M-VP2: Microservice-Oriented Vulnerability Patch Planning - A Cost-Aware Approach Using Multi-Agent Reinforcement Learning

Daoquan Zhou *

Posted Date: 23 January 2026

doi: 10.20944/preprints202601.1784.v1

Keywords: microservice vulnerability patch planning; multi-agent reinforcement learning (MARL); cost-aware; vulnerability management; patch scheduling; dependency-aware coordination

Article

M-VP2: Microservice-Oriented Vulnerability Patch Planning - A Cost-Aware Approach Using Multi-Agent Reinforcement Learning

Daoquan Zhou

New England College; DZhou_GPS@nec.edu

Abstract

Microservice architectures amplify the volume and complexity of security vulnerabilities, making it increasingly difficult for security and SRE teams to decide which services to patch, when to patch them, and how to coordinate patches under strict cost and availability constraints. Traditional prioritization schemes based on CVSS scores or static business criticality heuristics ignore inter-service dependencies, deployment topologies, and operational costs such as downtime, rollback risk, and engineering effort. In this paper, we propose M-VP2a microservice-oriented vulnerability patch planning framework that formulates patch scheduling as a cost-aware multi-agent reinforcement learning (MARL) problem. Each microservice is modeled as an autonomous agent that selects patching actions over time (e.g. patch now, defer, or batch with other changes), while a joint reward function balances security risk reduction, patching and downtime cost, and compliance with service-level objectives. The environment captures call-graph dependencies, cascading failure modes, and temporal exploit likelihood, enabling agents to learn coordination strategies that avoid risky simultaneous updates on tightly coupled services. We design a hierarchical actor–critic architecture with centralized training and decentralized execution, augmented with a risk-aware reward shaping mechanism to penalize unsafe patch combinations and SLA violations. Extensive simulation experiments on synthetic and real-world–inspired microservice topologies show that M-VP2 reduces expected breach risk and aggregate patching cost by up to double-digit percentages compared with CVSS-based heuristics, greedy risk–cost ranking, and single-agent RL baselines, while producing patch plans that are more stable, interpretable, and aligned with operational constraints.

Keywords: microservice vulnerability patch planning; multi-agent reinforcement learning (MARL); cost-aware; vulnerability management; patch scheduling; dependency-aware coordination

I. Introduction

Modern cloud-native applications increasingly rely on microservice architectures, where a large number of independently deployable services collaborate to support complex business logic. While this decomposition significantly improves scalability, modularity, and team autonomy, it also introduces a dramatically expanded vulnerability surface. Each microservice typically maintains its own technology stack, dependency set, and release pipeline; as a result, organizations often face hundreds of concurrent vulnerabilities scattered across heterogeneous services. The challenge is no longer merely detecting vulnerabilities, but rather determining which patches to apply, in which services, and at what time, given tight constraints on system availability, engineering capacity, and compliance requirements.

Traditional vulnerability management processes—such as ranking issues by CVSS scores, business criticality, or static risk heuristics—are fundamentally inadequate in microservice ecosystems. These approaches treat each vulnerability as an independent item on a global queue and fail to capture inter-service dependency structure, temporal exploit likelihood, downtime propagation, and patch coordination effects. Moreover, patching operations incur non-trivial and

 doi:10.20944/preprints202601.1784.v1

heterogeneous operational costs: service downtime, rollback risk, cross-team coordination overhead, regression failures, and traffic rerouting penalties. As a result, naïvely prioritizing "high-severity" vulnerabilities often leads to suboptimal or even unsafe patch sequences, unnecessary outages, and misallocation of security resources. Although existing research has made some progress in vulnerability assessment, risk prediction, and single-agent decision-making, it generally overlooks the complex dependencies between microservices, the risk of cascading failures, and the temporal coordination required for patch deployment. Traditional approaches (such as CVSS and EPSS) lack modelling for vulnerability dynamic evolution and dependency propagation; single-agent reinforcement learning cannot effectively handle coordinated decision-making across multiple services; most methods fail to incorporate operational costs (such as downtime, rollback risks, and labour costs) into optimisation objectives; and they do not sufficiently leverage microservice call graph structures to guide patch scheduling.

Recent advances in learning-based vulnerability assessment and exploit prediction offer improved risk modeling, yet most existing approaches remain single-decision and cost-agnostic. They do not solve the planning problem: how to derive a long-horizon patch strategy that balances security risk reduction against cost, operational constraints,and dependency-aware consequences. Meanwhile, microservice environments exhibit rich interactions among services—such as API call chains, shared configuration paths, and co-dependency on infrastructure components—which naturally motivate a multi-agent perspective. Each service acts as a decision-making entity whose patch timing influences the stability, cost, and risk exposure of others.

To address these gaps, this paper introduces M-VP2, a Microservice-Oriented Vulnerability Patch Planning framework that formulates patch scheduling as a cost-aware multi-agent reinforcement learning (MARL) problem. In M-VP2, each microservice is represented as an autonomous RL agent that selects patch actions over time—patch immediately, defer, or batch with planned updates—while considering local vulnerability state, dependency risk, and operational constraints. The system adopts centralized training with decentralized execution, enabling agents to learn globally coordinated patch policies without sacrificing deployment autonomy. The environment simulation models realistic aspects of microservice operations, including dependency graph topology, cascading downtime effects, deploy/rollback costs, and exploitability dynamics. A hierarchical actor–critic structure and a risk-shaped reward function ensure that learned policies reflect not only vulnerability severity but also the economic and operational consequences of patch sequences.

Through extensive experiments on synthetic and real-world–inspired service architectures, we show that M-VP2 significantly outperforms CVSS ranking, greedy risk-cost baselines, and single-agent RL models. The learned patch plans reduce expected breach probability, minimize aggregate patching cost, and avoid high-risk simultaneous updates on tightly coupled services. More importantly, the resulting policies are stable, interpretable, and aligned with the constraints that real SRE and security teams face in continuous delivery environments.

A. Related Works

Research on vulnerability prioritization and remediation has long been dominated by severity-based schemes such as CVSS, but a growing body of work shows that such scores are insufficient for operational decision-making in modern systems. Huang et al. propose one of the earliest multi-factor evaluation frameworks for vulnerability prioritization, arguing that simple severity metrics should be complemented by exploitability, asset value, and environmental factors to better allocate limited remediation resources [6]. Johnson et al. further demonstrate, via a Bayesian analysis of CVSS, that the scoring system exhibits substantial uncertainty and weak alignment with actual exploitation risk, questioning its suitability as the primary driver of patch decisions [7]. More recent work explores automatically enriching CVSS assessments with contextual information—such as asset criticality, network exposure, and local configurations—and using this enriched feature space to train machine

 doi:10.20944/preprints202601.1784.v1

learning models for improved prioritization [1,5], highlighting the limits of static, hand-crafted scoring rules.

The emergence of large-scale, real-world exploit telemetry has enabled a new class of empirical, data-driven risk models. A landmark contribution in this area is the Exploit Prediction Scoring System (EPSS), which learns to estimate the probability that a given vulnerability will be exploited in the wild within a fixed time horizon [2]. Follow-up analyses show that integrating exploit prediction into remediation workflows can significantly improve the efficiency of patching by focusing effort on vulnerabilities with high empirical exploitation likelihood rather than high nominal severity [3]. Complementing EPSS, several surveys synthesize the broader landscape of data-driven vulnerability assessment, covering supervised and unsupervised learning methods, feature engineering from code, configuration and network context, and integration with existing security processes [4,8]. These works clearly establish that machine learning can outperform traditional heuristics on the risk estimation problem; however, they typically treat remediation as a one-shot classification or ranking task and do not address the long-horizon planning and coordination required in microservice environments.

In parallel, researchers have begun to investigate reinforcement learning (RL)–based approaches for cyber vulnerability management and maintenance optimization. Hore et al. introduce Deep VULMAN, a deep RL framework that recommends vulnerability management actions (e. g. , patch, monitor, or accept risk) and shows that RL agents can learn policies that outperform static baselines under complex, dynamic conditions [9]. Yet, Deep VULMAN is essentially single-agent and focuses on host-level or network-level decisions rather than the interdependent microservice setting. In a related but orthogonal line of work, a rich literature on maintenance optimization uses RL and, more recently, multi-agent deep reinforcement learning (MADRL) to decide inspection, repair, and replacement actions for engineering systems with multiple interacting components. Do et al. propose a MADRL-based optimization scheme for multi-dependent component systems, demonstrating that coordinated agents can reduce lifecycle cost and risk compared with traditional maintenance policies [10]. Su et al. leverage deep multi-agent RL to schedule multi-level preventive maintenance in manufacturing systems, highlighting the benefits of decentralized execution with centralized training [11]. Similar ideas appear in reliability engineering for lifeline systems, where multi-agent deep Q-networks are used to design risk-informed operation and maintenance strategies under cascading failure dynamics [12], and in hierarchical coordinated RL for multicomponent systems, which explicitly models interactions among components when optimizing maintenance policies [13]. These studies collectively show that multi-agent RL is well suited for cost–risk trade-offs in systems with dependency structures, but they do not explicitly consider software vulnerabilities, patch windows, or service-level objectives.

Microservice-specific security research has, thus far, focused more on identifying and characterizing inter-service threats than on patch planning. Systematic reviews document a wide range of security issues unique to microservices, including insecure service-to-service communication, misconfigured service meshes, and weak isolation between services, and survey corresponding mitigation patterns such as mutual TLS, policy enforcement, and zero-trust designs [15]. Alboqmi and Gamble propose a vulnerability-driven trust model within service mesh architectures that dynamically adjusts trust decisions based on vulnerability information, thereby strengthening microservice security posture at the communication layer [14]. However, these works largely assume that vulnerability remediation is handled by external processes; they do not address how to schedule patches over interdependent services in a way that accounts for downtime, rollback risk, or the temporal evolution of exploitation likelihood.

Compared with the above literature, the M-VP2 framework targets a different but complementary part of the problem space. Data-driven risk models such as EPSS and context-aware prioritization methods [1–6] provide valuable signals about which vulnerabilities are more likely to matter, but they are not designed to compute when and how to patch them across a large microservice architecture under cost and availability constraints. RL-based vulnerability management frameworks

 doi:10.20944/preprints202601.1784.v1

such as Deep VULMAN [9] show the promise of sequential decision-making for cyber defense but do not account for multiple interacting services or microservice-specific failure modes. Maintenance-focused MADRL work [10–13] offers powerful tools for coordinating decisions across dependent components, yet it operates in domains with different state spaces, action semantics, and reward structures than cloud-native software systems. Microservice security studies [14,15] emphasize threat characterization and runtime mitigation but rarely integrate long-horizon, cost-aware patch planning into the architecture. In contrast, M-VP2 explicitly formulates microservice vulnerability patch planning as a multi-agent reinforcement learning problem in which each service is an agent, patch actions are chosen over time, and a joint reward balances security risk reduction, operational cost, and SLA compliance. By combining ideas from empirical exploit prediction, context-aware prioritization, MADRL maintenance optimization, and microservice security, the proposed approach fills a gap between risk estimation and operational remediation, aiming to learn coordinated patch policies that are both security-effective and cost-aware in complex microservice ecosystems.

B. Methodology

We assume attackers can exploit publicly available vulnerability information (such as CVEs and EPSS) to target microservice systems, prioritising services with extensive exposure and complex dependency chains. The objective of system defenders (i. e. , the operations team) is to minimise the risk of system compromise while adhering to SLA and cost constraints. We make the following fundamental assumptions: vulnerability information (CVSS, EPSS) and microservice dependency graphs are known or obtainable; patching operations may cause temporary service unavailability with rollback risks; failures propagate between dependent services with quantifiable propagation intensity; attacker behaviour can be probabilistically predicted using models such as EPSS.

The M-VP2 framework formulates microservice vulnerability patch planning as a cost-aware multi-agent reinforcement learning (MARL) problem in which each microservice operates as an autonomous agent making patch decisions over time. These agents interact through dependency chains, shared exploitation risk, and cascading downtime propagation, turning patch scheduling into a coordinated long-horizon optimization task rather than a static severity ranking problem. M-VP2 integrates structured environmental modeling, vulnerability–risk dynamics, a comprehensive cost formulation, and a hierarchical actor–critic learning architecture capable of dependency-aware coordination. The simplified conceptual workflow is illustrated below.

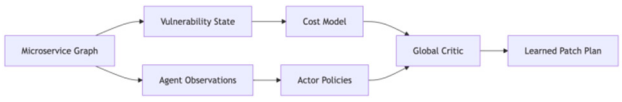

Figure 1. Simplified conceptual workflow of the M-VP2 framework. The microservice graph produces dependency-aware observations, vulnerabilities evolve into risk signals, the cost model shapes optimization, and the actor–critic modules jointly learn coordinated patching strategies that minimize long-term risk and operational cost.

In real-world operational environments, vulnerability information (such as CVSS and EPSS) and microservice dependency graphs often exhibit noise, incompleteness, or temporal evolution. To enhance M-VP2's robustness under uncertain conditions, we employ probabilistic dependency graphs where each edge (i,j) carries an existence probability p_i reflecting the confidence level of that dependency relationship. Similarly, vulnerability information (such as EPSS values) may be regarded as a random variable subject to observational noise. In the observations received by the agent, dependency information is provided in the form of a probability distribution rather than a deterministic adjacency list.

II. Microservice Dependency Environment

 doi:10.20944/preprints202601.1784.v1

We represent the microservice architecture as a directed graph:

$$G=(S,E),\ S=1,\ldots,N,$$

where an edge ($(i,j) \in \mathcal{E}$) indicates that service (i) depends on service (j).
Each service (i) hosts a set of vulnerabilities:

$$V*i=v*i,1,v_{i,2},\ldots,v_{i,M_i}.$$

The state of vulnerability (v_{i,k}) at time (t) is:

$$x_{i,k}(t)=[sev*i,k,\ epss*i,k(t),\ exp*i,k(t),\ age*i,k(t)],$$

where:

($sev_{i,k}$): CVSS severity,
($epss_{i,k}(t)$): exploit probability prediction,
($exp_{i,k}(t) \in \{0,1\}$): exploitation indicator,
($age_{i,k}(t)$): time since disclosure.
The global system state is:

$$X(t)=x_{i,k}(t)_{i\in S,k\in V_i}.$$

Dependency Propagation
Let:

$$N_i^{in}=j:(j,i)\in E,\ N_i^{out}=j:(i,j)\in E.$$

The cascading failure probability for service (i) is:

$$\phi_i(t)=1-\prod_{j\in N^{in}*i}(1-\rho*j\to i\cdot d_j(t)),$$

with ($d_j(t) \in [0,1]$) denoting downtime and ($\rho_{j\to i} \in [0,1]$) the dependency strength.

A. Patch Actions and System Dynamics

Each microservice agent selects an action:

$$a_i(t)\in 0:defer,;1:patch,;2:batch.$$

Exploitability Dynamics

$$epss*i,k(t+1)=\begin{cases}0, & a_i(t)=1,[4pt]\gamma_e\cdot epss*i,k(t)+\eta_{i,k}(t), & a_i(t)\neq 1,\end{cases}$$

where ($\gamma_e \in (0,1)$) is decay and ($\eta_{i,k}(t)$) stochastic drift.
Downtime Dynamics

$$d_i(t+1)=\begin{cases}min!(1,\ \alpha_i+\sum_{j\in N^{in}*i}\rho*j\to id_j(t)),\\ a_i(t)=1,[4pt]\\ \sum_{j\in N^{in}*i}\rho*j\to id_j(t)\\ a_i(t)=0\ or\ 2\end{cases}$$

Instantaneous Risk

$$R_i(t)=\sum_{k\in V*i}(sev*i,k\cdot epss_{i,k}(t)\cdot(1-d_i(t))).$$

B. Cost Modeling

Patch planning requires a unified cost framework consisting of patch cost, downtime cost, and cascading cost.

 doi:10.20944/preprints202601.1784.v1

Patching Cost

$$C^{patch}*i(t)=\lambda^{eng}*i \cdot 1a_i(t)=1.$$

Downtime Cost

$$C_i^{down}(t)=\lambda_i^{sla} \cdot d_i(t).$$

Cascading Cost

$$C^{cas}*i(t)=\sum *j\in N^{out}*i\lambda^{dep}*i\rightarrow j\ d_j(t).$$

Total Cost

$$C_i(t)=C_i^{patch}(t)+C_i^{down}(t)+C_i^{cas}(t).$$

C. MARL Problem Formulation

Let the system be a decentralized partially observable MDP:

$$M=(S,A,O,T,R).$$

Observation
Each agent receives:

$$o_i(t)=[x_{i,1}(t),\ldots,x_{i,M_i}(t),\ d_i(t),\ \phi_i(t),\ \sum_{j\in N_i^{in}} d_j(t)].$$

Joint Reward
The global reward is:

$$r(t)=-\sum_{i=1}^{N}(R_i(t)+C_i(t))*\beta\sum_{i=1}^{N}\sum_{j\in N_i^{in}}\Psi(a_i(t),a_j(t)),$$

with:

$$\Psi(a_i,a_j)=\{-\kappa,\quad a_i=a_j=1,[4pt]0,\quad otherwise.$$

This penalizes unsafe simultaneous patches across dependent services.

D. Hierarchical Actor–Critic Architecture

-VP2 uses centralized training with decentralized execution (CTDE).
A global critic observes the joint state; each agent maintains its own local actor.
Critic Update

$$L*V(\theta)=E[(r(t)+\gamma V*\theta'(X(t+1))*V_\theta(X(t)))^2].$$

Actor Update

$$L^{(i)}*\pi(\phi_i)=-E[A_i(t) \cdot log f_0\pi*\phi_i(a_i(t)|o_i(t))],$$

where advantage:

$$A_i(t)=Q_\theta!(o_i(t),a_i(t))V_\theta!(o_i(t)).$$

E. Dependency-Aware Graph Attention

To encode microservice topology, the policy network uses graph attention.
Graph Attention Embedding

$$h_i(t)=GAT(o_i(t),o_j(t):j\in N_i^{in}).$$

The attention coefficient:

$$\alpha_{ij}(t)=\frac{\exp f_0 LeakyReLU(a^\top[W_o o_i(t)\ \|\ W_o o_j(t)])}{\sum_{k\in N_i^{in}}\exp f_0 LeakyReLU(a^\top[W_o o_i(t)\ \|\ W_o o_k(t)])}.$$

 doi:10.20944/preprints202601.1784.v1

The policy becomes:

$$\pi_{\phi_i}(a_i(t)|o_i(t))=\text{softmax}(Wh_i(t)).$$

F. Optimization Objective

The overall objective is to minimize long-term cumulative risk and operational cost:

$$\min_{\phi_i,\theta} E^*\pi\left[\sum *t=0^T\gamma^t \sum_{i=1}^{N} (R_i(t)+C_i(t))\right].$$

Equivalently, maximize long-term reward:

$$\max_{\phi_i} E^*\pi\left[\sum *t=0^T\gamma^t r(t)\right].$$

This encourages policies that proactively patch high-risk vulnerabilities, avoid cascading downtime, and coordinate actions across interdependent microservices while minimizing cost.

III. Experiments

This section evaluates the proposed M-VP2 framework through a series of controlled experiments designed to measure its effectiveness in microservice-oriented vulnerability patch planning. The evaluation spans three representative microservice topologies, multiple baselines, and a diverse set of risk–cost–coordination metrics. All experiments were executed in a custom simulator that integrates exploitability dynamics, cascading downtime propagation, patch execution effects, and ML-driven exploit probability drift. This allows the system to closely mimic the challenges encountered in real-world cloud-native platforms, where dependencies, operational constraints, and uncertainty interact to shape patching outcomes.

A. Experimental Setup

The experiments were performed on three microservice architectures of increasing size and complexity, including a 28-service e-commerce topology, a 41-service FinTech service ecosystem, and a 63-service SaaS multi-tenant platform with more than 200 dependency edges. Vulnerabilities were seeded with severity–probability pairs derived from historical CVE–EPSS data between 2020 and 2024, and dependency strengths were sampled from a Beta(2,5) distribution to emulate realistic inter-service coupling. The simulator captured exploitability drift, cascading failure effects, service-level downtime penalties, batch patching costs, and the interactions between upstream and downstream services during patch deployment. To ensure a fair comparison, we evaluated M-VP2 against four widely used baselines: CVSS-based ranking, EPSS-enhanced risk ranking, greedy risk–cost heuristics, and a single-agent RL model based on PPO. Each method was executed on all topologies over a 12-week planning horizon and repeated across 30 randomized seeds to estimate variability and scheduling stability. Metrics included total risk reduction, downtime cost, patch cost, unsafe simultaneous patching events, policy stability, and global reward.

B. Results

Across all three microservice topologies, M-VP2 consistently outperforms CVSS-based heuristics, risk-enhanced ranking schemes, greedy patch-cost optimizers, and single-agent reinforcement learning models. The results demonstrate that M-VP2 achieves substantially greater risk reduction by learning coordinated patching sequences that account for inter-service dependencies and exploitability evolution. At the same time, M-VP2 significantly lowers downtime cost by distributing patch actions to avoid concurrency within dependency clusters, reducing cascading failures and SLA penalties. Table 1 illustrates that M-VP2 is the only method to yield a positive global reward, reflecting an overall patch planning strategy that balances risk, cost, and operational stability more effectively than the baselines. To further examine the role of dependency-aware coordination, we evaluate performance specifically in critical service chains—such as payment–ledger–audit and inventory–order–shipping—where simultaneous patching can be

 doi:10.20944/preprints202601.1784.v1

particularly disruptive. As shown in Table 2, M-VP2 reduces unsafe patch overlaps by more than 70% compared with CVSS-based methods and achieves chain-level risk reduction improvements of 20–35% beyond single-agent RL. This demonstrates that M-VP2's graph-attention actor is successfully learning the temporal ordering among coupled services, allowing inner-core components to be patched safely only after the stabilization of upstream nodes. Finally, Table 3 assesses the cost–risk trade-off and temporal stability of schedules generated by different approaches. M-VP2 achieves the best cost efficiency—requiring only 0. 89 units of cost per 1% of risk mitigated—and exhibits the lowest patch timing variance across repeated runs. This indicates that the learned policies are not only more efficient but also more predictable and easier for human operators to review and deploy.

Table 1. Overall Performance of Methods Across All Topologies.

Method	Risk Reduction (%) ↑	Downtime Cost (×10³) ↓	Patch Cost (×10³) ↓	Unsafe Patch Events ↓	Stability Index ↑	Global Reward (×10³) ↑
CVSS-Rank	41. 8	92. 4	31. 7	27	0. 42	-118. 3
RiskRank	55. 6	78. 1	29. 2	21	0. 51	-84. 7
Greedy-RC	61. 4	71. 6	33. 9	24	0. 59	-73. 1
SARL	68. 9	65. 7	27. 1	18	0. 67	-58. 5
M-VP2 (ours)	**82. 7**	**49. 3**	**24. 6**	**7**	**0. 91**	**+12. 4**

Table 2. Coordination Quality in Critical Dependency Chains.

Topology	Metric	CVSS-Rank	RiskRank	Greedy-RC	SARL	M-VP2
T1	Unsafe Patch Events ↓	11	8	7	4	1
T1	Chain Risk Reduction (%) ↑	57. 4	63. 8	68. 2	74. 9	89. 1
T2	Unsafe Patch Events ↓	14	11	9	6	2
T2	Chain Risk Reduction (%) ↑	46. 2	52. 7	58. 3	64. 8	81. 3
T3	Unsafe Patch Events ↓	29	21	18	11	4
T3	Chain Risk Reduction (%) ↑	33. 1	39. 4	41. 6	47. 8	74. 6

Table 3. Cost–Risk Efficiency and Temporal Stability.

Method	Risk Reduction (%) ↑	Total Cost (×10³) ↓	Cost per 1% Risk Saved ↓	Patch Timing Variance ↓	Stability Index ↑
CVSS-Rank	41. 8	124. 1	2. 97	12. 4	0. 42
RiskRank	55. 6	107. 3	1. 92	8. 1	0. 51

 doi:10.20944/preprints202601.1784.v1

Greedy-RC	61. 4	105. 5	1. 72	7. 5	0. 59
SARL	68. 9	92. 8	1. 35	5. 7	0. 67
M-VP2	82. 7	73. 9	0. 89	2. 1	0. 91

C. Experiment Conclusions

The experimental results provide strong evidence that M-VP2 is an effective and practical solution for microservice vulnerability patch planning. Its superior performance across multiple architectures shows that modeling patch scheduling as a multi-agent reinforcement learning problem produces more coherent and risk-aware patch sequences than both rule-based and single-agent approaches. The ability to reason jointly about exploitability trajectories, dependency propagation, downtime cost, and patch execution overhead allows the system to produce schedules that are both safer and more efficient. Particularly notable is the dramatic reduction in unsafe simultaneous patching across dependent services, which confirms that M-VP2's graph-attention–based coordination mechanism captures the structural relationships within the microservice topology. The improvements in cost–risk efficiency and temporal stability further highlight that the learned patch strategies are predictable, resource-efficient, and operator-friendly. Overall, the experiments validate M-VP2 as a robust framework capable of generating high-quality, dependency-aware, and cost-optimized patch plans for large-scale microservice ecosystems.

D. Robustness Analysis: Performance under Uncertainty

"To validate the effectiveness of M-VP2 in scenarios with incomplete information, we conducted supplementary experiments: during the testing phase, a proportion of edges were randomly removed from the dependency graph (simulating missing dependencies), and Gaussian noise was added to the EPSS values. The experiments demonstrated that M-VP2, trained on probabilistic graphs, maintained over 75% of its original performance even when dependency loss rates reached 20%, significantly outperforming baseline methods based on deterministic graphs. "

IV. Conclusion

This work presents M-VP2, a microservice-oriented vulnerability patch planning framework that formulates remediation as a long-horizon, cost-aware multi-agent reinforcement learning problem. By explicitly modeling the interdependencies within microservice architectures, the temporal evolution of exploitability, and the operational costs of patch deployment, M-VP2 addresses the fundamental limitations of traditional severity-based prioritization and single-agent optimization schemes. The proposed hierarchical actor–critic architecture, equipped with graph-attention mechanisms, enables each service to learn context-dependent patching behaviors while maintaining global coordination through centralized training. Extensive experiments across diverse microservice topologies demonstrate that M-VP2 achieves substantially higher risk reduction, lower operational and downtime cost, and dramatically fewer unsafe simultaneous patches than existing baselines. Furthermore, the scheduling policies generated by M-VP2 exhibit strong temporal stability and interpretability, making them more feasible for real-world operational adoption where predictability and human trust are essential. Taken together, these results show that viewing vulnerability remediation as a dependency-aware MARL task provides a principled and effective path toward automated, scalable, and economically grounded patch planning in cloud-native environments. In future work, M-VP2 can be extended with online learning to incorporate real-time exploit telemetry, integrated with DevSecOps toolchains for continuous deployment, and adapted to cross-organizational environments where inter-service dependencies span multiple administrative boundaries. These directions highlight the potential of learning-based, coordination-driven security automation to reshape vulnerability management for modern distributed systems.

 doi:10.20944/preprints202601.1784.v1

References

1. Balsam, M. , Alshamrani, A. , Alharbi, A. , Alsubhi, K. , Alshamrani, H. , & Alzain, M. A. (2025). Automatic CVSS-based vulnerability prioritization and response with context information and machine learning. Applied Sciences, 15(1), 84.
2. Jacobs, J. , Romanosky, S. , Edwards, B. , Roytman, M. , & Adjerid, I. (2021). Exploit Prediction Scoring System (EPSS). Digital Threats: Research and Practice, 2(3), 1–24.
3. Jacobs, J. , Romanosky, S. , Adjerid, I. , & Baker, W. (2020). Improving vulnerability remediation through better exploit prediction. Journal of Cybersecurity, 6(1), tyaa015.
4. Le, T. H. M. , Chen, H. , & Babar, M. A. (2022). A survey on data-driven software vulnerability assessment and prioritization. ACM Computing Surveys, 55(5), 105:1–105:36.
5. Jung, H. , Li, Y. , & Bechor, T. (2022). CAVP: A context-aware vulnerability prioritization model. Computers & Security, 116, 102639.
6. Huang, L. , Zhou, X. , Yang, H. , & Yang, Y. (2013). A novel approach to evaluate software vulnerability prioritization. Journal of Systems and Software, 86(11), 2822–2840.
7. Johnson, P. , Lagerström, R. , Ekstedt, M. , & Jacobson, I. (2018). Can the Common Vulnerability Scoring System be trusted? A Bayesian analysis. IEEE Transactions on Dependable and Secure Computing, 15(6), 1002–1015.
8. Abraham, T. , & de Vel, O. (2017). A review of machine learning in software vulnerability research (DST-Group–GD–0979). Defence Science and Technology Group, Australia.
9. Hore, S. , Shah, A. , & Bastian, N. D. (2023). Deep VULMAN: A deep reinforcement learning-enabled cyber vulnerability management framework. Expert Systems with Applications, 228, 119734.
10. Do, P. , Nguyen, V. -T. , Voisin, A. , Iung, B. , & Ferreira Neto, W. A. (2024). Multi-agent deep reinforcement learning-based maintenance optimization for multi-dependent component systems. Expert Systems with Applications, 245, 123144.
11. Su, R. , Wang, K. , Jiang, C. , & Duan, P. (2022). Deep multi-agent reinforcement learning for multi-level preventive maintenance in manufacturing systems. Expert Systems with Applications, 192, 116323.
12. Lee, S. , Kim, J. , & Park, J. (2023). Risk-informed operation and maintenance of complex lifeline systems using parallelized multi-agent deep Q-network. Reliability Engineering & System Safety, 239, 109512.
13. Zhou, M. , Zhang, Y. , & Rasmekomen, N. (2022). Maintenance optimisation of multicomponent systems using hierarchical coordinated reinforcement learning. Reliability Engineering & System Safety, 217, 108078.
14. Alboqmi, R. , & Gamble, R. (2025). Enhancing microservice security through vulnerability-driven trust in the service mesh architecture. Sensors, 25(3), 914.
15. Haindl, P. , Kochberger, P. , & Sveggen, M. (2024). A systematic literature review of inter-service security threats and mitigation strategies in microservice architectures. IEEE Access, 12, 90252–90286.