based on exposure and service context. The service owner records that compatibility testing is incomplete. Urgency remains high, while deployment readiness remains unresolved.

4.2. Validation Capacity and Maintenance-Window Feasibility

If the next maintenance window is too short for validation, deployment, and recovery, the owners can request an emergency window or record a time-limited deferral with an interim mitigation. The record identifies the accountable owner and a fixed reassessment time.

4.3. Rollback Feasibility and Operational Resilience

Before rollout, the owners define the service-health signals that would indicate regression, identify who can halt deployment, and verify the recovery path. This is especially important when a patch can change persistent-state formats or shared runtime behavior.

Table 1 shows the scheduling record for this illustrative case. Every value is illustrative and none is field data.

Table 1. Illustrative scheduling record

Field	Recorded value
Vulnerability	Remote-code-execution flaw in a shared runtime library
Severity	High
Exploitation likelihood	Elevated; public exploit code reported
Exposure	Service reachable by external clients
Service criticality	Business-critical
Dependencies	One downstream client must be validated with the update
Validation status	Compatibility testing with the downstream client incomplete
Available capacity	One validation slot before the next window
Maintenance window	Next window too short for validation, deployment, and rollback
Rollback readiness	Prior version retained; restoration path verified
Decision	Time-limited deferral; emergency window requested
Interim mitigation	Restrict external access to the affected endpoint
Owner	Service owner, with the security owner
Reassessment	Fixed date before the next maintenance window

4.4. Integrated Assessment and Practical Applicability

A future evaluation could compare exposure time, overdue deferrals, validation failures, and service disruption across comparable services. This illustrative example provides no measured improvement over existing practice.

5. Future Research Directions

5.1. Intelligent Service Dependency Discovery

Future work could improve dependency discovery across cloud services, containers, virtual machines, databases, and microservices. A validated topology model could help identify direct and indirect dependencies before a coordinated patch. Such automation would require evaluation for accuracy and operational safety.

5.2. Adaptive Validation Resource Optimization

Future work could evaluate optimization methods for allocating limited validation capacity. Candidate inputs include vulnerability urgency, estimated test effort, service criticality, concurrent validation demand, and maintenance-window constraints. Any recommendation system should preserve human review and record its basis for a proposed schedule.

5.3. Predictive Rollback and Autonomous Recovery

Future research could examine whether deployment-failure patterns and service-health indicators improve rollback planning. Any predictive approach would need evaluation on permitted operational evidence, clear error handling, and human control for production changes.

5.4. Multi-cloud and edge infrastructure adaptation

The proposed record could be adapted for public, private, hybrid, multi-cloud, and edge environments. Future work could evaluate which policy fields should vary by infrastructure, geography, maintenance constraints, and local vulnerability intelligence.

5.5. Compliance review and decision records

Future work could add compliance requirements and more structured decision explanations. A governance platform might retain vulnerability assessment, validation approval, deployment, rollback, and reassessment records. Such integration should improve traceability without claiming automated correctness.

These extensions would require separate design and evaluation. The present work remains a constraint-aware decision-record framework for documenting human patch-scheduling decisions.

6. Conclusion

A single severity score does not establish that an urgent update is ready to deploy. The proposed scheduling record preserves both the urgency assessment and the readiness assessment, including the reason for a deferral and the next review time.

The worked example illustrates a decision procedure. It does not establish earlier remediation, fewer incidents, or improved operational outcomes. Any empirical evaluation would require dated change records, a documented comparison policy, and permission to use de-identified operational evidence.

References

[1] Yadav, G., Gauravaram, P., Jindal, A. K., & Paul, K. (2022). SmartPatch: A patch prioritization framework. Computers in Industry, 137, Article 103595. https://doi.org/10.1016/j.compind.2021.103595

[2] Olswang, A., Gonda, T., Puzis, R., Shani, G., Shapira, B., & Tractinsky, N. (2022). Prioritizing vulnerability patches in large networks. Expert Systems with Applications, 193, Article 116467. https://doi.org/10.1016/j.eswa.2021.116467

[3] Shah, A., Farris, K. A., Ganesan, R., & Jajodia, S. (2022). Vulnerability selection for remediation: An empirical analysis. The Journal of Defense Modeling and Simulation, 19(1), 13–22. https://doi.org/10.1177/1548512919874129

[4] Jiang, Y., & Atif, Y. (2022). Towards automatic discovery and assessment of vulnerability severity in cyber–physical systems. *Array, 15*, Article 100209. https://doi.org/10.1016/j.array.2022.100209

[5] Patil, R., & Modi, C. (2019). Designing an efficient framework for vulnerability assessment and patching (VAP) in virtual environment of cloud computing. *The Journal of Supercomputing, 75*, 2862–2889. https://doi.org/10.1007/s11227-018-2698-6

[6] Angelini, M., Blasilli, G., Catarci, T., Lenti, S., & Santucci, G. (2019). Vulnus: Visual vulnerability analysis for network security. *IEEE Transactions on Visualization and Computer Graphics, 25*(1), 183–192. https://doi.org/10.1109/TVCG.2018.2865028

[7] Lee, J. H., Sohn, S. G., Chang, B. H., & Chung, T. M. (2009). PKG-VUL: Security vulnerability evaluation and patch framework for package-based systems. ETRI Journal, 31(5), 554–564. https://doi.org/10.4218/etrij.09.0108.0578https://doi.org/10.4218/etrij.09.0108.0578

[8] Al-Ayed, A., Furnell, S., Zhao, D., & Dowland, P. (2005). An automated framework for managing security vulnerabilities. *Information Management & Computer Security, 13*(2), 156–166. https://doi.org/10.1108/09685220510589334

[9] Ahmadi, V., Arlos, P., & Casalicchio, E. (2020). Normalization of severity rating for automated context-aware vulnerability risk management. 2020 IEEE International Conference on Autonomic Computing and Self-Organizing Systems Companion. https://doi.org/10.1109/ACSOS-C51401.2020.00056

[10] Ahmadi, V., Arlos, P., & Casalicchio, E. (2021). Normalization framework for vulnerability risk management in cloud. 2021 IEEE International Conference on Future Internet of Things and Cloud. https://doi.org/10.1109/FICLOUD49777.2021.00022

[11] Zhang, F., & Li, Q. (2020). Dynamic risk-aware patch scheduling. 2020 IEEE Conference on Communications and Network Security. https://doi.org/10.1109/CNS48642.2020.9162225
[12] Fruhwirth, C., & Mannisto, T. (2009). Improving CVSS-based vulnerability prioritization and response with context information. In *Proceedings of the 2009 International Symposium on Empirical Software Engineering and Measurement*.
[13] Zhang, F., Huff, P., McClanahan, K., & Li, Q. (2020). A machine learning-based approach for automated vulnerability remediation analysis. 2020 IEEE Conference on Communications and Network Security. https://doi.org/10.1109/CNS48642.2020.9162309
[14] Wang, X., Wang, S., Sun, K., Batcheller, A., & Jajodia, S. (2020). A machine learning approach to classify security patches into vulnerability types. 2020 IEEE Conference on Communications and Network Security. https://doi.org/10.1109/CNS48642.2020.9162237
[15] Walkowski, M., Krakowiak, M., Jaroszewski, M., Oko, J., & Sujecki, S. (2021). Automatic CVSS-based vulnerability prioritization and response with context information. 2021 International Conference on Software, Telecommunications and Computer Networks. https://doi.org/10.23919/SOFTCOM52868.2021.9559094
[16] Cybersecurity and Infrastructure Security Agency. (2022). CISA Stakeholder-Specific Vulnerability Categorization guide. https://www.cisa.gov/ssvc
[17] Spring, J., Hatleback, E., Householder, A., Manion, A., & Shick, D. (2021). Time to change the CVSS? IEEE Security & Privacy, 19(2), 74–78. https://doi.org/10.1109/MSEC.2020.3044475
[18] Jacobs, J., Romanosky, S., Edwards, B., Adjerid, I., & Roytman, M. (2021). Exploit Prediction Scoring System (EPSS). Digital Threats: Research and Practice, 2(3), 1–17. https://doi.org/10.1145/3436242
[19] Cybersecurity and Infrastructure Security Agency. (n.d.). Known Exploited Vulnerabilities catalog. https://www.cisa.gov/known-exploited-vulnerabilities-catalog
[20] Souppaya, M., & Scarfone, K. (2022). Guide to enterprise patch management planning: Preventive maintenance for technology (NIST Special Publication 800-40 Rev. 4). National Institute of Standards and Technology. https://doi.org/10.6028/NIST.SP.800-40r4