December 2024

Enterprise Security Patch Management with Deep Reinforcement Learning

Qian Jia
Nanjing University, qianjia@smail.nju.edu.cn

Xinxue Qu
University of Notre Dame, xqu2@nd.edu

Zhengrui Jiang
Nanjing University, zjiang@nju.edu.cn

Chengjun Wang
Nanjing University, wangchengjun@smail.nju.edu.cn

Enterprise Security Patch Management with Deep Reinforcement Learning

Completed Research Paper

Qian Jia
Nanjing University
22 Hankou Road, Nanjing, China
qianjia@smail.nju.edu.cn

Xinxue Qu
University of Notre Dame
354 Mendoza, Notre Dame, IN
xqu2@nd.edu

Zhengrui Jiang
CUHK-Shenzhen
2001 Longxiang, Shenzhen, China
zjiang@cuhk.edu.cn

Chengjun Wang
Nanjing University
22 Hankou Road, Nanjing, China
wangchengjun@smail.nju.edu.cn

Abstract

Patching in a timely manner has proven to be one of the most effective ways to protect enterprise information systems from cyberattacks. However, as patching operations are not cost-free, enterprises typically delay patch deployments. Balancing operational expenses and system security risks to determine the optimal timing of patching remains an ongoing challenge. This study addresses the patching decision problem by proposing a novel deep reinforcement learning-based approach. Specifically, we model the patching problem as a Markov decision process with a thorough consideration of various costs, dynamics, and uncertainties. To avoid the curse of dimensionality and obtain an effective patching policy, we develop a novel reinforcement learning method called Action-Decomposed Proximal Policy Optimization (ADPPO). Experimental results indicate that the proposed approach significantly outperforms benchmarks. This study contributes to both the cybersecurity management and the reinforcement learning communities.

Keywords: security patch management, vulnerability, reinforcement learning, Markov Decision Process, enterprise information system

Introduction

With the rapid expansion of information assets in recent years, enterprises are facing a crucial challenge in information system security management. According to the Cost of a Data Breach Report 2023, the worldwide average total cost of a data breach has reached US$4.45 million in 2023, reflecting a 15% increase over the past three years.[1] The primary cause of security breaches is typically attributed to unpatched vulnerabilities. A study conducted by ServiceNow shows that 60% of information systems were breached due to known vulnerabilities that remain unpatched, despite the availability of released patches.[2] For example, the infamous ransomware "WannaCry" rapidly spread in 2017, infecting tens of thousands device across over 150 countries. Hackers exploited a critical Windows SMB vulnerability to gain access to enterprise servers, despite Microsoft having released a security update several months earlier.[3]

Deploying security patches is the most effective way to remediate known vulnerabilities (Souppaya & Scarfone, 2013). In theory, a system manager should apply a security patch as soon as possible after its release to minimize security risks. However, it is common for enterprises to postpone patch deployment (Cavusoglu et al., 2008; Dissanayake et al., 2022). Only about 50% of industry devices are patched within

[1] https://www.ibm.com/reports/data-breach
[2] https://www.servicenow.com/lpayr/ponemon-vulnerability-survey.html
[3] https://www.cisa.gov/news-events/alerts/2017/05/12/indicators-associated-wannacry-ransomware

60 days of vulnerability disclosure (Wang et al., 2017). A recent report reveals that approximately one-third of enterprises are using unpatched or outdated information systems. [4] The main reasons for this phenomenon are twofold. First, remediating vulnerabilities in a timely manner requires frequent patching operations, leading to a series of operational costs, such as those related to human and device resources, downtime, system rebooting, business disruption, and so on (Dissanayake et al., 2022; Jia et al., 2021). Second, deploying a patch prematurely, particularly if it is not thoroughly tested, can result in system errors or crashes, leading to further costs associated with system recovery (Beattie et al., 2002; Jenkins et al., 2024). As a result, the patching issue goes beyond mere security considerations; it poses a management challenge with a series of associated operational costs to be considered.

How can we strike a balance between the costs of patching operations and the security risks associated with unpatched vulnerabilities to minimize the total system cost? To address this problem, several studies have been conducted. For example, Beattie et al. (2002) examine the timing of applying security patches, considering the security costs associated with flaws in unreliable patches and vulnerabilities exploited by hackers. They reveal that, after the software vendor releases the patch, the security risk to the system increases over time as more hackers gain access to vulnerability information. Conversely, as more users adopt the patch and confirm its stability, the risk associated with a flawed patch decreases over time. Cavusoglu et al. (2008) develop a game-theoretic model to study vendors' patch-release policies and users' patch-update policies. Their findings demonstrate that the total social cost reaches its minimum when the vendor releases the patch and the user adopts it synchronously. They also reveal how users and vendors can coordinate with each other to achieve a lower social cost. Utilizing a comprehensive analytical framework, Dey et al. (2015) find that either the time-based policy or the total-control policy can be the optimal one in specific scenarios. Under the time-based policy, the system manager deploys patches periodically, while under the total-control policy, patching operations are performed only when the cumulative severity of patches within the system exceeds a predetermined optimal threshold. By merging these two policies, they introduce a hybrid policy, with experimental evidence indicating its superiority over other approaches.

While the above works typically formulate the patching issue as a static decision problem, considering the problem's dynamic nature, Jia et al. (2021) formulate it as a sequential decision-making problem under a Markov decision process (MDP) framework. They propose a dynamic patching policy (DPP) and derive the optimal policy under both finite and infinite time horizons. Experimental results show that DPP is superior to other approaches. Their findings suggest that dynamic policies can significantly outperform static ones. However, due to the curse of dimensionality, it becomes practically impossible to derive the optimal dynamic policy when dealing with a large number of patches and considering more variables.

In recent years, deep reinforcement learning methods like deep Q-network (DQN), deep deterministic policy gradients (DDPG), asynchronous advantage actor-critic (A3C), trust region policy optimization (TRPO), and proximal policy optimization (PPO) have been widely utilized to tackle the curse of dimensionality in large-scale sequential decision problems (Lillicrap et al., 2015; Mnih et al., 2016; Mnih et al., 2015; Schulman et al., 2015; Schulman et al., 2017; Wang et al., 2020). For example, Oroojlooyjadid et al. (2022) optimize the inventory decision with a DQN-based method. Yang et al. (2022) introduce dynamic pricing and information disclosure strategies based on the PPO algorithm. Liu (2023) derives dynamic coupon targeting strategies using a batch deep reinforcement learning approach. Moreover, a recent study related to ours presents a PPO-based framework (Deep VULMAN) for prioritizing and selecting vulnerabilities to mitigate and proves the proposed PPO-based method is superior to benchmarks (Hore et al., 2023).

Drawing inspiration from prior research, our study introduces a novel approach based on deep reinforcement learning to tackle the patching decision problem. Specifically, we formulate the patching problem as a MDP problem with a thorough consideration of various costs. To avoid the curse of dimensionality inherent in the MDP problem and derive the patching policy effectively, we introduce a new deep reinforcement learning method termed *Action-Decomposed Proximal Policy Optimization (ADPPO)*. Experimental results demonstrate that the proposed ADPPO outperforms benchmarks significantly. Moreover, in contrast to existing research, our model incorporates a wider range of operational costs and real-world uncertainties, including patching costs, patching failure costs, and the dynamics of patch

[4] https://www.kaspersky.com/blog/it-security-economics-2020-part-2/

arrivals, severity, and patching failure risks. The comprehensive model renders our proposed method more practical and applicable for enterprises.

Our study makes the following contributions. First, this design science research offers a practical and innovative tool for enterprises to enhance their security patch management processes. Second, we formulate the patching problem with a thorough consideration of various costs, dynamics, and uncertainties. Our findings illustrate the importance of incorporating these factors to minimize overall enterprise costs. Effectively managing patches from an economic standpoint proves to be more effective than solely focusing on security engineering perspectives. Third, we contribute to the field of reinforcement learning by introducing a novel algorithm (ADPPO). The proposed approach is also applicable to other similar optimal control scenarios, such as inventory management and financial trading. Our study has significant implications for practice as well. Due to the advantages of the proposed method, enterprises can implement it across a wide range of security patch management scenarios and utilize it to enhance automation in security patch management. The method is able to adjust patching policies dynamically by interacting with and learning from the environment in real-time, effectively reducing enterprise costs.

Related Work

Deploying security patches in a timely manner is critical to protecting enterprise information systems from cyberattacks. However, most enterprises delay their patching operations and use outdated software (Dissanayake et al., 2022; Wang et al., 2017) while security experts suggest eliminating identified vulnerabilities as soon as possible (Reeder et al., 2017). Several empirical studies have uncovered the underlying reasons for this phenomenon. A longitudinal study (Wang et al., 2017), examining the impact of vulnerability disclosures on users' patching behaviors, finds that only about 50% of industry devices are patched within 60 days of vulnerability disclosure. The authors show that users' patching behaviors can be forecasted well by a variation of the Bass model. Other factors that may lead software users to delay the patching or update operations include the system manager's personal experience, individual risk preference, and the opportunity cost of a patching operation (Rajivan et al., 2020). Furthermore, a study conducted based on a longitudinal dataset from the healthcare sector reveals that the delay in coordination among humans is the key reason for patching delays (Dissanayake et al., 2022). The study also presents several other technology-related, people-related, and organization-related reasons, such as complexity of patches, limitations of available tools, failures due to poor planning and execution, organizations' capacity limitations and service-availability restrictions, and so on. The results of a recent survey show that new issues introduced by patches and the size of the organization can influence patching behaviors as well (Jenkins et al., 2024). While the studies mentioned above highlight numerous factors influencing users' patching behaviors from different perspectives, another recent study indicates that no single factor outweighs the others in importance (Bondar et al., 2023). The findings suggest that system managers are equally attentive to these factors when they make patching decisions. Therefore, it is essential to carefully balance the costs and benefits of various factors in the process of security patch management.

One of the critical challenges in security patch management is the tradeoff between system security risks and operational costs of enterprises. Prior studies propose several patching policies to address this obstacle and thus optimize system managers' decisions. For example, one early work (Beattie et al., 2002) provides a mathematical model to balance system penetration risks and bad patch risks. The authors consider that, after the software vendor releases the patch, the system security risk grows with time because more and more hackers can obtain the vulnerability information. In contrast, as more and more users adopt this patch and prove the stability of the patch, the bad patch risk decrease with time, which is supported by the data collected by the authors. Addressing this tradeoff, their proposed model finds the optimal time point to deploy the patch. Taking into account both the software vendor's patch-release problem and the user's patch-update problem, a study (Cavusoglu et al., 2008) provides a game-theoretic model to analysis the implications of different patch management strategies: time-driven and event-driven policies. The results show that, under a centralized system, the total social cost is minimized when the vendor releases the patch and the user adopts it synchronously. However, under a decentralized system, the patch may not be released and deployed synchronously. In this case, the authors analyze two coordination mechanisms (i.e., cost sharing and liability) and provides the conditions under which the social cost is minimized. Another study (Dey et al., 2015) compares different patching policies (i.e., one-for-one policy, time-based policy, patch-based policy, total-control policy, and emergency-control policy) under a proposed quantitative framework

to figure out the optimal one for balancing security costs and operational costs. They find that either the total-control policy or the time-based policy can be optimal under specific conditions. Based on this finding, they propose a hybrid policy, which is consistently optimal, by mixing the total-control and time-based policies. Based on the above works, a recent study (Jia et al., 2021) proposes a dynamic patching policy based on the Markov decision process considering future system costs. Experimental results show that the dynamic policy is superior to all prior patching policies due to its flexibility and the stochastic nature of patch arrival processes.

Nonetheless, the above studies consider either a single type of patch or the case in which enterprises can deploy all patches immediately after they decide to patch, regardless of the fact that enterprises have to handle thousands of patches for their information systems with limited resources in practice. Determining whether and which patches to deploy at each decision period is more desirable than solely choosing to patch or not to patch. Therefore, another stream of research on security patch management focuses on the prioritization of patches or vulnerabilities. Severity serves as a crucial metric for prioritizing patches or vulnerabilities. Enterprises can determine severity levels of patches directly based on the industry standard Common Vulnerability Scoring System (Mell et al., 2006), but research shows that a better way to estimate the severity of a vulnerability is combining the exploit probability and the impact on a system if the vulnerability is exploited (Le et al., 2022). Existing studies develop several machine learning-based tools to predict the probabilities and impacts of exploitation activities. For the prediction of exploit likelihood, one of the earliest data-driven methods is based on support vector machines (Bozorgi et al., 2010), which is followed by recent approaches such as Darkembed (Tavabi et al., 2018), FastEmbed (Fang et al., 2020), Exploit Prediction Scoring System (Jacobs et al., 2021), and its latest version (Jacobs et al., 2023). Regarding the impact prediction, there are three common metrics: *Confidentiality*, *Integrity*, and *Availability* (Le et al., 2022). Studies develop prediction models to learn a single metric (Elbaz et al., 2020; Jiang & Atif, 2020) or learn multiple metrics at the same time (Gong et al., 2019; Spanos & Angelis, 2018). With the predicted exploit likelihood and the negative impact of each vulnerability, we can prioritize patches in a comprehensive and effective way (Costa & Tymburibá, 2022). Taking into account future uncertainties, a recent study close to ours proposes a reinforcement learning-based framework (Deep VULMAN) for prioritizing and selecting vulnerabilities to mitigate (Hore et al., 2023). However, the above studies in this stream typically ignore the operational costs considered by the last stream of research, such as patching costs and business disruption costs (Cavusoglu et al., 2008; Dey et al., 2015; Jia et al., 2021). They also overlook the patching failure costs incurred by unsound patches, and the fact that a patch's severity and patching failure risk change over time after the patch is released by the vendor (Beattie et al., 2002). Our study aims to address the gap between these two research streams.

	Various Types of Patches	Patching Failure Cost	Variable Patching Cost	Dynamic Risks	Dynamic Decision	Partial Updating
Cavusoglu et al. (2008)	No	No	Yes	No	No	No
Dey et al. (2015)	Yes	No	No	No	Partially Yes	No
Jia et al. (2021)	Yes	No	No	No	Yes	No
Hore et al. (2023)	Yes	No	Yes	No	Yes	Yes
This work	Yes	Yes	Yes	Yes	Yes	Yes

Table 1. Research Comparison

Note. The dynamic risks mean the security threats and patching failure risks changing over time after the release of patches, and the partial updating refers to allowing system managers to select a part of patches for installation.

Our work is fundamentally different from prior literature in the following ways. First, we focus on optimizing the patching decisions for enterprises to minimize their total costs rather than predicting the severity and exploitation activities of vulnerabilities to prioritize patches. Enterprises can use such kind of prediction methods to estimate parameters in our model. Second, compared to existing research, our model takes into account a wider range of operational costs and uncertainties from the reality at the same time, such as patching costs, patching failure costs, and the dynamics of patch arrivals, severity, and patching failure risks. Third, we formulate the patching problem as a MDP problem and solve it with a novel deep reinforcement learning algorithm–Action-Decomposed Proximal Policy Optimization (ADPPO)–to avoid the curse of dimensionality. Our method is applicable to large enterprise information systems that have thousands of patches to deploy. In a nutshell, our study is more comprehensive and practical for enterprises than previous research and provides a new effective approach for patching decisions. Table 1 summarizes the primary differences between our study and the key literature.

Model

In this section, we formulate the patching decision problem under the MDP framework. We assume the system manager scans the system and make patching decisions periodically (e.g., every week), which is a common practice (Cavusoglu et al., 2008; Jia et al., 2021). Let $t \in \{0,1,\dots,t,\dots,T\}$ denote the decision period under time horizon T and τ denote the time interval of each decision period. The objective is to minimize the total system cost over a time horizon T by optimizing the patching decision for each decision period t. We note that while it is technically feasible to check and update the system in real-time, doing so is not practical or economical for enterprises. Patching operations can disrupt business operations, incurring significant financial losses. Scheduling periodic maintenance during off-peak business hours, such as Saturday nights, for deploying patches proves to be more beneficial. We next define the system state space, action space, state transition function, and cost functions for the patching problem. Table 2 provides a summary of the notation used in this paper.

System State

We define the system state by the number of patches with varying characteristics. Specifically, we consider two key features: *severity level* and *age*. The severity level of a patch refers to the severity of vulnerability within the system that it can mitigate. Security patch severity levels vary as different patches target various types of system vulnerabilities. The age of a patch is determined by the duration since its release, reflecting its freshness within the system. As time progresses, older patches pose a decreased risk of patching failure but an increased risk of exploitation. This is because newly released patches typically undergo refinement over time. As more users install them and provide feedback, their reliability increases, enhancing the likelihood of successful installation (Beattie et al. 2002). Conversely, with patches available for longer periods, cyber attackers gain greater access to corresponding vulnerability information, leading to more opportunities for attacks. Therefore, we define the system state at decision period t as $\boldsymbol{S}_t = \left(\boldsymbol{s}_t^0, \cdots, \boldsymbol{s}_t^f, \cdots, \boldsymbol{s}_t^F\right)$, where $\boldsymbol{s}_t^f = \left(s_t^{1,f}, \cdots, s_t^{l,f}, \cdots, s_t^{L,f}\right)$ and $s_t^{l,f}$ denotes the number of security patches with severity level l and age f, which means patches have been delayed for f decision periods. The system state space $\mathbb{S}$ consists of all possible states across decision periods:

$$\boldsymbol{S}_t = \begin{bmatrix} s_t^{1,0} & \cdots & s_t^{1,f} & \cdots & s_t^{1,F} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ s_t^{l,0} & \cdots & s_t^{l,f} & \cdots & s_t^{l,F} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ s_t^{L,0} & \cdots & s_t^{L,f} & \cdots & s_t^{L,F} \end{bmatrix}, \boldsymbol{S}_t \in \mathbb{S}.$$

Action

At the beginning of each decision period t, the system manager determines the patches to deploy to minimize the total system cost in response to the system state. We define the action at decision epoch t as $\boldsymbol{A}_t = \left(\boldsymbol{a}_t^0, \cdots, \boldsymbol{a}_t^f, \cdots, \boldsymbol{a}_t^F\right)$, where $\boldsymbol{a}_t^f = \left(a_t^{1,f}, \cdots, a_t^{l,f}, \cdots, a_t^{L,f}\right)$, $a_t^{l,f} \in \left[0, s_t^{l,f}\right]$, and $a_t^{l,f}$ denotes the number of patches with severity level l and age f to be deployed at the period t. When $a_t^{l,f} = 0$, there is no patch with

severity level l and age f should be deployed at the current period t. The action space $\mathbb{A}$ consists of all possible actions across decision periods:

$$\boldsymbol{A}_t = \begin{bmatrix} a_t^{1,0} & \cdots & a_t^{1,f} & \cdots & a_t^{1,F} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ a_t^{l,0} & \cdots & a_t^{l,f} & \cdots & a_t^{l,F} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ a_t^{L,0} & \cdots & a_t^{L,f} & \cdots & a_t^{L,F} \end{bmatrix}, \boldsymbol{A}_t \in \mathbb{A}.$$

In the next section, a refined action space is derived to avoid the disadvantages incurred by the above large action space. Based on the refined action space and decision decomposition, we design a novel reinforcement learning algorithm to address the patching problem.

Notation	Description
T	Time horizon.
t	Decision Period.
τ	Time interval of each decision period.
l	Severity level.
f	Age of patch.
$s_t^{l,f}$	Number of patches with severity level l and age f at decision period t.
$a_t^{l,f}$	Number of patches with severity level l and age f to be deployed at decision period t.
$\boldsymbol{S}_t$	System state at decision period t.
$\mathbb{S}$	State space.
$\boldsymbol{A}_t$	Action at decision period t.
$\mathbb{A}$	Action space.
P	State transition probability.
φ_f	Failure rate of patches with age f.
c^{FP}	Fixed patching cost.
c^{VP}	Variable patching cost.
θ_l	Unit variable patching cost of patches with severity level l.
σ_l	Unit patching failure cost of patches with severity level l.
c^E	Exploitation cost.
$\beta_{l,f}$	Unit exploitation cost of patches with severity level l and age f.
r	Single period cost.
V	Total discounted system cost.
Π	Patching policy.
γ	Discount factor.
δ	Policy parameter.
π	A stochastic patching policy.
$\widehat{\mathcal{A}_t}$	Estimator of advantage function used in policy gradient methods.
ϵ	Clip parameter.

Table 2. Summary of Notation

State Transition Function

Given the system state $\boldsymbol{S}_t$ and action $\boldsymbol{A}_t$ at the current decision period t, we define a state transition function $P: \mathbb{S} \times \mathbb{A} \times \mathbb{S} \to [0,1]$ to calculate the possibility of the system state $\boldsymbol{S}_{t+1}$ at the next decision period t+1, that is, $P(\boldsymbol{S}_{t+1}|\boldsymbol{S}_t, \boldsymbol{A}_t)$, where $\boldsymbol{S}_t, \boldsymbol{S}_{t+1} \in \mathbb{S}$ and $\boldsymbol{A}_t \in \mathbb{A}$. This state transition process is stochastic, owing to both the probability of failure during patch deployment and the randomness of the patch arrival process. We emphasize that our model does not require specific information about this stochastic process, as the proposed method can interact with and learn from the environment in real-time. To simulate the

environment in our experiments, we suppose the failure rate of patches with age f in the system is φ_f, and the arrivals of patches with severity level l follows the Poisson distribution with arrival rate λ_l (Dey et al. 2015; Jia et al. 2021). Figure 1 illustrates the state transition process, providing an example where the maximum patch age in the system is set to 3, which implies that after three decision periods, the failure rates and severity levels of patches will stabilize. For instance, at the decision period t, let $s_t^{1,1}$ denote the number of patches with severity level 1 and age 1, and $a_t^{1,1}$ represent the action taken for this type of patches. Then, at decision period t+1, the number of patches with severity level 1 and age 2 $s_{t+1}^{1,2} = s_t^{1,1} - a_t^{1,1}(1-\varphi_1)$, where φ_1 is the failure rate of patches with age 1. Patches with age 0 at decision period t+1 are newly arrived patches at the next period and the number of patches with the maximum age (i.e., 3) $s_{t+1}^{1,3} = s_t^{1,2} - a_t^{1,2}(1-\varphi_2) + s_t^{1,3}$.

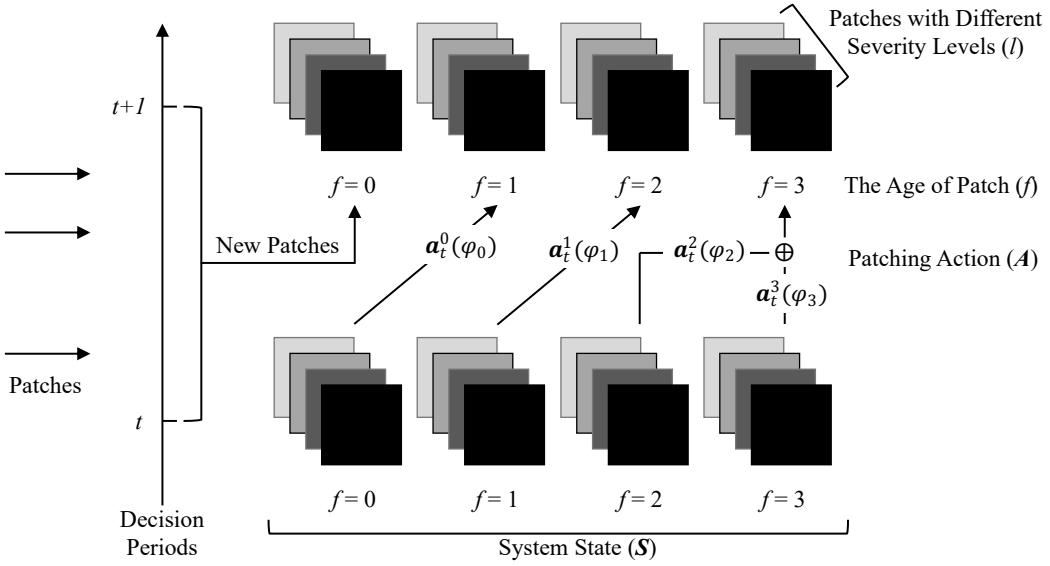

Figure 1. State Transition Process

System Cost Functions

There are two main types of costs to consider: patching costs associated with patching operations and exploitation costs stemming from security risks before the patch deployment.

Patching Costs. We define a *fixed patching cost* c^{FP} and a *variable patching cost* c^{VP}. The fixed patching cost comprises various expenses incurred during patching operations, such as human resource cost, device resource cost, downtime cost, system rebooting cost, business disruption cost, and so on. The variable patching cost varies depending on the type and quantity of patches requiring deployment. For each patch, the unit variable cost consists of expenses specific to its type, such as the testing cost and patching failure cost. During the testing process, IT experts assess the system to ensure that deploying the patch will not cause any issues. However, it is important to note that testing does not guarantee the patches will not encounter problems in the production environment. Therefore, we define a unit patching failure cost σ_l for patches with severity level l, then the expected patching failure cost of a patch with severity level l and age f is $\varphi_f \sigma_l$, where φ_f is the failure rate of patches with age f. We also use a unit variable patching cost θ_l for patches with severity level l to capture other variable patching costs not related to the age of patches. The former is only incurred in case of patch deployment failure, while the latter is triggered with every patch deployment. Factoring in these costs, the total variable cost at decision period t takes the following form:

$$\mathbb{E}[c_t^{VP}(\boldsymbol{A}_t)] = \sum_{l=1}^{L}\sum_{f=0}^{F}(\theta_l + \varphi_f\sigma_l)a_t^{l,f}\,. \tag{1}$$

Then, the total patching cost at decision period t is $c_t^{FP} + c_t^{VP}$.

Exploitation Cost. A security vulnerability is a type of bug in a system or application that can be exploited by malicious actors. Deploying patches can effectively mitigate these known vulnerabilities within a system. However, because of the considerable expenses involved, including those related to business disruption, system reboots, and the potential for patching failures, enterprises typically postpone deploying released patches. Before patches are deployed, hackers may exploit vulnerabilities for malicious activities. Delayed installation of a patch with a higher severity level can escalate exploitation costs. As mentioned, the longer the delay in patch installation, the higher the probability of an attack, indicating a corresponding increase in expected exploitation costs. Therefore, in our model, we introduce an exploitation cost c^E to quantify the expense incurred from being attacked due to delayed patch installation. In the calculation, we utilize a unit exploitation cost $\beta_{l,f}$ for patches with severity level l and age f. The expected exploitation cost at decision period t is given by:

$$\mathbb{E}[c_t^E(\boldsymbol{S}_t, \boldsymbol{A}_t)] = \sum_{l=1}^{L}\sum_{f=0}^{F}\beta_{l,f}\left(s_t^{l,f} - (1-\varphi_f)a_t^{l,f}\right), \tag{2}$$

where $s_t^{l,f} - (1-\varphi_f)a_t^{l,f}$ is the number of patches with severity level l and age f that we opt to delay or that failed to deploy at decision period t.

There is also a check interval exploitation cost between the arrival time of new patches and the next decision point. Given the time interval of each decision periods and patch information, the expected check interval exploitation cost is fixed and cannot be avoided or decreased by the patching decisions at decision points. For simplicity, we do not include this cost in our model since it has no influence on the patching policy.

Taking into account above costs, the *expected single-period cost* for decision period t takes the following form:

$$r_t(\boldsymbol{S}_t, \boldsymbol{A}_t) = sgn(\|\boldsymbol{A}_t\|)c^{FP} + \mathbb{E}[c_t^{VP}(\boldsymbol{A}_t)] + \mathbb{E}[c_t^E(\boldsymbol{S}_t, \boldsymbol{A}_t)], \tag{3}$$

where $sgn(\cdot)$ is a sign function:

$$sgn(\|\boldsymbol{A}_t\|) = \begin{cases} 1 & if\ \|\boldsymbol{A}_t\| > 0, \\ 0 & if\ \|\boldsymbol{A}_t\| = 0. \end{cases} \tag{4}$$

Our objective is to obtain an optimal policy Π^* minimizing the *value function* of the system state, that is, the *expected total discounted system cost*:

$$V^{\Pi^*}(\boldsymbol{S}_t) = \min_{\boldsymbol{A}_t} \mathbb{E}\left[\sum_{k=0}^{T-t}\gamma^k r_{t+k}(\boldsymbol{S}_{t+k}, \boldsymbol{A}_{t+k})\right], \tag{5}$$

where γ is the discount factor, and the bellman equation for V^{Π} is given by

$$V^{\Pi}(\boldsymbol{S}_t) = r(\boldsymbol{S}_t, \boldsymbol{A}_t) + \gamma \sum_{\boldsymbol{S}_{t+1}\in\mathbb{S}} P(\boldsymbol{S}_{t+1}|\boldsymbol{S}_t, \boldsymbol{A}_t)V^{\Pi}(\boldsymbol{S}_{t+1})\,, \tag{6}$$

Unfortunately, owing to the curse of dimensionality, we are not able to obtain the optimal policy by addressing the MDP problem directly. Hence, in the next section, we develop a reinforcement learning approach to solve the patching problem effectively. To train the reinforcement learning agent, we simulate a patching environment based on our model in this section.

A Deep Reinforcement Learning Approach

In the patching decision problem, the action space rapidly expands with the increasing number and variety of patches. In this section, we initially present a refined action space and a decision decomposition method to mitigate the challenges posed by the large action space. Building upon this, we introduce the action-

decomposed proximal policy optimization (ADPPO) method to learn an effective patching policy for the patching decision model.

Refined Action Space and Decision Decomposition

Numerous disadvantages arise from learning in a large action space. First, exploring in a large action space is challenging. The agent may struggle to explore all possible actions, leading to slower convergence and suboptimal policies. Second, a large action space also improves sample complexity, making training less sample-efficient. The learning process of an agent requires more sample data. Third, policy representation in a large action space demands more sophisticated function approximators, resulting in a slower learning process and diminishing the method's capacity for generalization. Fourth, the increased computational complexity can slow down the learning process and make real-time decision-making impractical in some cases. Meanwhile, it also increases memory requirements, potentially making learning unfeasible in resource-constrained scenarios. To address these issues, we propose refining the action space based on the following rule:

Refined Action Space. *At each decision period, if one decide to deploy a patch with severity level l and age f, all the patches with severity level l and an age equal to or greater than f should be deployed.*

It is straightforward that, at a decision period t, the fixed patching cost c_t^{FP} is incurred when the system manager decides to deploy a patch with severity level l and age f. Patches with severity level l and an age equal to or greater than f have the same variable cost θ_l, equal or lower failure risks, and equal or higher exploitation costs. From an economic perspective, these patches are either equally urgent as or more urgent than the patch with severity level l and age f. Hence, the system manager should install them at the same decision period to minimize the total system cost.

Further, we decompose the patching decision for each period into two levels of decision-making. First, deciding whether to perform the patching operation, and then, identifying which patches to deploy. The reasons for the decomposition are as follows. First, as we mentioned, as long as we decide to perform the patching operation, there will always be a fixed patching cost, regardless of the number of patches deployed. To minimize the system cost, it is shown that the patching operation should be carried out only when the cumulative severity of patches to be deployed exceeds a specific threshold (Jia et al. 2021). Hence, we can first determine whether to perform the patching operation based on the system state. If we decide not to patch, no operation is performed during this period; otherwise, we next identify which patches to deploy. We assume that if we opt to perform the patching operation in the first step, then for each severity level of patches, at least patches with the maximum age should be deployed. This assumption is reasonable because, upon choosing to proceed with the installation operation, the patches prioritized for installation are those with the lowest variable patching cost and the highest exploitation cost (i.e., patches with the maximum age). If patches with severity level l and the maximum age F should not be deployed, then patches with severity level l will never be deployed. In practice, such a scenario is extremely rare. Even if it does happen, there is no necessity to consider it in our model, as patches of such type should never be deployed.

The refined action space and decision decomposition can reduce the size of action space effectively. Figure 2 shows an example of our action space refining process. The number pairs within the parentheses represent the types of patches. For instance, (1,0) means patches with severity level 1 and age 0. In the example, the system manager decides to deploy patches at the first step, then at least the patches with the maximum age for each severity level should be deployed, i.e., patches of (1,3) and (2,3). If the manager decides to install patches of (1,0) and (2,1), then patches with the same severity levels as them and older than them should also be deployed, i.e., patches of (1,1), (1,2), and (2,2). Suppose there are 2 different severity levels and the maximum age is 3 periods old, the number of possible actions will be $N^{2\times4}$, where N stands for the number of each type of patches. However, under the refined action space, we only have 4^2+1 different actions at each decision period, which is significantly smaller than the original action space.

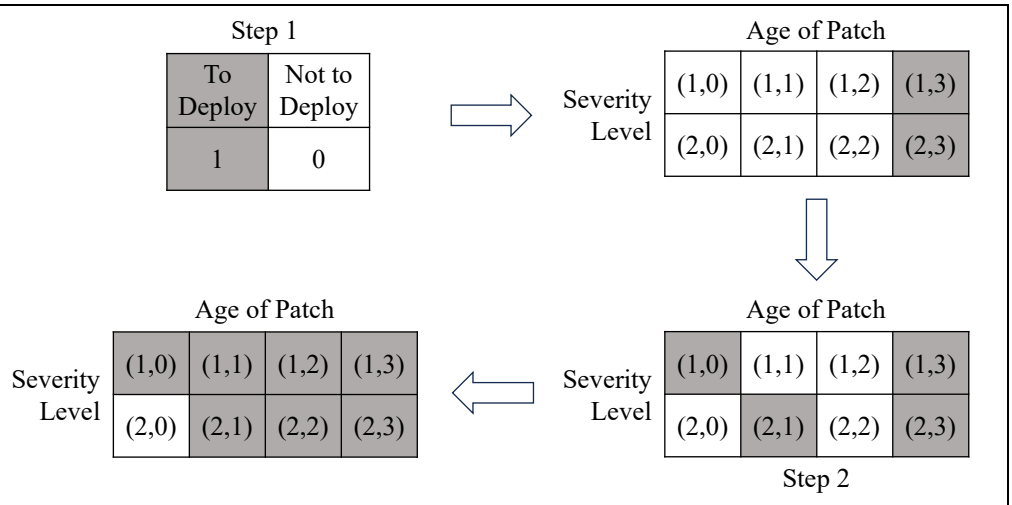

Figure 2. An Example of Action Space Refining

Action-Decomposed Proximal Policy Optimization

In this subsection, we develop a new algorithm–Action-Decomposed Proximal Policy Optimization (ADPPO)–to learn the (approximate) optimal policy for the patching decision model. The proposed approach is based on the above decision structure and a state-of-the-art reinforcement learning algorithm–PPO (Schulman et al., 2017).

The success of PPO can be primarily attributed to the adoption of the following clipped surrogate objective function in the training process:

$$L^{CLIP}(\delta) = \mathbb{E}\left\{\min\left[\frac{\pi_\delta(\boldsymbol{A_t}|\boldsymbol{S_t})}{\pi_{\delta_{old}}(\boldsymbol{A_t}|\boldsymbol{S_t})}\widehat{\mathcal{A}}_t, \text{clip}\left(\frac{\pi_\delta(\boldsymbol{A_t}|\boldsymbol{S_t})}{\pi_{\delta_{old}}(\boldsymbol{A_t}|\boldsymbol{S_t})}, 1-\epsilon, 1+\epsilon\right)\widehat{\mathcal{A}}_t\right]\right\}, \tag{7}$$

where δ is the policy parameters, δ_{old} is the policy parameters before update, π_δ denotes a stochastic policy, $\widehat{\mathcal{A}}_t$ is an estimator of the advantage function used in policy gradient methods, $\frac{\pi_\delta(\boldsymbol{A_t}|\boldsymbol{S_t})}{\pi_{\delta_{old}}(\boldsymbol{A_t}|\boldsymbol{S_t})}$ stands for the probability ratio, and ϵ is a hyperparameter for clipping. The clipped part in the objective function ensures that the ratio stays within a specified range $[1-\epsilon, 1+\epsilon]$, preventing excessively large policy updates that can result in falling off the cliff and learning a worse policy.

While PPO demonstrates outstanding performance in reinforcement learning tasks, it cannot be directly applied to address our patching decision model. Furthermore, as we mentioned, learning a satisfactory policy in a large discrete action space still poses challenges. To tackle these challenges and obtain a better patching policy, we introduce the ADPPO method. This approach divides the actor network into two components: a *leader network* and an *executor network*, aligning with the decision decomposition and PPO framework. The leader network is responsible for learning whether to execute the patching operation, while the executor network concentrates on learning which patches to deploy. As we demonstrated in the last subsection, such architectural design can significantly reduce the size of action space. This reduction has the potential to enhance the learning process's efficiency and result in a policy that is more effective. Figure 3 shows the framework of the proposed ADPPO.

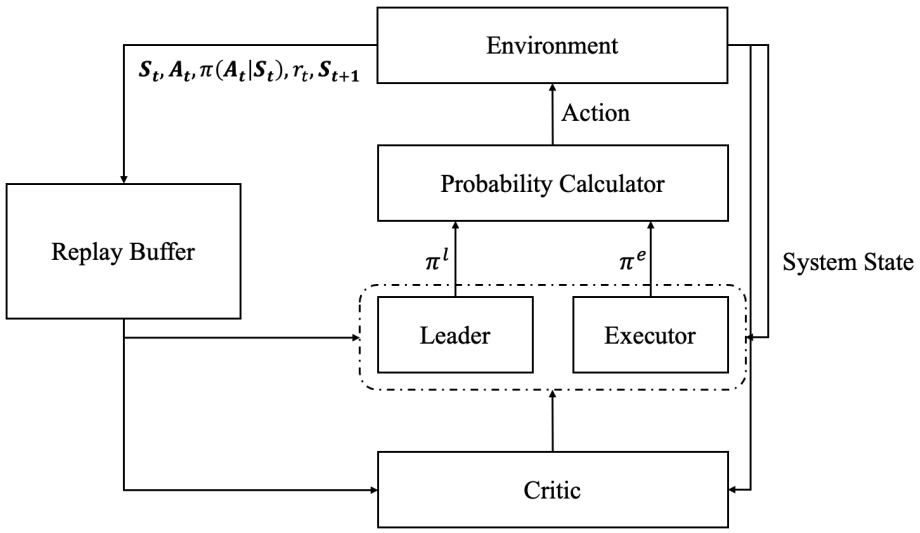

Figure 3. The Framework of Action-Decomposed Proximal Policy Optimization (ADPPO)

Algorithm 1. Action-Decomposed Proximal Policy Optimization (ADPPO)

Input: initial leader policy, executor policy, and value function parameters.
Output: near-optimal patching policy.
for *episode* = 0, 1, 2, … **do**
  for t = 0, 1, 2, …, T **do**
    Observe the current system state from the simulated environment.
    Obtain the action with leader network, executor network, and probability calculator (Equation 8).
    Obtain the value with critic network.
    Execute the action and obtain the cost and next state from the simulated environment.
    Add the trajectory to the replay buffer.
  end
  Compute costs-to-go.
  Estimate the advantage function $\widehat{\mathcal{A}_t}$
  for *epoch* = 0, 1, 2, … **do**
    Shuffle the indices of the training data and split it into mini-batches.
    Update the policy (leader network and executor network) by minimizing the clipped surrogate objective functions in Equations (9) and (10).
    Update the value function (critic network) by minimizing the clipped loss:

$$L_V^{CLIP} = \max\left[\left(V_t - V_{target}\right)^2, \left(\text{clip}(V_t, V_{t-1} - \epsilon, V_{t-1} + \epsilon) - V_{target}\right)^2\right].$$

  end
end

In ADPPO, we use a *probability calculator* to obtain the probability of the whole action after leader and executor networks provide probabilities of their own actions. The probability is calculated as the following:

$$\pi(\boldsymbol{A_t}|\boldsymbol{S_t}) = \begin{cases} \pi^l\left(\boldsymbol{A_t^l}|\boldsymbol{S_t}\right) & if\ \boldsymbol{A_t} = \boldsymbol{0}; \\ \pi^l\left(\boldsymbol{A_t^l}|\boldsymbol{S_t}\right)\pi^e(\boldsymbol{A_t^e}|\boldsymbol{S_t}) & otherwise, \end{cases} \tag{8}$$

where $\boldsymbol{A_t}$ is decomposed into $\boldsymbol{A_t^l}$ and $\boldsymbol{A_t^e}$, $\pi^l\left(\boldsymbol{A_t^l}|\boldsymbol{S_t}\right)$ is the probability of that the leader takes the action $\boldsymbol{A_t^l}$, and $\pi^e(\boldsymbol{A_t^e}|\boldsymbol{S_t})$ is the probability of action $\boldsymbol{A_t^e}$ for the executor when $\boldsymbol{S_t}$ is given.

Then, if $\boldsymbol{A_t^l = 0}$,

$$L^{CLIP}(\delta) = \mathbb{E}\left\{\min\left[\frac{\pi^l_{\delta^l}(\boldsymbol{A_t^l}|\boldsymbol{S_t})}{\pi^l_{\delta^l_{old}}(\boldsymbol{A_t^l}|\boldsymbol{S_t})}\widehat{\mathcal{A}_t}, \text{clip}\left(\frac{\pi^l_{\delta^l}(\boldsymbol{A_t^l}|\boldsymbol{S_t})}{\pi^l_{\delta^l_{old}}(\boldsymbol{A_t^l}|\boldsymbol{S_t})}, 1-\epsilon, 1+\epsilon\right)\widehat{\mathcal{A}_t}\right]\right\}; \tag{9}$$

otherwise,

$$L^{CLIP}(\delta) = \mathbb{E}\left\{\min\left[\begin{array}{c}\frac{\pi^l_{\delta^l}(\boldsymbol{A_t^l}|\boldsymbol{S_t})\pi^e_{\delta^e}(\boldsymbol{A_t^e}|\boldsymbol{S_t})}{\pi^l_{\delta^l_{old}}(\boldsymbol{A_t^l}|\boldsymbol{S_t})\pi^e_{\delta^e_{old}}(\boldsymbol{A_t^e}|\boldsymbol{S_t})}\widehat{\mathcal{A}_t}, \\ \text{clip}\left(\frac{\pi^l_{\delta^l}(\boldsymbol{A_t^l}|\boldsymbol{S_t})\pi^e_{\delta^e}(\boldsymbol{A_t^e}|\boldsymbol{S_t})}{\pi^l_{\delta^l_{old}}(\boldsymbol{A_t^l}|\boldsymbol{S_t})\pi^e_{\delta^e_{old}}(\boldsymbol{A_t^e}|\boldsymbol{S_t})}, 1-\epsilon, 1+\epsilon\right)\widehat{\mathcal{A}_t}\end{array}\right]\right\}. \tag{10}$$

We minimize this clipped surrogate objective function, as our objective is to minimize the total system cost in the patching decision problem. Algorithm 1 presents the pseudocode for ADPPO.

Experiments and Results

In this section, we conduct a series of experiments to compare our proposed approach with benchmarks. We select the PPO-based method and the dynamic patching policy (DPP) as our benchmarks, because they have been shown to be superior to other existing approaches (Hore et al., 2023; Jia et al., 2021), such as VULCON (Farris et al., 2018), VPSS (Hore et al., 2022), time-based policy, total-control policy, and hybrid policy (Dey et al., 2015). In the prior study (Hore et al., 2023), the PPO-based method is called Deep VULMAN and the authors utilize it to prioritize vulnerabilities. In our experiments, however, we implement and tailor it specifically for the patching decision problem. We also introduce a DQN-based method as another benchmark, as DQN is also a well-known RL algorithm (Mnih et al., 2015).

Table 3 shows the key hyperparameters of DQN, PPO, and ADPPO. The DQN-based method has a Q-Netwrok and a target network with one hidden layer (64 neurons) and uses a linear decaying epsilon (maximum value is 1.00 and minimum value is 0.01) for exploration. The exploration fraction is 0.1. In the PPO-based method, we use neural networks with one hidden layer (64 neurons) for the actor network and critic network. Similarly, in the proposed ADPPO, we use neural networks with one hidden layer (64 neurons) for the leader network, executor network, and critic network. For all networks, we use the ReLU activation function and the Adam optimizer. We use 32 vectorized environments for training RL agents. In contrast to a single environment, vectorized environments allow agents to interact with multiple environments per step to improve the computational resource efficiency and speed up the training process.

Hyperparameter	DQN	PPO and ADPPO
Number of environments	32	32
Number of steps	512	512
Total steps	1,000,000	1,000,000
Replay buffer size	10,000	—
Batch size	64	32×512
Train frequency	Every 4 steps	—
Minibatch size	—	4096
Number of epochs	—	10
Discount factor	0.99	0.99
GAE parameter	—	0.95
Clipping parameter ϵ	—	0.2

Table 3. Key Hyperparameters

Table 4 provides the key parameters of the simulated patching environment. We assume there are two severity levels of vulnerabilities and the maximum age of patches is 3, which means that if patches are delayed for more than 3 decision cycles, their exploitation costs and failure rates remain unchanged. When the system manager decides to patch the system, there is a fixed patching cost regardless of the number of patched installed, and patches of varying ages have corresponding failure rates of 0.2, 0.1, and 0.01, respectively. No matter the success or failure of a patch installation, it incurs a variable patching cost. Meanwhile, as shown in Table 4, patches of different severity levels have different arrival rates, unit exploitation costs, unit patching failure costs, and unit variable patching costs. All costs are measured in US dollars.

Parameter	Value
Number of severity levels	2
Maximum age	3
Arrival rates	[0.6, 0.2]
Unit exploitation costs	[[20, 30, 50], [40, 60, 90]]
Failure rates	[0.2, 0.1, 0.01]
Unit patching failure costs	[30, 60]
Unit variable patching costs	[2, 4]
Fixed patching cost	300

Table 4. Patching Model Settings

We conduct all experiments in Python on a MacBook Pro with a 2 GHz quad-core Intel i5 processor and 16 GB of RAM, running macOS Ventura 13.5.1. Figure 4 depicts the training results. It shows that, compared to DQN and PPO, ADPPO can lead to a significantly lower average system cost (i.e., the average cost of the last 512 steps). The final patching policy learned by ADPPO can reduce the average cost by about 4% compared to DQN and PPO. The advantage of the ADPPO algorithm lies in its architectural design, which enables it to leverage the prior knowledge of the patching decision problem during the learning process.

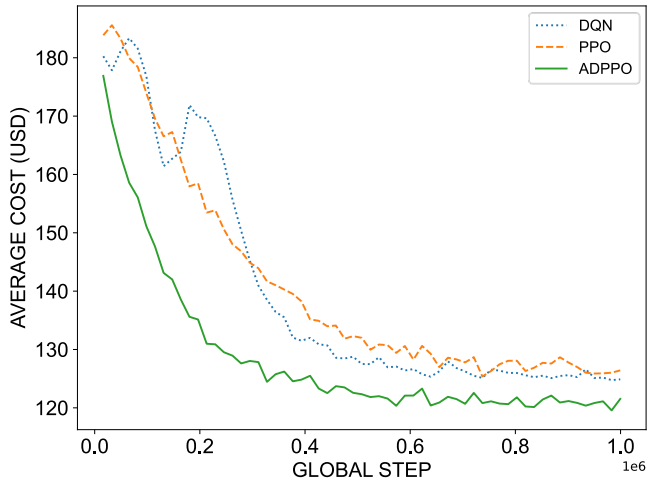

Figure 4. Learning Curves

With the trained agents, we conduct another experiment to compare ADPPO with benchmarks in the simulated patching environment. We first calculate the optimal policy under DPP based on the parameters of the patching environment. Then, we let an agent under this derived policy (i.e., DPP) and the three learned agents (i.e., DQN, PPO, and ADPPO) interact with the simulated environment and record costs of each step. Figure 5 shows the total discounted system costs (the average result after 100 runs) under different policies with varying decision steps. Compared to DPP, DQN, and PPO, ADPPO can reduce total costs by an average of 6.78%, 3.65%, and 3.53%, respectively, indicating that the proposed ADPPO is consistently and significantly superior to benchmarks.

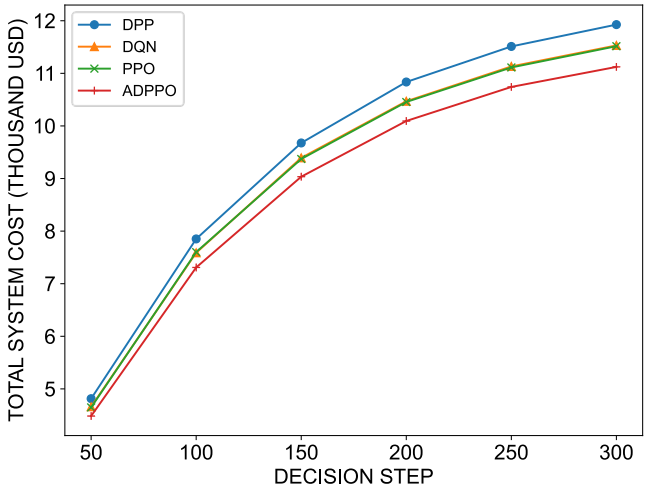

Figure 5. Policy Comparison

The strengths of ADPPO are as follows. First, compared to DPP, ADPPO takes into account the patching failure risk and other variable patching costs, and allows the system manager to choose which patches to deploy. In contrast, to control the size of action space and state space, a system manager under DPP can only choose to deploy all the patches or do nothing at each decision period and the patching cost is simplified to a fixed patching cost. Meanwhile, when the system's action space and state space is large, ADPPO can still learn a patching policy efficiently by leveraging neural networks and its architectural design. Second, ADPPO has a leader and an executor to decide whether to deploy and which patches to deploy, while there is only one actor to make every decision in PPO. As we mentioned, the architectural design allows ADPPO to enjoy a smaller action space than PPO and thus make decision more easily and better with the prior knowledge. Noteworthily, in our model, we take into account more operational costs, dynamics, and uncertainties than the prior study (Hore et al. 2023) that proposes the PPO-based method (Deep VULMAN). Even though we implement the PPO-based method considering these additional factors, our proposed ADPPO still proves to be more effective. Third, compared to DQN, ADPPO benefits not only from its unique structural design but also from the advantages inherent in policy gradient methods. For example, ADPPO directly optimizes the policy using gradients, leading to potentially better policies in terms of minimizing long-term costs. In contrast, DQN optimizes the action-value function, which is indirectly used to determine the policy. This can lead to suboptimal policies, especially in environments with complex cost structures. Therefore, the total system cost incurred by DQN is always higher than ADPPO even when they share the same action space in our experiment. In a word, the superiority of ADPPO arises from its architectural design and thorough consideration of various costs, dynamics, and uncertainties in the patching decision problem.

Concluding Remarks

Patching in a timely manner has been demonstrated as one of the most effective ways to protect enterprise information systems from cyberattacks. However, in practice, as patching operations are not cost-free, enterprises typically delay the installation of patches. Achieving a balance between operational expenses and system security risks remains an ongoing challenge. In this paper, we propose a novel deep reinforcement learning-based approach to solve the patching decision problem. Specifically, we formulate the patching problem as a MDP problem with a thorough consideration of various costs, dynamics, and uncertainties, such as patching costs (fixed and variable costs), patching failure likelihoods and costs, exploitation costs, and the dynamics of these factors after the patch is released. To avoid the curse of dimensionality of the MDP problem and address it effectively, we develop a novel reinforcement learning method: Action-Decomposed Proximal Policy Optimization (ADPPO). Experimental results show that the proposed ADPPO is significantly superior to benchmarks. Compared to DPP, DQN, and PPO, ADPPO can reduce total costs by an average of 6.78%, 3.65%, and 3.53%, respectively.

Our study makes the following contributions. First, this design science research provides a viable and novel artifact for enterprises to assist them in security patch management. Second, we formulate the patching problem with a thorough consideration of various costs, dynamics, and uncertainties. Our results demonstrate that taking these factors into account is valuable for minimizing the total enterprise cost. Managing patches from an economic standpoint is more effective than solely from a security engineering perspective. Third, we also make a contribution to the field of reinforcement learning by proposing a new algorithm (ADPPO). This method is also applicable to other analogous optimal control scenarios, such as inventory management and financial trading. Our study also has significant practical implications. Owing to the advantages of the proposed approach, enterprises can adopt it across a wide range of security patch management scenarios. The proposed approach is able to dynamically adjust patching policies by interacting with and learning from the environment in real-time, effectively reducing enterprise costs. It is recommended that enterprises leverage ADPPO to enhance automation in security patch management.

The limitations are as follows. First, we do not use any real operational datasets from enterprises for the patching decision problem. Once we obtain a real-world dataset, it is straightforward to extend our approach. The simplest way is to modify the input layers of the neural networks in ADPPO to incorporate detailed real-world data. Second, in the current setting, the system manager makes patching decisions periodically (e.g., every week). Before a patching decision, the system state is checked through system scan, which may incur costs as well. Future research is expected to extend the method to achieve the joint optimization of scanning and patching decisions.

References

Beattie, S., Arnold, S., Cowan, C., Wagle, P., Wright, C., & Shostack, A. (2002). Timing the application of security patches for optimal uptime. *LISA*, 233-242.

Bondar, T., Assal, H., & Abdou, A. (2023). Why do internet devices remain vulnerable? A survey with system sdministrators. *The Workshop on Measurements, Attacks, and Defenses for the Web (MADWeb 2023),* San Diego, CA.

Bozorgi, M., Saul, L. K., Savage, S., & Voelker, G. M. (2010). Beyond heuristics: Learning to classify vulnerabilities and predict exploits. *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Washington DC, 105-114.

Cavusoglu, H., Cavusoglu, H., & Zhang, J. (2008). Security patch management: share the burden or share the damage? *Management Science, 54*(4), 657-670.

Costa, T. F. & Tymburibá, M. (2022). Challenges on prioritizing software patching. *Proceedings of the 15th International Conference on Security of Information and Networks (SIN)*, Sousse, Tunisia, 1-8.

Dey, D., Lahiri, A., & Zhang, G. (2015). Optimal policies for security patch management. *INFORMS Journal on Computing, 27*(3), 462-477.

Dissanayake, N., Zahedi, M., Jayatilaka, A., & Babar, M. A. (2022). Why, how and where of delays in software security patch management: An empirical investigation in the healthcare sector. *Proceedings of the ACM on Human-computer Interaction* (6:CSCW2), 1-29.

Elbaz, C., Rilling, L., & Morin, C. (2020). Fighting n-day vulnerabilities with automated cvss vector prediction at disclosure. *Proceedings of the 15th International Conference on Availability, Reliability and Security*, online, 1-10.

Fang, Y., Liu, Y., Huang, C., & Liu, L. (2020). Fastembed: Predicting vulnerability exploitation possibility based on ensemble machine learning algorithm. *Plos One, 15*(2), e0228439.

Farris, K. A., Shah, A., Cybenko, G., Ganesan, R., & Jajodia, S. (2018). Vulcon: A system for vulnerability prioritization, mitigation, and management. *ACM Transactions on Privacy and Security (TOPS), 21*(4), 1-28.

Gong, X., Xing, Z., Li, X., Feng, Z., & Han, Z. (2019). Joint prediction of multiple vulnerability characteristics through multi-task learning. *Proceedings of the 24th International Conference on Engineering of Complex Computer Systems (ICECCS)*, Guangzhou, China, 31-40.

Hore, S., Moomtaheen, F., Shah, A., & Ou, X. (2022). Towards optimal triage and mitigation of context-sensitive cyber vulnerabilities. *IEEE Transactions on Dependable and Secure Computing, 20*(2), 1270-1285.

Hore, S., Shah, A., & Bastian, N. D. (2023). Deep Vulman: A deep reinforcement learning-enabled cyber vulnerability management framework. *Expert Systems with Applications, 221*, 119734.

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.

Jacobs, J., Romanosky, S., Suciu, O., Edwards, B., & Sarabi, A. (2023). Enhancing vulnerability prioritization: Data-driven exploit predictions with community-driven insights. *Proceedings of 2023 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, Delft, Netherlands, 194-206.

Jenkins, A., Liu, L., Wolters, M., & Vaniea, K. (2024). Not as easy as just update: Survey of system administrators and patching behaviours. *Proceedings of the CHI Conference on Human Factors in Computing Systems (CHI' 24)*, Honolulu, HI, 1-17.

Jia, Q., Qu, X., & Jiang, Z. (2021). A dynamic patching policy for enterprise information systems. *Proceedings of the 14th China Summer Workshop on Information Management (CSWIM 2021)*, online, 159-164.

Jiang, Y. & Atif, Y. (2020). An approach to discover and assess vulnerability severity automatically in cyber-physical systems. *Proceedings of the 13th International Conference on Security of Information and Networks*, Merkez, Turkey, 1-8.

Le, T. H., Chen, H., & Babar, M. A. (2022). A survey on data-driven software vulnerability assessment and prioritization. *ACM Computing Surveys, 55*(5), 1-39.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., & Wierstra, D. (2015). Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*.

Liu, X. (2023). Dynamic coupon targeting using batch deep reinforcement learning: An application to livestream shopping. *Marketing Science, 42*(4), 637-658.

Mell, P., Scarfone, K., & Romanosky, S. (2006). Common vulnerability scoring system. *IEEE Security & Privacy, 4*(6), 85-89.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., & Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning. *Proceedings of the International Conference on Machine Learning: PMLR*, New York, New York, 1928-1937.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., & Ostrovski, G. (2015). Human-level control through deep reinforcement learning. *Nature, 518*(7540), 529-533.

Oroojlooyjadid, A., Nazari, M., Snyder, L. V., & Takáč, M. (2022). A deep q-network for the beer game: Deep reinforcement learning for inventory optimization. *Manufacturing & Service Operations Management, 24*(1), 285-304.

Rajivan, P., Aharonov-Majar, E., & Gonzalez, C. (2020). Update now or later? Effects of experience, cost, and risk preference on update decisions. *Journal of Cybersecurity, 6*(1), tyaa002.

Reeder, R. W., Ion, I., & Consolvo, S. (2017). 152 simple steps to stay safe online: Security advice for non-tech-savvy users. *IEEE Security & Privacy, 15*(5), 55-64.

Schulman, J., Levine, S., Abbeel, P., Jordan, M., & Moritz, P. (2015). Trust region policy optimization. *Proceedings of the International Conference on Machine Learning: PMLR*, Lille, France, 1889-1897.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Souppaya, M. & Scarfone, K. (2013). Guide to enterprise patch management technologies. *NIST Special Publication, 800*(40).

Spanos, G. & Angelis, L. (2018). A multi-target approach to estimate software vulnerability characteristics and severity scores. *Journal of Systems and Software, 146*, 152-166.

Tavabi, N., Goyal, P., Almukaynizi, M., Shakarian, P., & Lerman, K. (2018). Darkembed: Exploit prediction with neural language models. *Proceedings of the AAAI Conference on Artificial Intelligence*, New Orleans, Lousiana.

Wang, B., Li, X., de Aguiar, L. P., Menasche, D. S., & Shafiq, Z. (2017). Characterizing and modeling patching practices of industrial control systems. *Proceedings of the ACM on Measurement and Analysis of Computing Systems,* 1-23.

Wang, H., Liu, N., Zhang, Y., Feng, D., Huang, F., Li, D., & Zhang, Y. (2020). Deep reinforcement learning: A survey. *Frontiers of Information Technology & Electronic Engineering, 21*(12), 1726-1744.

Yang, C., Feng, Y., & Whinston, A. (2022). Dynamic pricing and information disclosure for fresh produce: An artificial intelligence approach. *Production and Operations Management,* 31(1), 155-171.