3-2023

Deep VULMAN: A Deep Reinforcement Learning-enabled Cyber Vulnerability Management Framework

Soumyadeep Hore

Ankit Shah

Nathaniel D. Bastian
Army Cyber Institute, U.S. Military Academy, nathaniel.bastian@westpoint.edu

Expert Systems With Applications 221 (2023) 119734

Contents lists available at ScienceDirect

Expert Systems With Applications

journal homepage: www.elsevier.com/locate/eswa

Deep VULMAN: A deep reinforcement learning-enabled cyber vulnerability management framework

Soumyadeep Hore [a], Ankit Shah [a,*], Nathaniel D. Bastian [b]

[a] University of South Florida, Department of Industrial & Management Systems Engineering, Tampa, FL, 33620, United States of Amercia
[b] United States Military Academy, Army Cyber Institute, West Point, NY, 10996, United States of America

ARTICLE INFO

Keywords:
Cyber vulnerability management
Vulnerability prioritization
Security resources optimization
Deep reinforcement learning
Integer programming
DRL defender framework

ABSTRACT

Cyber vulnerability management is a critical function of a cybersecurity operations center (CSOC) that helps protect organizations against cyber-attacks on their computer and network systems. Adversaries hold an asymmetric advantage over the CSOC, as the number of deficiencies in these systems is increasing at a significantly higher rate compared to the expansion rate of the security teams to mitigate them. The current approaches in cyber vulnerability management are deterministic and one-time decision-making methods, which do not consider future uncertainties when prioritizing and selecting vulnerabilities for mitigation. These approaches are also constrained by the sub-optimal distribution of resources, providing no flexibility to adjust their response to fluctuations in vulnerability arrivals. We propose a novel framework, Deep VULMAN, consisting of a deep reinforcement learning agent and an integer programming method to fill this gap in cyber vulnerability management process. Our sequential decision-making framework, first, determines the near-optimal amount of resources to be allocated for mitigation under uncertainty for a given system state, and then determines the optimal set of prioritized vulnerability instances for mitigation. Results show that our framework outperforms the current methods in prioritizing the selection of important organization-specific vulnerabilities, on both simulated and real-world vulnerability data, observed over a one-year period.

1. Introduction

Adversaries are actively looking to exploit unpatched vulnerabilities in the computer and network systems to cause significant damage to public and private organizations. Such pervasive cyber-attacks are often aimed at debilitating the security posture of an organization and capturing their high value assets. Recently, the United States White House had issued a memo urging organizations to *promptly identify and remediate vulnerabilities* in their systems, among other recommendations to bolster cybersecurity against the adversaries (WhiteHouse, 2021). Major challenges faced by the organizations to implement this recommendation result from a significant recent increase in the number of new vulnerabilities that are reported in the National Vulnerability Database (NVD) (NIST, 2022), as well as the lack of security personnel (resources) available to mitigate them. This has resulted in vulnerabilities persisting in the computer and network systems of the organizations for a long time, thereby creating a significant advantage for the adversaries. Hence, it is critical to develop resource-constrained approaches for effectively identifying and mitigating important organization-specific security vulnerabilities to combat adversarial exploitation and minimizing damage from cyber-attacks.

1.1. Cyber vulnerability management process

Cyber vulnerability management is a continuous process involving scanning of an organization's network and mitigating the identified vulnerabilities. Fig. 1 shows a schematic of the cyber vulnerability management process. A typical cyber vulnerability management process starts with the scanning of the software and hardware components of an organization's network with a vulnerability scanner (such as Tenable, Qualys, or IBM) to find vulnerabilities reported in the NVD. The generated vulnerability report contains all vulnerability instances found in the network along with their attributes, which include the common vulnerability exposure (CVE) code, host name, description, and the common vulnerability scoring system (CVSS) severity rating, among others. The security teams at the cybersecurity operations centers (CSOCs) then assign the available resources to mitigate the vulnerability instances selected based on certain vulnerability triage mechanisms. Example of actions taken by security personnel are applying patches (vendor-supplied or CSOC-designed), upgrading software, disabling services, and adding IP filters, among others. The unmitigated vulnerabilities accumulate in the queue for selection in the subsequent

* Correspondence to: University of South Florida, 4202 East Fowler Avenue, Mail-Stop: ENG-030, Tampa, FL 33620-5530, United States of America.
E-mail addresses: soumyadeep@usf.edu (S. Hore), ankitshah@usf.edu (A. Shah), nathaniel.bastian@westpoint.edu (N.D. Bastian).

https://doi.org/10.1016/j.eswa.2023.119734
Received 1 November 2022; Received in revised form 15 February 2023; Accepted 20 February 2023
Available online 28 February 2023

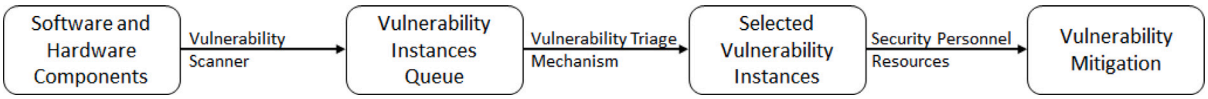

Fig. 1. Cyber vulnerability management process.

iteration(s). We provide a detailed literature review on the various triage mechanisms used for vulnerability selection and mitigation in Section 2.

1.2. Challenges in current approaches

The current approaches for vulnerability management, including methods employed at the CSOCs and proposed in recently published literature, use rule-based mechanisms or static (one-time) optimization models (Farris, Shah, Cybenko, Ganesan, & Jajodia, 2018; Hore, Moomtaheen, Shah, & Ou, 2022; Shah, Farris, Ganesan, & Jajodia, 2019) to prioritize the selection of vulnerabilities for mitigation, given the number of resources available at a particular time step (e.g., a week). The shortcomings in the current approaches are as follows.

- The vulnerability selection processes do not include a comprehensive list of factors associated with the host machine and the respective organizational environment to determine the true priority of a vulnerability instance found in a scan report. For instance, a CSOC security team performs many functions, which include intrusion detection system (IDS) alert management along with vulnerability management. An IDS alert log can identify host machines with possible intrusion attempts and integrating this information, along with other factors such as the CVSS severity rating, into prioritizing vulnerability instances found on such machines can help better protect against potential attacks.
- Recently proposed optimization models (Farris et al., 2018; Hore et al., 2022; Shah et al., 2019) have focused on selecting vulnerability instances from dense reports that maximize the cumulative vulnerability utility or exposure score. Such approaches make a biased decision of selecting a vulnerability instance based on the time it takes to patch or mitigate it. Hence, these methods will select a larger number of less important vulnerabilities if their mitigation time is considerably low when compared to an important vulnerability with a significantly higher mitigation time.
- The current approaches assume a deterministic environment for solving this problem (Farris et al., 2018; Hore et al., 2022; Shah et al., 2019). The number and type of vulnerability arrivals are assumed to be known and distributed uniformly across the time horizon. They do not consider the stochasticity in vulnerability arrivals and consider a pre-determined (often an equal) number of resources distributed across all the individual decision-making time steps to prioritize the selection of vulnerabilities for mitigation. For instance, if an organization has R number of resources available for T number of sequential time steps, then the current methods assign an equal distribution of the resources (i.e., $\frac{R}{T}$) at each time step (e.g., a week) of a given period (e.g., a month). In this example, the CSOC commits to use $\frac{R}{T}$ number of resources even during a time step where the potential threat associated with vulnerabilities present in the system is low, which can occur due to the arrival of very few or less severe vulnerabilities. Such an inflexible approach will prevent the security team from reacting appropriately in an adverse event (medium or high vulnerability arrival) that occurs in a subsequent time step due to the wastage of resources in the previous step. Hence, a deterministic, one-time decision-making model will produce a sub-optimal mitigation strategy in real-world stochastic conditions.

1.3. Research objective and approach

The cyber vulnerability management process aims to strengthen an organization's security posture in an infinite time horizon, requiring sequential decision-making. This resource-constrained process must be made robust to the stochasticity in vulnerability arrivals. Hence, it is imperative that (i) the number of resources to be allocated at each time step (iteration) is optimized and (ii) the important vulnerabilities are identified and prioritized for mitigation from the dense report (queue), given the optimized allocation of resources. Our research objective is to fill the current gap in the cyber vulnerability management process by proposing a novel artificial intelligence (AI) enabled framework powered by a deep reinforcement learning (DRL) agent and an integer programming method for effective vulnerability triage and mitigation. First, our proposed framework dynamically allocates resources for a given system state by considering future uncertainties in a resource-constrained environment. Next, a mathematical model for vulnerability prioritization and selection is formulated and solved to select a set of important vulnerabilities prioritized for mitigation based on the resource allocation decision. We provide a detailed description of our proposed method for cyber vulnerability management in Section 3.

1.4. Contributions of the research study

The main contributions of the study are as follows. First, we developed a novel dynamic cyber vulnerability triage framework, Deep VULMAN, designed to combat the uncertainty in the vulnerability management process and select the most important vulnerability instances for mitigation from a dense list of vulnerabilities identified in the network. Unlike other methods in recent literature, we pose the problem as a sequential decision-making problem and segregate the vulnerability management process in our proposed framework into two parts. (i) We determine the near-optimal amount of resources required for mitigation, given the observed system state. (ii) We determine the optimal set of prioritized vulnerability instances for mitigation. Second, we developed a DRL agent based on a policy gradient approach that learns to make near-optimal resource allocation decisions under stochastic vulnerability arrivals. The agent continuously interacts with a simulated CSOC operations environment built using real-world vulnerability data. The agent gets feedback from a novel reward signal engineered from (i) the mitigation of important vulnerabilities and (ii) the number of resources utilized at each time step. Third, we developed a combinatorial mathematical model using an integer programming method for vulnerability prioritization and selection for mitigation with the allocated resource decision from the DRL agent. Unlike the current methods in the literature, we present a unique formulation that generates an optimal set of prioritized vulnerability instances for mitigation, with the maximum average cumulative attribute score among all the vulnerability instances. Fourth, to the best of our knowledge, this study is the first to propose a framework that integrates IDS alert information to vulnerability data to improve the vulnerability management process at a CSOC. This approach is a major step towards building a robust defense system against adversaries. Our experiment results demonstrated that with this added information from the alert logs, through prioritized vulnerability instances, we were able to find machines that had very old or expired versions of software, making them easier targets for adversaries. Finally, we provided valuable insights obtained using our proposed framework by comparing our results with recent vulnerability prioritization and selection methods from the literature.

Our experimental results using real-world vulnerability data show that our approach is more efficient and effective in selecting important organization-specific vulnerabilities than the other methods.

The paper is organized as follows. Section 2 presents the related literature. Section 3 presents the proposed Deep VULMAN framework, which consists of the CSOC operations simulation environment and the AI-enabled decision-support component that recommends near-optimal decisions for vulnerability management. The DRL resource allocation and the vulnerability prioritization and selection model formulations and algorithms are presented in this section. Section 4 presents the numerical experiments performed using both simulated and real-world vulnerability scan data. Section 5 presents the experiment results and comparisons with recent methods from literature. Lastly, in Section 6, we present conclusions and meta-principles obtained from this study, followed by potential future research directions.

2. Related literature

We have organized the literature review by dividing the related literature into two topics: (i) vulnerability scoring systems and triage methods, and (ii) DRL approaches in solving sequential decision-making problems.

2.1. Vulnerability scoring systems and triage methods

To gauge the severity or threat of a vulnerability, it is important to have a mechanism for scoring the attributes or impacts of the vulnerability. In 2006, Mell, Scarfone, and Romanosky (2006) proposed the common vulnerability scoring system (CVSS) to provide a base score to quantify the vulnerability severity. Later, in 2007, the same authors proposed CVSS version 2 to cover the shortcomings of CVSS version 1 by reducing inconsistencies, providing additional granularity, and increasing the capability to reflect a wide variety of vulnerabilities (Mell et al., 2007). The CVSS framework is managed by the Forum of Incident Response and Security Teams (FIRST), and the latest version of CVSS in use today is version 3.1 (FIRST.org, 2020). The CVSS metric consists of eight base metrics, three temporal metrics, and four environmental metrics (FIRST.org, 2020). However, the computation of environmental metrics is complicated and not well proven (Gallon, 2010). The NVD omits the temporal and environmental metrics and considers only the base metrics when calculating the CVSS severity of reported vulnerabilities (Fruhwirth & Mannisto, 2009).

CVSS base metric group is a common choice of application among most organizations to gauge the severity of the vulnerabilities present in their network. However, anecdotal and literary evidences suggest that the CVSS base score alone is not sufficient to measure the impact of a vulnerability in a particular organization due to the absence of organizational context (Farris et al., 2018; Fruhwirth & Mannisto, 2009; Holm, Ekstedt, & Andersson, 2012; Holm, Sommestad, Almroth, & Persson, 2011). There have been many contributions from researchers to bridge this gap. Some of the important contributions are by McQueen, McQueen, Boyer, and Chaffin (2009), in which the authors proposed two metrics Median Active Vulnerabilities (MAV) and Vulnerability-Free Days (VFD) based on the report time of the vulnerability and time when the patch is issued by the vendor; Allodi and Massacci (2014) considered black-market exploit data to boost the statistical significance of the indication pertaining to the true severity of a vulnerability; Farris et al. (2018) proposed two performance metrics: Total Vulnerability Exposure (TVE) that scores the density of unmitigated vulnerabilities per month and Time-to-Vulnerability Remediation (TVR) based on the maximum amount of time (in months) an organization is willing to tolerate the presence of a certain vulnerability in their system; and Hore et al. (2022) presented a novel Vulnerability Priority Scoring System (VPSS) that takes into account the context of the vulnerability along with the CVSS score by considering relevant host machine information (positional significance of the host machine, level of importance of the host machine, and protection level of the host machine).

The vulnerability management environment at a CSOC is highly resource constrained. The amount of resources needed to mitigate all the vulnerabilities present in an organization is not adequate. Hence, developing a good scoring system only solves a part of the problem. To use the available resources in the best possible manner, it is imperative to develop triaging solutions to mitigate the most severe vulnerabilities first. Some of the research studies that addresses this part of the problem are as follows. Shah et al. (2019) formulated the triage problem as an integer program and presented a comparison between individual attribute (severity, persistence, age) and multiple attribute value optimization; Farris et al. (2018) presented a mixed-integer programming formulation and developed a goal programming based solution approach, in which the objective was to minimize the sum of weighted deviations from a target personnel-hours, target TVR and target TVE; Hore et al. (2022) proposed a two-step model, in which the first step consisted of an integer programming model with the objective of minimizing the VPSS score and the second step consisted of a mixed-integer programming approach for assigning the selected vulnerabilities from the first step to the most suitable analysts; Cavusoglu, Cavusoglu, and Zhang (2008) developed a game theoretic approach to patch management that takes into account the cost–benefit analysis of patching; Dondo (2008) proposed a fuzzy risk analysis approach to rank the vulnerabilities available for selection; and Sharma, Sibal, and Sabharwal (2021) proposed a text analytics based vulnerability prioritization with the help of word embedding and convolutional neural networks.

The aforementioned research studies contribute effective ways of solving the vulnerability triage problem in a deterministic environment as a one-time decision-making problem, in which the number and type of vulnerability arrivals are considered to be known and are assumed to be uniformly distributed across the time horizon. However, in reality, the vulnerability arrival process is highly stochastic. Hence, it is unlikely for a deterministic model to perform optimally in a stochastic environment. There exist a gap in the literature that demands the development of a model that is robust to the uncertainty of both number and types of vulnerability arrivals. Also, accumulating relevant attributes for a reliable vulnerability scoring system is a daunting task. There is a critical need to add information from other functions of the CSOC such as IDS alert management to improve the security posture of an organization.

2.2. Deep Reinforcement Learning (DRL) approaches

DRL is one of the most promising solution methods for obtaining near-optimal policies under stochastic conditions. DRL was first applied by Mnih et al. in 2013 to successfully learn a control policy from sensory inputs with high dimensions (Mnih, Kavukcuoglu et al., 2013). Today, DRL has been used in various application domains such as autonomous vehicles, stock trading, robotics, cyber-security, and marketing, among others (Bogyrbayeva, Jang, Shah, Jang, & Kwon, 2021; Kirtas, Tsampazis, Passalis, & Tefas, 2020; Liang, 2020). The model-free DRL methods in published literature can be broadly classified in two parts: value-based and policy-based. In value-based DRL approaches, we try to estimate the Q-value or a state–action pair by employing a deep neural network estimator. Policy based methods aim to directly learn the stochastic or deterministic policies, where the action is generated by sampling from the policy. Mnih et al. proposed a novel method, Deep Q Learning (DQN), which is a value-based method with superior performance demonstrated on Atari 2600 games. Some of the notable advancements made in the area of value-based DRL methods include the works by: Van hasselt et al. who proposed DRL with double q-learning (DDQN) to overcome the overestimation suffered by DQN (Van Hasselt, Guez, & Silver, 2016; Wang et al., 2016), who proposed the dueling network architectures for DRL with two identical

but separate neural network estimators for estimating the state value function and action advantage function, among others.

One of the popular advancements in policy-based methods include the work by Mnih et al. who presented asynchronous methods for DRL with parallel actor learners, asynchronous advantage actor critic (A3C), and outperformed others on Atari 2600 games (Mnih, Badia et al., 2016). Vanilla policy gradient algorithms generally suffer from high variance, poor sample efficiency, and slow convergence. Schulman, Levine et al. (2015) presented Trust Region Policy Optimization (TRPO) that limits the policy update with a certain KL-divergence constraint and also guarantees monotonic improvement. In 2017 Schulman, Wolski, Dhariwal, Radford, and Klimov (2017) proposed Proximal policy Optimization (PPO) that has all the advantages of TRPO, and in addition it is simpler, faster, and more sample efficient. PPO uses a clipped surrogate objective function that prevents large changes in the policy. The clipped surrogate objective is also a light weight replacement of the KL-divergence constraint in TRPO. Due to its simplicity, sample efficiency, and robustness to hyper parameter tuning, PPO is a promising approach to solving dynamic sequential decision-making problems.

An efficient and effective cyber vulnerability management process can help an organization bolster the security of its computer network systems. It is critical to continuously improve this process, consisting of vulnerability triage and allocation to an appropriate number of resources for mitigation. Most of the work in the literature focuses on formulating one-time (static) strategies for selecting vulnerabilities from dense vulnerability reports by considering a fixed amount of resource availability and without taking future vulnerability arrivals into account. To our knowledge, no research has addressed this gap by posing the cyber vulnerability management problem as a sequential decision-making problem under the stochasticity of vulnerability arrivals and resource fluctuations. This paper focuses on strengthening the security posture of the CSOCs by generating robust vulnerability management policies for real-world stochastic environments. Next, we present the proposed framework for dynamic vulnerability management.

3. Deep reinforcement learning-enabled cyber vulnerability management (deep VULMAN) framework

We propose the development of a sequential decision-making framework that provides a dynamic resource allocation strategy along with an optimal selection of vulnerabilities that are prioritized for mitigation. Fig. 2 shows a schematic representation of the Deep VULMAN framework. The framework consists of two key components: (i) a CSOC operations environment, where relevant computer and network data are collected and aggregated using various software applications, and (ii) a decision-support component, in which (a) a DRL agent is trained using a policy gradient algorithm to make near-optimal resource allocation decisions under uncertainty and (b) an integer programming model is developed to generate the set of vulnerabilities, which are prioritized for mitigation with the amount of resources allocated by the DRL agent. We first describe the CSOC operations environment, in which we propose a simulator to overcome the data insufficiency issues for training the DRL agent, followed by the decision-support component.

3.1. CSOC operations environment

A major challenge for cybersecurity researchers is obtaining an extensive data set for a research study. Very few studies in published literature, such as Farris, McNamara, Goldstein, and Cybenko (2016), Shah et al. (2019) and Xu, Schweitzer, Bateman, and Xu (2018), have investigated the process of cyber-incident or vulnerability emergence using historical data. However, these have been small and private data sets. Data sets are unavailable for research due to a lack of complete information in a cyber environment or for confidentiality reasons. Authors in Haldar and Mishra (2017) and Kuypers and Paté-Cornell (2016) studied cyber-incident data for large cyber breaches and found Poisson distribution to be the best fit for describing the arrivals in these data sets. The DRL agent must interact with an environment that closely resembles the real CSOC operations to learn the best policies that can be implemented in real-world conditions. Hence, to overcome the challenges of having insufficient data to properly train a DRL agent and learning in a slow-moving real-world environment (Dulac-Arnold, Mankowitz, & Hester, 2019), we built a simulator from the large amount of real data that we collected by working with a CSOC. We developed a discrete event simulation (DES) algorithm with fixed-increment time progression to model the vulnerability management process at a CSOC. The DRL agent (explained in the next section) continuously interacts with this simulated environment to learn the goodness of its actions. It is to be noted that the agent is unaware of the inner dynamics of the simulator. Algorithm 1 presents the simulation algorithm. The inputs to the algorithm are the various vulnerability scan reports and other relevant network-related information obtained from applications such as Nessus, Lansweeper, and IDS. The stochasticity in the vulnerability arrival process is realized in the simulator by following a randomly generated vulnerability arrival pattern at the beginning of each planning horizon (episode). The arrivals are generated using a Poisson distribution with varying mean obtained from the historical data. Vulnerability instances with varying characteristics are randomly sampled from the historical data sets at each time step in an episode, along with the collection of the related host machine data.

In a real-world implementation, training a DRL agent requires a significant amount of data, generated through repeated interactions between the agent and its environment. This would require a CSOC to build a simulator mimicking their real-world operations to enable offline training of the agent. There are inherent assumptions made regarding the number and types of vulnerabilities that will be encountered, which could lead to biased learning if the scenarios presented in the simulator do not accurately reflect the CSOC's actual state. To overcome this challenge, it is crucial to incorporate historical data from the CSOC to obtain a true representation of vulnerability arrival patterns. This information should be used in the simulator to expose the DRL agent to the real-world conditions observed in the respective CSOC.

The vulnerability instances, the respective host machine information, and the number of resources available are then passed on to the decision-support component (see Fig. 2) as the state of the system at the given time step. The action pertaining to this system state is then taken as input by the simulated environment from the decision-support component. The simulation algorithm executes this action, containing the vulnerability set selected for mitigation. A scalar reward is computed in the simulator, which consists of two terms, one related to the mitigation of important vulnerabilities and another to the number of resources utilized. The action selection and reward function details are presented in the next section. The cumulative time taken to mitigate the selected vulnerabilities is then deducted from the total time available at the beginning of the time step (e.g., the start of the week). The environment is then stepped forward to the next time step (the start of next week) with the remaining resources. A new set of vulnerability instances is then generated, and this process continues for the entire episode (e.g., a month). The simulator adds the new set of vulnerabilities (arrivals) to the unmitigated vulnerabilities from the previous time step. Next, we describe the decision-support component of the proposed framework.

3.2. Decision support for vulnerability management

The objective of this research is to identify and prioritize important vulnerabilities for mitigation under stochasticity in vulnerability arrivals in a resource-constrained system. It is to be noted that the decision-making problem can be broken down into obtaining two decisions: (i) determining the near-optimal resources to be allocated and (ii) determining the set of vulnerability instances for mitigation

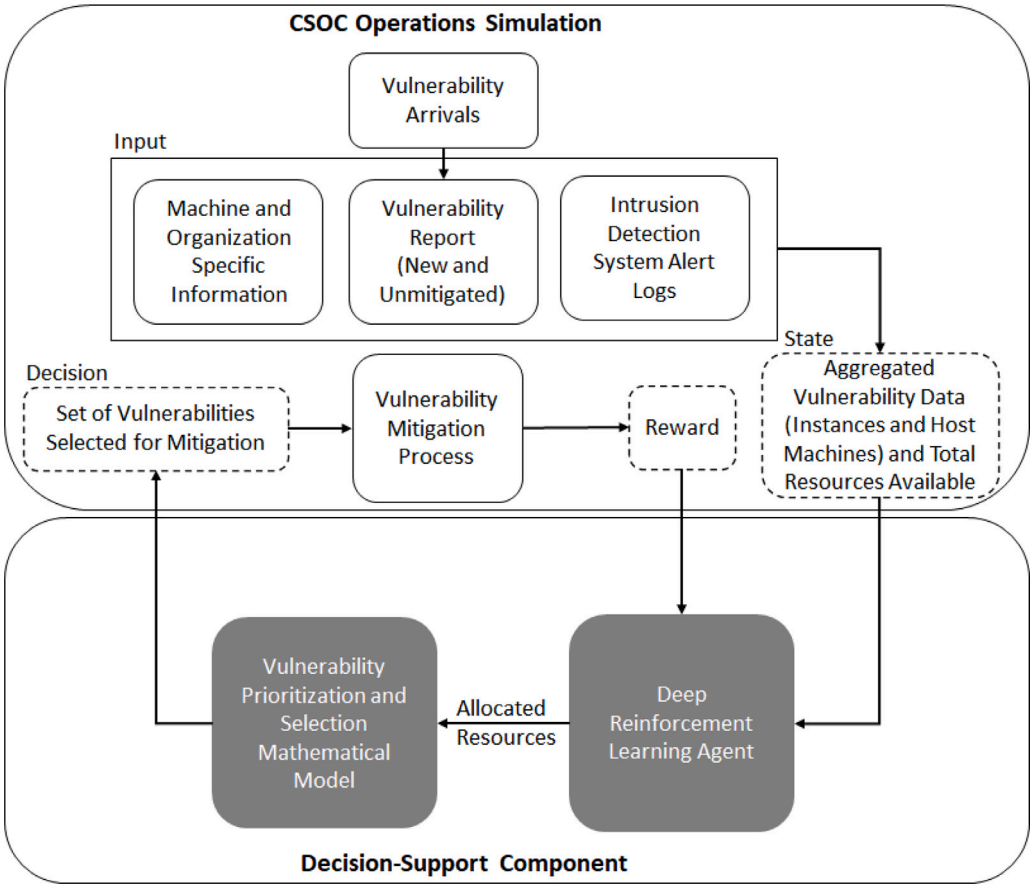

Fig. 2. Deep VULMAN framework for cyber vulnerability management.

Algorithm 1: CSOC operations simulation algorithm

Input: Nessus vulnerability report, Lansweeper report, IDS data, weights for reward terms, w_1 and w_2, total resources available for the episode, R.
Output: System state at time $t+1$, s_{t+1}, reward at time t, r_t.
Aggregate the input information to create a data set with vulnerability instance and host machine information.
/* Initiate the CSOC environment simulation */
repeat
  Create a set of vulnerability arrival patterns, P
  for *number of episodes,e <= E:* **do**
    Randomly choose an arrival pattern from the set, P
    Generate vulnerability instance arrivals for the first time step
    for *number of time step,t <= T* **do**
      Obtain action, a_t from Algorithm 2
      Implement action, a_t
      Calculate reward, r_t^1 for mitigating vulnerabilities
      Calculate reward, r_t^2 for resource utilization
      Calculate the cumulative reward, $r_t = w_1 * r_t^1 + w_2 * r_t^2$
      Update the resources available for $t+1$
      if *not terminal state* $(t < T)$ **then**
        Increment t by 1
        Simulate vulnerability arrivals for time $t+1$
        Add new arrivals to unmitigated vulnerabilities
      end
      else
        $e = e + 1$
      end
    end
  end
until *Stopping criteria /* Maximum number of episodes E is reached*;
return *System state at time t+1,* s_{t+1}, *Reward at time t,* r_t

given these resources, which reduces the vulnerability exposure of the organization in the long run. The former decision of allocating the appropriate amount of resources is affected by the stochasticity in the environment and the CSOC can strengthen their security with a dynamic resource allocation strategy. Once the decision on the amount of resources allocated is made, the mathematical model can be invoked to optimally select the set of important vulnerabilities for mitigation. The decision-support component consists of (i) a DRL agent that generates the resource allocation strategy and (ii) a mathematical model, which is formulated and solved using integer programming, that optimally selects important vulnerability instances from the dense report for prioritized mitigation. In this section, first, we describe the DRL problem formulation for optimizing the resource allocation strategy, followed by the DRL algorithm and the formulation of the mathematical model, which outputs the vulnerability selection decision.

3.2.1. DRL formulation

The problem of making sequential decisions for resource allocation to mitigate important vulnerabilities and thereby reducing the vulnerability exposure and strengthening the security posture of an organization in the long run can be formulated as a Markov decision process (MDP). The key elements of the MDP formulation are as follows:

- **State**, s_t, represents the information that is visible to the agent at time t, which consists of the vulnerability instances, their respective attributes, and the total amount of resources available. The state space is $N * (I+1)$ dimensional, where I is the number of attributes and N is set to a value greater than or equal to the maximum number of vulnerabilities historically found in the vulnerability scan reports. The system state is shown in a dotted box in Fig. 2, which provides the DRL agent with the information needed to make the resource allocation decision for vulnerability

selection. We use the concept of zero padding to fill empty rows (i.e, last N - J rows, where J is the number of vulnerabilities found in that scan) with zeros (Lin, Wang, Olkin, & Held, 2020).

- **Action**, a_t, represents the control. The action is the amount of resources to be allocated at time t, given a state, s_t. The action space is continuous for this problem to maintain tractability of obtaining decisions.
- **State transition function** determines the probability with which a system will transition from state s_t to s_{t+1} under action a_t. The state transition probabilities for this problem are unknown and the possible number of state transitions are very high (state space explosion). Hence, it is infeasible to determine the state transition probabilities. The simulator will provide the state transition to s_{t+1} under action a_t.
- **Reward**, r_t, is a measure of the goodness of an action, a_t, taken in a given state, s_t, at time t. The agent's goal is to maximize the long-term cumulative reward. Hence, setting up the reward signal is critical to train the agent to achieve the research objective. In this research, we engineer a novel reward function, which consists of two weighted terms. The agent receives this reward, as shown in the dotted box in Fig. 2, from: (i) the mitigation of important vulnerabilities (r^1) and (ii) the number of resources utilized (r^2). The reward function, at time t, is given by Eq. (1) where w_1 and w_2 are weights associated with the reward terms and whose sum must be equal to 1.

$$r_t = w_1 * r_t^1 + w_2 * r_t^2 \tag{1}$$

The importance of a vulnerability instance is determined by taking into consideration the following attributes: asset criticality, level of protection, organizational relevance of the host machine, the CVSS severity of the vulnerability instance, and if the host machine has been identified in any IDS alerts. These attributes are obtained using various applications from the organization's computer and network systems. Categorical attributes are transformed into numerical values based on certain rules from literature. We use the same scheme, as in Farris et al. (2018), Hore et al. (2022) and Shah, Ganesan, Jajodia, and Cam (2018), to identify various categories for each attribute and assign normalized numerical values. From our conversations with the CSOC and subject matter experts, we found that all these factors are considered equally important. There is a positive reward for selecting vulnerabilities, which is calculated by taking the average of all the attribute values of the selected vulnerabilities. If there are J number of selected vulnerabilities and v_{ij} represents the value of attribute i of the vulnerability instance j, then the positive reward can be calculated as:

$$r_t^1 = \{\frac{\sum_{j=1}^{J}\sum_{i=1}^{I} v_{ij}}{I * J} : V_t \subseteq U_t, 0 \leq r_t^1 \leq 1\}, \tag{2}$$

$$U_t \in \mathbb{R}^{N*(I+1)},$$

$$V_t \in \mathbb{R}^{J*(I+1)},$$

$$J \leq N,$$

where U_t is the set of all vulnerabilities present in the system at time t and V_t is the selected subset of vulnerability instances from U_t. Since the CSOC operations environment is resource constrained, there exists a trade-off between the number of vulnerabilities selected for mitigation and the number of resources that remain available for vulnerability selection in the next time step. Hence, we assign a small cost to the utilization of the resources in this formulation. For the J number of vulnerability instances selected for mitigation with S_j representing the time required to mitigate vulnerability j and C representing the cost per unit resource utilized, then the resource utilization penalty is calculated as:

$$r_t^2 = \{-C\sum_{j=1}^{J} S_j : C = \frac{1}{R_{total}}, 0 \leq r_t^2 \leq 1, S_j \in \mathbb{R}^+\}, \tag{3}$$

where R_{total} is the total number of resources available at the beginning of an episode. The S_j values are calculated using the approach proposed in Farris et al. (2018). It is important to note that the reward signal depends on both the allocation of the appropriate number of resources and the mitigation of important vulnerabilities. An inherent trade-off exists among the two reward terms, which produces this novel reward signal to guide the DRL agent in its decision-making.

The large state space and continuous action space makes this problem infeasible to solve using conventional reinforcement learning approaches. To overcome the issue of not being able to calculate and store the action–value (or Q value) for all possible state–action pairs due to state space explosion, we propose a deep neural network-based learning model with a policy gradient algorithm for efficiently solving this problem (Silver et al., 2014). The deep neural networks serve as an approximation for the complex, nonlinear decision-making process in the real-world, which map the inputs to outputs. The model takes into account the list of vulnerabilities and their attributes to estimate the resources to allocate. Next, we describe the DRL algorithm.

3.2.2. DRL algorithm

Vanilla policy gradient algorithms have disadvantages such as poor data efficiency, lack of robustness, and are often subjected to large changes in policies resulting in unstable learning. Hence, we propose the proximal policy optimization (PPO) approach (Schulman et al., 2017) for solving this problem, which is an on-policy algorithm that overcomes the aforementioned challenges. PPO ensures a smoother learning of the policies with the objective clipping feature. Additionally, PPO is easy to implement and tune, and provides better sample efficiency. Algorithm 2 shows the PPO approach proposed for solving the research problem. Our aim is to train a stochastic policy in an on-policy manner in this algorithm. We sample actions from a multivariate normal distribution with mean μ and covariance matrix Σ.

In on-policy learning, the exploration of new policies is achieved by sampling actions from the latest update of the stochastic policy. The amount of randomness in policy selection decreases over the course of training, leading to locally optimum policies due to insufficient exploration. However, it is essential that a DRL agent is able to explore sufficiently to avoid getting stuck in a local minima. Hence, to address this issue: (i) a standard deviation is added to the action computed by the policy network, and (ii) an entropy regularization term is added to the objective function (Ahmed, Le Roux, Norouzi, & Schuurmans, 2019). The value of added standard deviation is larger at the beginning of the training process and it decreases over the course of the training phase with a decay rate. For a discrete random variable x with probability mass function $P(x)$, the entropy, $H(x)$ can be calculated as:

$$H(X) = -\sum_{x \in X} P(x) log P(x) \tag{4}$$

In this paper, we use generalized advantage estimation (GAE) to compute the advantages used in the value function. Eq. (5) shows the GAE estimator, GAE (γ, λ), as the exponential-weighted average of k-steps (Schulman, Moritz et al., 2015), where γ is the discount factor and λ is the exponential weight discount. Both these values range between 0 and 1. δ_t^V is the temporal difference (TD) residual of an approximated value function V with discount factor γ (Sutton & Barto, 2018). If r_t is the reward at time t then δ_t^V can be expressed as, $\delta_t^V = r_t + \gamma V(s_{t+1}) + V(s_t)$.

$$\tilde{A}_t^{GAE(\gamma,\lambda)} = \sum_{t=0}^{\infty} (\gamma\lambda)^l \delta_{t+1}^V \tag{5}$$

The salient feature of the PPO approach is the clipped surrogate objective function, which is a modification over the objective function proposed in conservative policy iteration (Kakade & Langford, 2002). Vanilla policy gradient algorithms often result in destructively large policy updates, which cause the learning to be unstable; the clipped surrogate objective function ensures a small policy update at each epoch with the selection of a small value for ϵ, which defines the maximum permissible change in the policy. Eq. (6) shows the clipped surrogate objective, where ϵ is a hyperparameter that determines the threshold of clipping.

$$L^{CLIP}(\theta) = \tilde{\mathbb{E}}[min\frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{old}}(a_t \mid s_t)}\tilde{A}_t, clip(\frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{old}}(a_t \mid s_t)}, 1-\epsilon, 1+\epsilon)\tilde{A}_t] \quad (6)$$

We use a neural network architecture that shares the weight parameters between the policy and the value function estimators. Hence, we take into consideration the value function error term along with the clipped objective. Additionally, an entropy regularization term is added to ensure sufficient exploration (Mnih, Badia et al., 2016). Eq. (7) shows the consolidated objective function, in which c_1 and c_2 are coefficients for the value function error term and the entropy, respectively.

$$L_t^{CLIP+VF+H}(\theta) = \tilde{\mathbb{E}}[L_t^{CLIP}(\theta) - c_1 L_t^{VF}(\theta) + c_2 H[\pi_\theta](s_t)] \quad (7)$$

Next, we update the policy network parameters, θ_k using stochastic gradient ascent and the value function parameters, ϕ_k using stochastic gradient descent, as shown in Algorithm 2. The learning phase continues until a maximum number of episodes, E, is reached. The number E can greatly vary depending on the state and action space dimensions and complexity of the environment. The approximate function represented by the policy network with the final learned weights estimates the required resources (output), given the vulnerability instances and their attributes (input) in the system. The number of resources, a_t allocated for mitigation at any given time t, obtained from this near-optimal policy is then passed on to the vulnerability prioritization and selection mathematical model, which is described next.

3.2.3. Vulnerability prioritization and selection model

The prioritization and selection of vulnerability instances is achieved by solving a mathematical model, whose solution provides the set of prioritized vulnerabilities selected for mitigation by the available resources (decision made by the DRL agent). The vulnerability selection problem is posed as a combinatorial optimization problem and solved using integer programming. The static vulnerability prioritization and selection models in Farris et al. (2018), Hore et al. (2022) and Shah et al. (2019) directly maximize the cumulative utility or exposure scores of their respective factors to obtain the sets of prioritized vulnerability instances. Such sets of vulnerabilities may not contain all important vulnerabilities, as their formulations do not maximize the average value of the selected vulnerabilities. For instance, consider five vulnerabilities obtained from a scan report with respective utilities of 7, 2, 2, 3, and 2, and the respective expected mitigation time of 7.5, 2, 2, 2, and 1.5 h. If, at time t, there exists only 7.5 h remaining, then these models will pick vulnerability instances 2, 3, 4, and 5 as this selection set will have a cumulative value of $2 + 2 + 3 + 2 = 9$ units, while the total time required will be 7.5 h. However, it would have been better to pick vulnerability instance 1 as the utility or exposure associated with it is higher than others. This is a major shortcoming of these algorithms, which will miss out on picking important vulnerabilities in such cases. In our proposed formulation, we counter this issue by maximizing the average of the cumulative value of all the attributes across all selected vulnerability instances subject to the total time available for mitigation in any given time-period. In addition, we take into consideration the largest set of attributes associated with any vulnerability and its respective host machine when compared to other methods in published literature. The notations used in the formulation of the mathematical model are defined in Table 1. Below, we present the input parameters, decision variables, objective function, constraints, and the output of the vulnerability selection model.

Algorithm 2: Deep VULMAN learning algorithm

Input: Initiate initial policy parameters, θ_0, value function parameters, ϕ_0, starting standard deviation value, sd_0, $t = 0$, $e = 0$, decay rate, δ, and replay buffer, B.
Output: Near-optimal vulnerability management policy (near-optimal policy network parameters, θ).
repeat
  for *number of episodes,e <= E:* **do**
    Obtain an initial state, s_0 from Alg. 1
    for *number of time steps,t <= T* **do**
      Use policy network with weights θ_t to obtain action mean, $\bar{a_t}$ using policy π_{θ_t}
      Use standard deviation, sd_e to create a covariance matrix, Σ_e
      Sample action, a_t from a multivariate normal distribution with mean,$\bar{a_t}$ and covariance, Σ_e
      /* Initiate a solution search using an integer programming solver */
      repeat
        for *a set of* $z_j = 1$, *check for feasibility:* **do**
          $\sum_{j=1}^{J} S_j * z_j \leq a_t$ /* Resource availability constraint */
        end
        if *Feasible* **then**
          Calculate $y = \frac{\sum_{j=1}^{J}\sum_{i=1}^{I} v_{ij} * z_j}{\sum_{j=1}^{J} z_j}$
        end
      until *Stopping criteria /* Optimal value for y is found */*;
      Obtain s_{t+1}, r_t from Alg. 1 based on decision, $z_j \forall j$
      Add the trajectory, (s_t, a_t, r_t, s_{t+1}) to B
      Compute rewards-to-go, G_t
      Decay standard deviation with a decay rate, δ
      Compute advantage estimates, $\tilde{A}_t$ using GAE based on the current value function V_t with network weights ϕ_t
      Compute the PPO-Clip objective function:
      $L_t^{CLIP+VF+Entropy}(\theta) = \mathbb{E}_{\approx}[L_t^{CLIP}(\theta) - c_1 L_t^{VF}(\theta) + c_2 * Entropy]$
      Update the policy by maximizing the PPO-Clip objective via stochastic gradient ascent.
      $\theta_{k+1} = argmax_\theta \frac{1}{l*T}$
      $\sum_{\tau \in R}\sum_{t=0}^{T} min(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)}) A^{\pi\theta_k}(s_t, a_t), g(\epsilon, A^{\pi\theta_k}(s_t, a_t))$
      The parameters for the value function network by minimizing the mse loss by stochastic gradient descent, ϕ_{θ_k} is updated using:
      $\phi_{\theta_{k+1}} = argmin_\phi \frac{1}{l*T}\sum_{\tau \in R}\sum_{t=0}^{T}(V_\phi(s_t) - G_t)^2$
    end
  end
until *Stopping criteria /* Maximum number of episodes, E, is reached*;
return *Near-optimal vulnerability management policy (near-optimal policy network parameters, θ)*

Algorithm 3: Deep VULMAN implementation algorithm

Input: Near-optimal vulnerability management policy from Alg. 2, state s_t from real-world data set.
Output: Set of vulnerabilities selected for mitigation.
Obtain action a_t for state s_t according to the learned policy from Alg. 2
/* Initiate a solution search using an integer programming solver */
repeat
  for *a set of* $z_j = 1$, *check for feasibility:* **do**
    $\sum_{j=1}^{J} S_j * z_j \leq a_t$ /* Resource availability constraint */
  end
  if *Feasible* **then**
    Calculate $y = \frac{\sum_{j=1}^{J}\sum_{i=1}^{I} v_{ij} * z_j}{\sum_{j=1}^{J} z_j}$
  end
until *Stopping criteria /* Optimal value for y is found */*;
return $z_j \ \forall j$.

Input parameters:

- The attribute scores for all vulnerability instances, v_{ij} $\forall i, j$.
- Expected time taken to mitigate a vulnerability instance j, S_j.
- Total number of vulnerability instances in the scan report, J.
- Total resources available at time t (action from the DRL agent), a_t.

Table 1
Definitions of notations.

Notation	Definition
Indices:	
j	Vulnerability instance index
i	Vulnerability attribute index
Inputs:	
J	Total number of vulnerability instances
I	Total number of vulnerability attributes
S_j	Expected mitigation time for vulnerability j
a_t	Total resources (in hours) available at time t
v_{ij}	Value of attribute i for vulnerability j
Variables:	
z_j (Binary)	Selection of vulnerability instance j

Decision variables:

- $z_j = 1$ if vulnerability instance j is selected, and 0 otherwise.

Objective function: The objective of the model is to select the set of vulnerability instances prioritized for mitigation that maximizes the average of the cumulative value of the attribute scores across all selected vulnerability instances. The objective function is given by:

$$y = Max \quad \frac{\sum_{j=1}^{J}\sum_{i=1}^{I} v_{ij} * z_j}{\sum_{j=1}^{J} z_j} \tag{8}$$

Constraint: The constraint for the model is the availability of resource time, a_t, at any given time t, which is obtained from the DRL agent. The constraint for the total time taken to mitigate the selected vulnerability instances not being higher than the total resource time available at time t is expressed as:

$$\sum_{j=1}^{J} S_j * z_j \leq a_t \tag{9}$$

Output: The output of the vulnerability prioritization and selection model is the set of prioritized vulnerability instances selected for mitigation. This decision is then communicated to the CSOC operations environment, as shown in Fig. 2.

At the end of the learning phase (as shown in Algorithm 2), the nonlinear approximation function with the final learned weights of the policy network is obtained. This function near-optimally maps the list of vulnerability instances and their attributes to the required resources to allocate for mitigation, representing the near-optimal vulnerability management policy. Algorithm 3 shows the learned version of the Deep VULMAN method. Given a new state, s_t, obtained at time t from previously unseen data (simulated or real-world data set), the algorithm, using the learned approximation function, outputs the set of vulnerability instances selected for mitigation, i.e., $z_j \ \forall j$. Next, we present the numerical experiments and analysis of results.

4. Numerical experiments

In this section, we first describe the data collection process and the simulation environment setup for the CSOC vulnerability management operations, followed by the experimental details of the training and testing phases of the DRL algorithm. We worked closely with a CSOC to collect the vulnerability data and other relevant computer network information. Our conversations with the security analysts helped us determine various parameter values that were used in setting up the environment, and training and testing the proposed Deep VULMAN framework.

4.1. Data collection and simulation environment for CSOC operations

We developed a simulator from the real-world data that we collected by working with a CSOC. We used two applications: Tenable's Nessus vulnerability scanner and Lansweeper to collect the vulnerability data. The data set contained 98,842 unique vulnerability instances, which were found historically in the network. We also collected relevant host machine data and alert data generated by the IDS. The Lansweeper report contained information about the host machines in the network, which included the software versions of the operating system and SQL server, among others. In this research study, we also integrated information from the IDS alert logs to obtain the intrusion status of the host machine. If a host machine with the reported vulnerability was identified in the IDS alert log for the respective time-period (say, between time $t-1$ and t), then this information was recorded. The intrusion status attribute of the vulnerability instance was set accordingly. All the machine-specific information for the host machines on which the vulnerabilities were found was added to the consolidated data set. The aggregated data set contained information about the host machine and vulnerability instances such as the host IP, CVE code, description, CVSS value indicating the vulnerability severity rating, importance of the machine in the network, versions of software running on the host machine, and estimated personnel-hours required to mitigate the vulnerability instance (Farris et al., 2018), among other known information.

We applied vulnerability data preprocessing techniques to quantify attributes. We used the same categories and the quantification process as in Farris et al. (2018) and Hore et al. (2022). For completeness, we describe the process as follows. We first surveyed and interviewed the stakeholders (security team). We obtained responses to specific questions pertaining to each attribute. Our questions were categorized into schemes shown in Table 2. Three main categories of responses were recorded: high, medium, and low. Depending on the organization's requirements, this list can be expanded to include more categories, such as critical. Our discussions identified important sub-domains with critical assets and important devices, including database servers, web servers, and important stakeholder machines. We then obtained their respective category ratings from the stakeholders. For example, the machines from these important sub-domains were assigned a critical rating. Similarly, the database, web, and important stakeholder machines were assigned critical, high, and medium levels of relevancy, respectively. The level of protection was measured by evaluating the version of software running on the host machine, including the operating system or specialized systems like SQL. The level of protection decreases as the software version becomes outdated, as vendors may no longer provide support for these versions, leaving them vulnerable to new cyber threats. If the host machine's software version was unsupported by the vendor, it was considered to have low protection and, therefore, was assigned a high relevance. If it was close to being unsupported (for e.g., Windows Server 2012), then it was considered to be at a medium level of relevancy.

Next, we assigned each category a numerical value. The attribute with the highest priority was assigned a numerical value of 1, while the lowest was assigned 0.1. The categories in between the highest and lowest priorities were assigned numerical values using a linear scale. For example, suppose the asset criticality of a particular machine is deemed critical (highest priority). In that case, it will be assigned a value of 1, and if it is low (lowest priority), it will be assigned 0.1. The CVSS severity score for a vulnerability was obtained from the NVD (using a Nessus scanner) and normalized between 0 and 1. If a machine was detected as having a possible intrusion in the IDS alert logs, the intrusion status attribute was assigned a value of 1, otherwise 0. Table 2 shows the attributes and their respective values.

We then used this consolidated data set to build a simulator to closely resemble the vulnerability arrival and mitigation process at a CSOC. We created an agent-based DES that mimics the arrival and

Table 2
Attributes and quantification schemes.

Attribute	Scheme	Category	Value
Asset Criticality	Based on the machine's location among different sub-domains.	Critical High Med Low	1 0.5 0.25 0.1
Protection Level	Based on the version of operating system and SQL.	High Med Low	1 0.5 0.1
Relevance	Based on whether the machine is a web or a database server.	High Med Low	1 0.5 0.1
CVSS Severity	Numerical value from Nessus scan report.	Cont.	0.1–1
Intrusion Status	Machine identified in IDS alert log.	Yes No	1 0

Table 3
Hyper-parameter values used in Algorithm 2.

Hyper-parameter	Value
Number of training episodes (E)	500,000
Number of time steps (T)	4
Number of hidden layers	2
Hidden layer sizes	[64, 64]
Activation for hidden layers	Tanh
Buffer size (B)	512
Batch size	256
Gamma (γ)	0.99
K-epochs	30
Eps-Clip (ϵ)	0.2
Learning rate - Actor	0.0002
Learning rate - Critic	0.001
Value function co-efficient (c_1)	0.5
Entropy co-efficient (c_2)	0.01
Maximum action standard deviation (start) (sd_0)	0.65
Action standard deviation decay rate	0.025
Action standard deviation decay freq.	5000 episodes
Min. action standard deviation (end)	0.01

mitigation process of vulnerability instances in a CSOC. With the help of a simulation model, we generated diverse patterns of new vulnerability arrivals to expose the DRL agent to stochasticity it may find in a real-world environment. From our discussions with the CSOC security personnel and historical evidence, along with the information published in literature (Farris et al., 2018), we modeled the vulnerability instance arrival process using a Poisson distribution and varied the average number of arrivals from 40 to 600 per week (representing a large network). We segregated our arrivals into three different categories, namely, high, medium, and low. Different patterns of arrivals, based on the aforementioned average numbers per week, were simulated for training the DRL agent. Some examples of arrival patterns for four consecutive weeks in a month include, [high, high, low, low], [medium, medium, medium, low], and [low, high, medium, high], among others. Vulnerability instances were randomly sampled from the data set based on the arrival pattern (i.e., using Poisson distribution with the respective average number of arrivals) at each time step (i.e., $t = 1$ week in Algorithm 1) emulating the arrival process in the CSOC. All the information about the vulnerability instances is then passed to the decision-support component (explained in the next sub-section) to obtain the vulnerability instances set selected for mitigation. Upon receiving this information, the simulation algorithm steps forward and the selected vulnerability instances are mitigated utilizing the time assigned to each of them in the consolidated data set. The total mitigation time of the selected vulnerability instances is then deducted from the available resource time from the previous time step and the remaining resource time is carried forward to the next time step. Each week represents as a decision time step in the simulator. Next, we describe the training and testing phases of the proposed Deep VULMAN framework.

4.2. Training phase

The objective of the decision-support component in Fig. 2 is to take near-optimal actions, which involve allocating the security personnel resources and selecting the important vulnerability instances for mitigation, in the wake of stochastic vulnerability arrival process. The DRL agent interacts with the environment to learn the near-optimal policy of allocating the appropriate number of resources that would be required for mitigating the important vulnerabilities. As discussed in the previous section, based on the characteristics of the problem, we developed an advanced policy gradient algorithm using the proximal policy optimization (PPO) approach to efficiently solve this problem.

We conducted our experiments with some of the hyper-parameter values from published literature (Carvalho Melo & Omena Albuquerque Máximo, 2019; Schulman et al., 2017) and tuned the remaining by trial-and-error, which involved running experiments with different sets of values. The PPO approach is known to be more forgiving to sub-optimal initialization of hyper-parameter values. Table 3 shows the hyper-parameters used to train the DRL agent. Algorithm 2 takes an entropy factor in the surrogate loss function, like in the original PPO paper by Schulman et al. (2017). This helps in ensuring sufficient exploration. We conducted the experiments on a machine with 11th Gen Intel Core i7-12700H processor with NVIDIA GeForce RTX 2080 graphics card and 16 GB RAM.

We used a multi-layer perceptron (MLP) model with two hidden layers, each containing 68 perceptrons and Tanh activation functions for the actor and critic networks. We tried various architectures with larger number of hidden layers and perceptrons but did not find any significant improvements in the performance of the DRL agent. We selected the two hidden layer architecture due to its computationally efficiency when compared with the others. The DRL agent took actions using the policy network. There was some standard deviation added to these actions, which started with a value of 0.65 and decayed to 0.01 with a rate of 0.025. The decay rate and decay frequency are problem-specific, and hence we had to tune it with a trial-and-error approach. To avoid getting stuck in a local optimum and encourage exploration, we used 0.01 as the entropy co-efficient value, which was multiplied by the entropy and subtracted from the loss function. The value of the entropy factor was adopted from the literature (Schulman et al., 2017). At each time step, the output of the DRL agent is provided as an input to the vulnerability prioritization and selection mathematical model (Algorithm 2) and the vulnerability instances are selected for mitigation, which are then passed on to the CSOC operations environment. Based on these actions, a scalar reward value is calculated, which is derived from the two terms in the reward function (as shown in Eq. (1)). This reward signal helps the DRL agent to adjust its actions based on the uncertainties experienced in the learning environment. We considered equal values of the weights used for the two reward terms in the reward function (in Eq. (1)) and assigned a value of 10^{-5} to the cost per unit resource utilized (C). We also performed sensitivity analysis by considering a different set of weights for these terms during the training phase. In particular, we assigned a larger weight (0.8) to the first reward term associated with mitigating critical vulnerabilities and a smaller weight (0.2) to the second term associated with conserving resources. It is to be noted that assigning a smaller weight to the first term will contradict the objective of the CSOC vulnerability management team and hence, it was not included in the analysis. Next, we explain the different test arrangements we used to evaluate the performance of the Deep VULMAN approach.

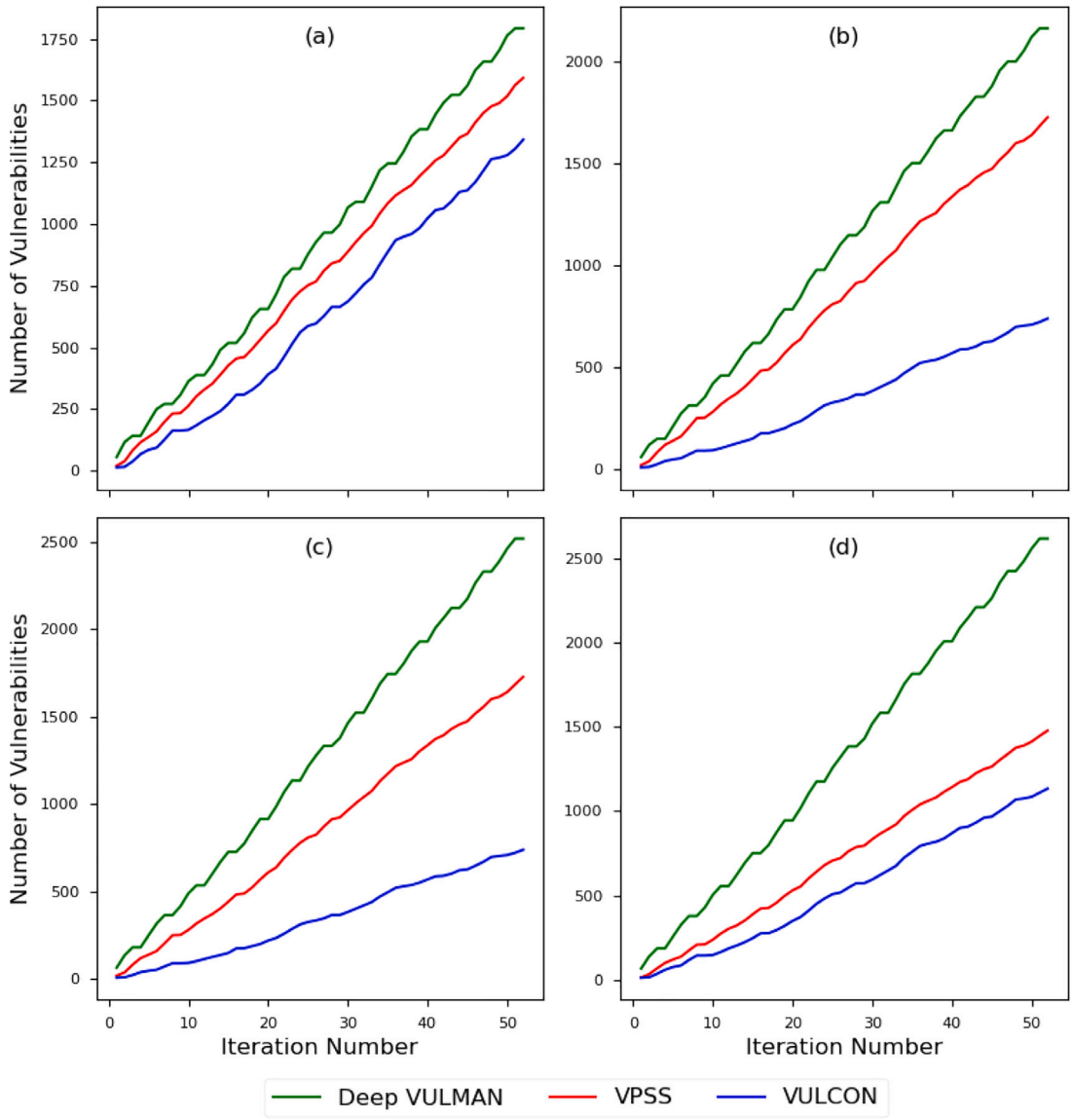

Fig. 3. Comparison of the total number of vulnerabilities selected in Scenario 1 from (a) high value assets, (b) machines with low level of protection, (c) organization-specific relevant machines, and (d) machines with intrusion alert signals.

4.3. Testing phase and evaluation criteria

We evaluated the DRL-enabled Deep VULMAN framework with two types of data: (i) simulated data from the CSOC operations simulator (Algorithm 1) and (ii) real-world vulnerability data from the collaborating CSOC. We set the standard deviation to zero during the testing phase, to avoid any further exploration by the DRL agent when taking actions using the policy network.

First, we evaluated the performance of the trained DRL agent on previously unseen data obtained from the simulation environment (Algorithm 1). To test the DRL agent on diverse set of conditions, we generated episodes that were very different from one another in terms of the number and types of vulnerability arrivals. We broadly divided them into four different scenarios. Table 4 shows the vulnerability arrival patterns in the different scenarios, where X indicates either a low or medium arrival pattern. Scenario 1 indicates a large number of vulnerability arrivals in the first half of the month. Scenario 2 indicates a large number of vulnerability arrivals during the middle of the month, while scenario 3 indicates such high arrivals towards the last half of the month. Scenario 4 represents a uniform arrival of vulnerabilities throughout the month. Finally, we evaluated the performance of the Deep VULMAN framework on previously unseen real-world data collected from the collaborating CSOC. We ran our experiments for a period of one year (i.e., 52 weeks).

Table 4
Different scenarios of vulnerability arrival patterns.

Scenario number	Arrival pattern/week
Scenario 1	[high, high, X, X]
Scenario 2	[X, high, high, X]
Scenario 3	[X, X, high, high]
Scenario 4	uniform

We compared our method with two recent vulnerability selection methods from published literature, namely, VPSS (Hore et al., 2022) and VULCON (Farris et al., 2018). We did not consider the CVSS-value based selection method in our comparison due to its limitation in taking context of an organization into consideration. None of the methods had any prior knowledge of the vulnerability arrival patterns and their performances were measured at each decision-making time step. To compare the three approaches, we recorded the vulnerabilities that were selected for mitigation from (a) high value assets, (b) machines

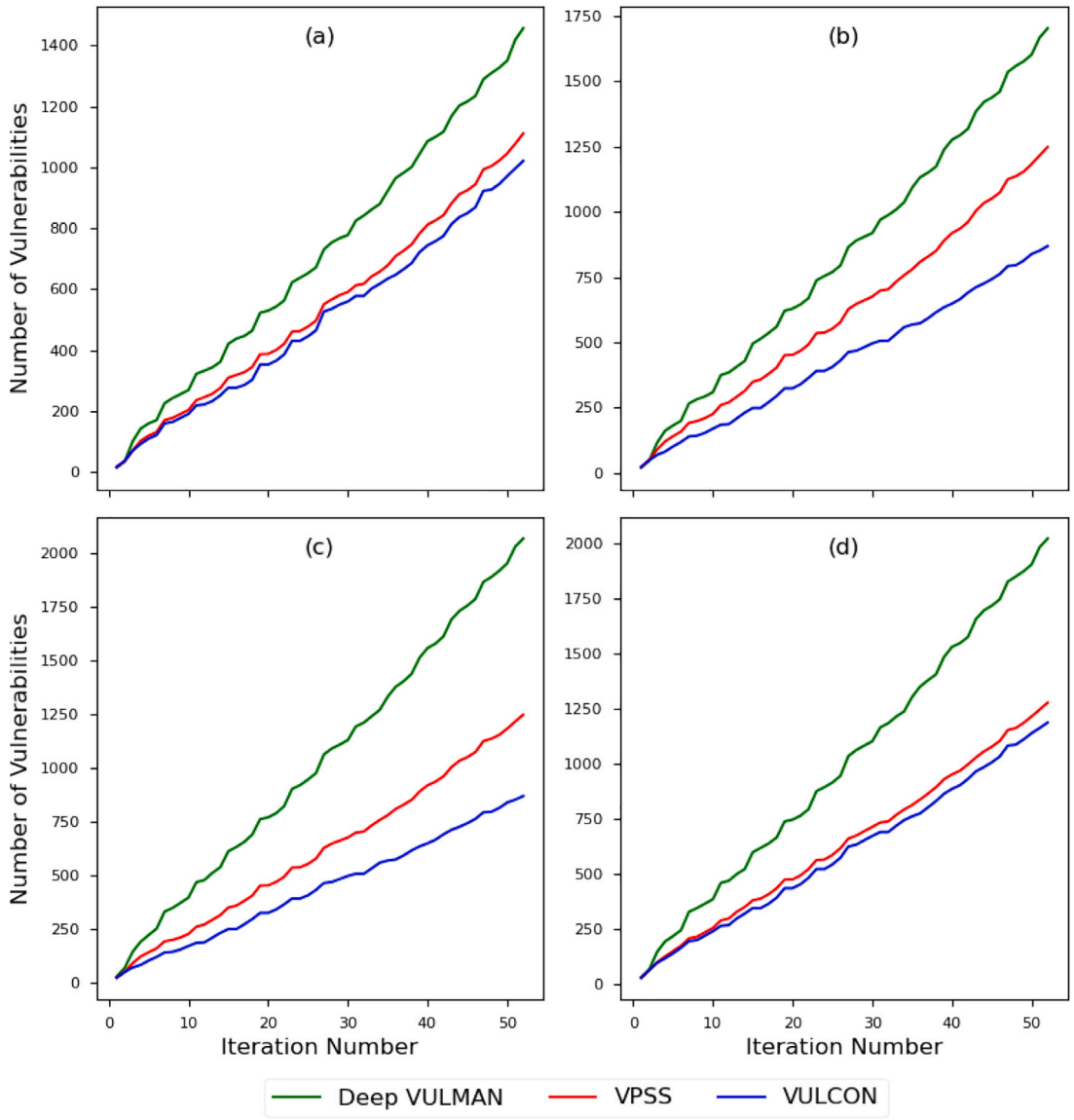

Fig. 4. Comparison of the total number of vulnerabilities selected in Scenario 2 from (a) high value assets, (b) machines with low level of protection, (c) organization-specific relevant machines, and (d) machines with intrusion alert signals.

with lower level of protection, (c) organizationally relevant machines (i.e., web and database servers), and (d) machines with intrusion detection alert signals. Next, we describe and analyze the results obtained from the aforementioned experiments.

5. Analysis of results

In this section, we first present the evaluation results obtained using the simulated environment data, followed by the results obtained using the real-world CSOC data. We conclude this section with a sensitivity analysis on the weights used in the reward function to train the DRL agent.

Figs. 3–6 show a comparison among the various methods of vulnerability management for each of the scenarios described in Table 4. Each of these figures show the total number of vulnerabilities that were selected for mitigation from (a) high value assets, (b) machines with lower level of protection, (c) organizationally relevant machines (i.e., web and database servers), (d) and machines with intrusion detection alert signals. It can be seen that the Deep VULMAN approach outperforms all the other methods in selecting the maximum cumulative number of vulnerability instances in the 52-week period in all the aforementioned factors across all scenario types (vulnerability arrival patterns). It is to be noted that in the Deep VULMAN approach, the DRL agent first determines the number of resources that would be required based on the observation (system state) at any given time (i.e., week t) and then the vulnerability prioritization and selection model utilizes the decision of the DRL agent to select the vulnerability instances for mitigation. The results show that the trained DRL agent is able to make good decisions on the number of resources to be allocated at each time step for selecting the important vulnerabilities, and the selection model is able to prioritize the selection of vulnerability instances that are critical for mitigation. The results also demonstrate that static optimization methods such as the VPSS and the VULCON used for prioritizing vulnerabilities suffer from the inflexibility in allocation of resources as they do not consider future uncertainties into consideration when making decisions.

Next, we evaluate the performance of our approach on the real-world vulnerability data set, which was collected from a collaborating CSOC for one year. Fig. 7 shows the results obtained using this data for the respective year. We can observe that the Deep VULMAN method outperforms the other two methods on this data set. The DRL agent takes a 'lean' approach when there is a lower number of critical

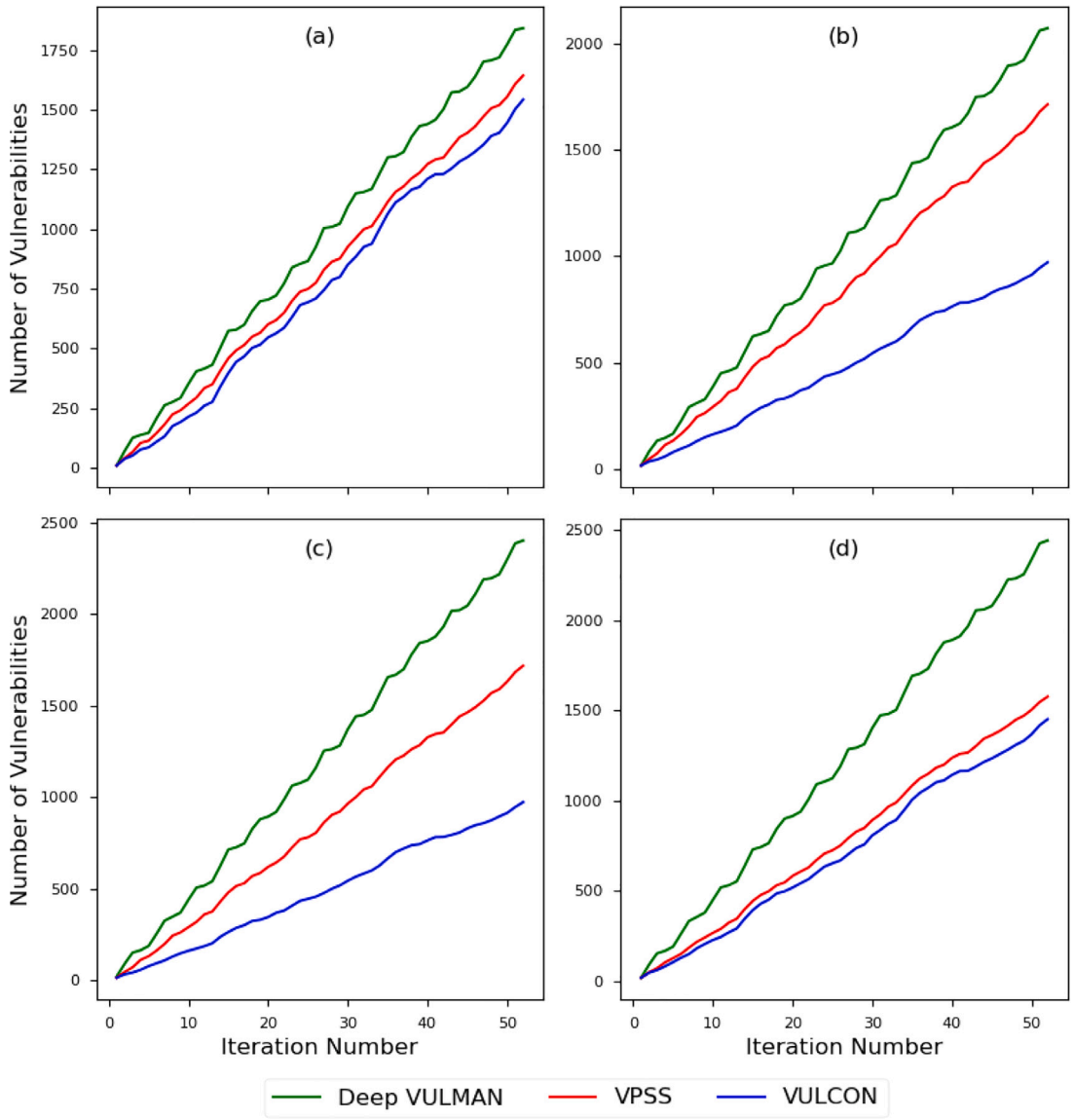

Fig. 5. Comparison of the total number of vulnerabilities selected in Scenario 3 from (a) high value assets, (b) machines with low level of protection, (c) organization-specific relevant machines, and (d) machines with intrusion alert signals.

vulnerabilities in the system, and by conserving resources for the anticipated 'big' event, it does a better (more) resource allocation at that respective time step. After that, the selection model is able to identify and prioritize these important vulnerabilities. Similar to the results obtained using the simulated environment, with our approach, more vulnerabilities are prioritized for mitigation from the important machines, i.e., web and database servers. These results matched the requirements we had gathered from the security personnel at the CSOC. Another interesting result obtained using our method is the prioritization of vulnerabilities found on machines identified in potential attacks using the IDS alert data. Further investigation of these machines also revealed that the majority of these machines were identified to have a lower level of protection (i.e., old software versions with no or limited support from the vendor), indicating that they were an easier target of adversaries and hence, vulnerabilities found in them must be prioritized. The results point towards achieving a high degree of robustness by employing the proposed DRL-enabled framework in the vulnerability triage process.

Fig. 8 shows a particular episode (month) in which the vulnerability arrival pattern fluctuates between high, medium, and low across the four-time steps (weeks). The orange bar shows the total expected mitigation time required (in minutes) to mitigate all the vulnerabilities identified in the network, and the blue bar shows the number of resources allocated by the DRL agent. To show the effectiveness of our methodology in making these resource allocation decisions, we plot another bar (shown in red in Fig. 8) displaying the total expected mitigation time of *critical* vulnerabilities. We identify *critical* vulnerabilities as follows. We first calculate the average attribute value of each vulnerability instance and then identify the ones that are at or above a threshold as follows:

$$\frac{\sum_{i=1}^{I} v_{i,j}}{I} \geq threshold \quad \forall j \tag{10}$$

Note that the attribute values are normalized and hence the maximum average attribute value that can be obtained in Eq. (10) is 1. We set the *threshold* at 0.75 and considered all the vulnerabilities with the average attribute value between 0.75 and 1 as *critical*. The dotted line in the figure represents the even distribution of resources, a commonly employed practice at the CSOCs and utilized by the other two methods (VPSS and VULCON). It can be observed that the DRL agent allocates a lower-than-average number of resources in the first week to match the arrival pattern of vulnerabilities, followed by a lower-than-average

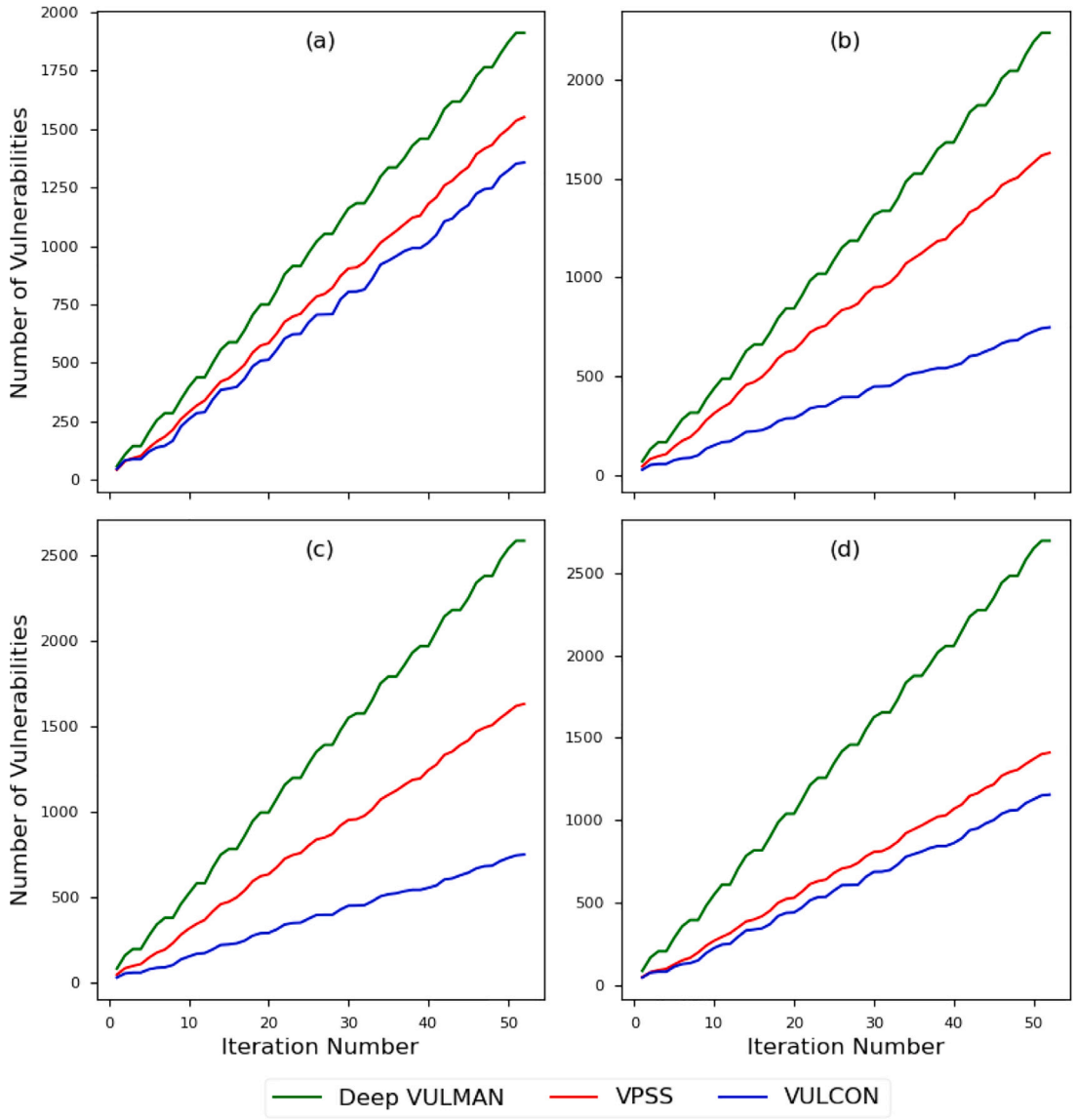

Fig. 6. Comparison of the total number of vulnerabilities selected in Scenario 4 from (a) high value assets, (b) machines with low level of protection, (c) organization-specific relevant machines, and (d) machines with intrusion alert signals.

number of resources in the second week, thereby saving more resources in anticipation of a larger number of new vulnerability arrivals in the last two weeks of the month. In this case, the DRL agent demonstrates that it can react appropriately in the first two time steps by allocating fewer resources and has learned non-trivial decisions of utilizing conserved resources to counter an anticipated future event (of high arrivals). Accordingly, the prioritization and selection model is able to prioritize the selection of vulnerabilities across all the factors. Fig. 9 (a–d) shows the scatter plots of vulnerability severity scores and the expected mitigation time for each of the four consecutive weeks in this episode. This set of plots clearly shows that our approach, with a unique reward function for the DRL agent and an effective objective function for the selection model, helps improve the organization's security posture by successfully mitigating all critical vulnerability instances over one month. We also observed superior Deep VULMAN performance in episodes with different vulnerability arrival patterns, indicating that the DRL agent has learned to make better decisions in the wake of stochastic vulnerability arrivals.

5.1. Sensitivity analysis on the reward term weights

We perform a sensitivity analysis of the reward function and present the results in this section. The reward, as shown in Eq. (1), is derived from two factors: (i) the number of important vulnerabilities that are mitigated and (ii) the number of resources that are utilized. Based on our conversations with CSOCs, we found that a CSOC's resources often comprise regular in-house security personnel and a surge team (i.e., security personnel on standby/on-call). The objective of the CSOC is to mitigate all threats to the organization by identifying and remediating important vulnerabilities. Hence, we consider two cases in performing the sensitivity analysis of the reward function used in our methodology. In case 1, we consider that a CSOC values the mitigation of important vulnerabilities and the number of resources utilized equally. An example is when a CSOC only has access to regular in-house security personnel for vulnerability management. In case 2, we consider that a CSOC values the mitigation of important vulnerabilities more than conserving resources. An example of this case is when a CSOC has a large pool of resources, including regular in-house and

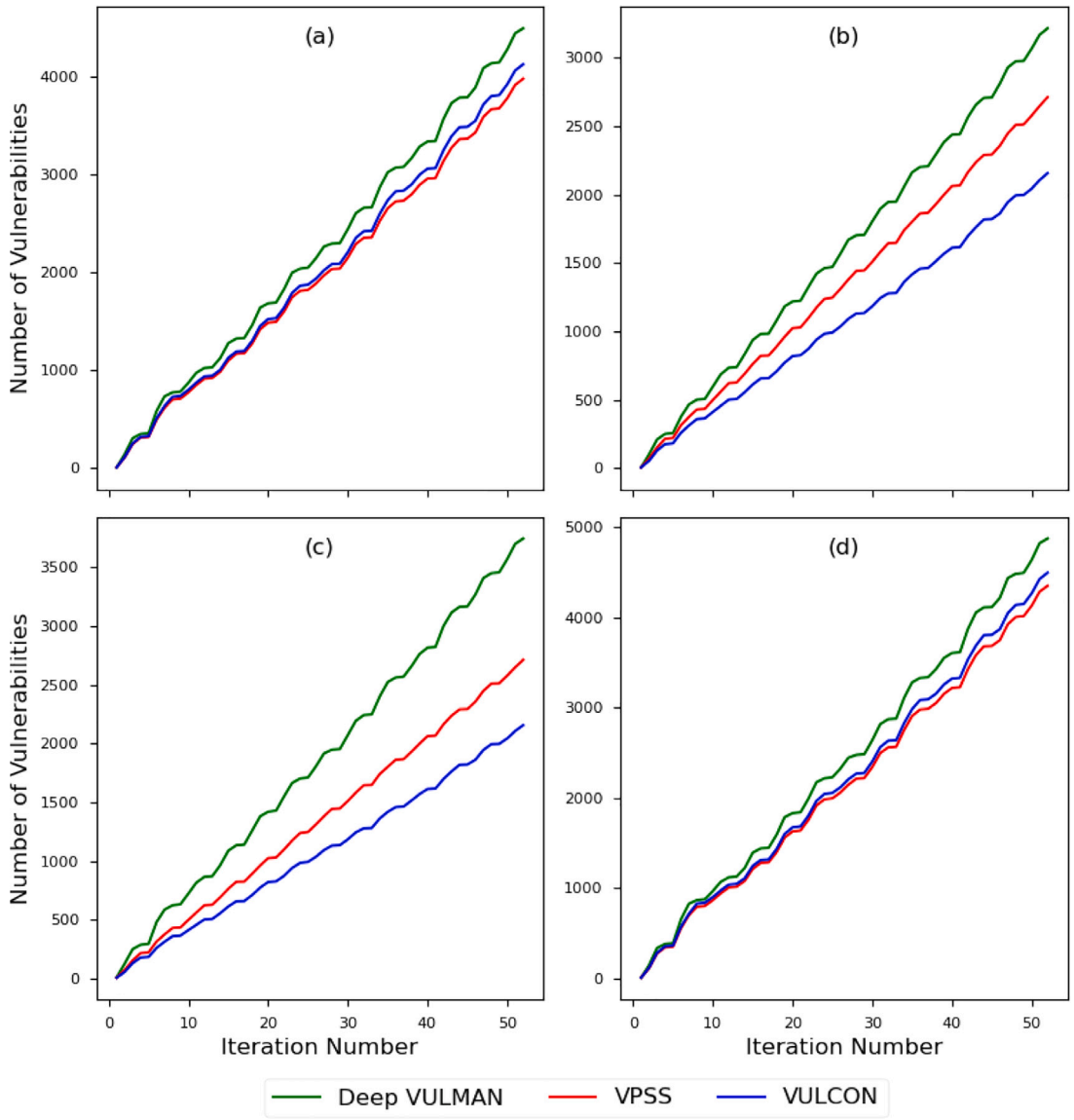

Fig. 7. Comparison of the total number of vulnerabilities selected from real-world data (one year) from (a) high value assets, (b) machines with low level of protection, (c) organization-specific relevant machines, and (d) machines with intrusion alert signals.

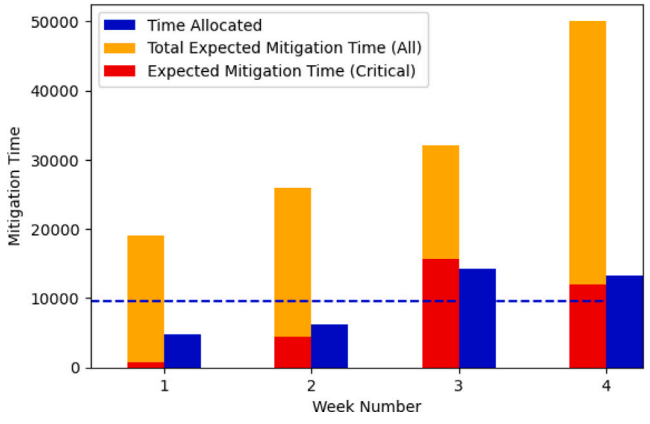

Fig. 8. Comparison between expected mitigation time of critical vulnerabilities and mitigation time allocated by the DRL agent.

surge team members. We do not consider the case in which CSOC values conserving resources more than mitigating the threats, as that will be a contradiction to its objective. We model the above two cases for the sensitivity analysis study as follows. We assign equal weights to both the reward terms in case 1 and a higher weight to the important vulnerability mitigation term (first term) compared to the resource conservation term (second term) in case 2. Fig. 10 (a–d) shows the results obtained for the four different time steps (weeks) in an episode (month). The x-axis shows the average number of critical vulnerabilities that arrived for the respective time step, and the y-axis shows the resources allocated (in minutes).

The results obtained by conducting the experiments on the simulated data indicate that case 2 yields a policy that tends to allocate more resources than the policy obtained with case 1, which is intuitive as the focus in case 2 is more on mitigating a larger number of vulnerabilities at each time step. However, we found that such an approach utilizes resources faster. For instance, in experiments where the vulnerability arrivals followed a pattern such as the one in scenario 2 or 3 (Table 4), the resources were over-utilized in the first couple of time steps with respect to the *critical* vulnerabilities and a smaller number of resources were available for the second half of the month

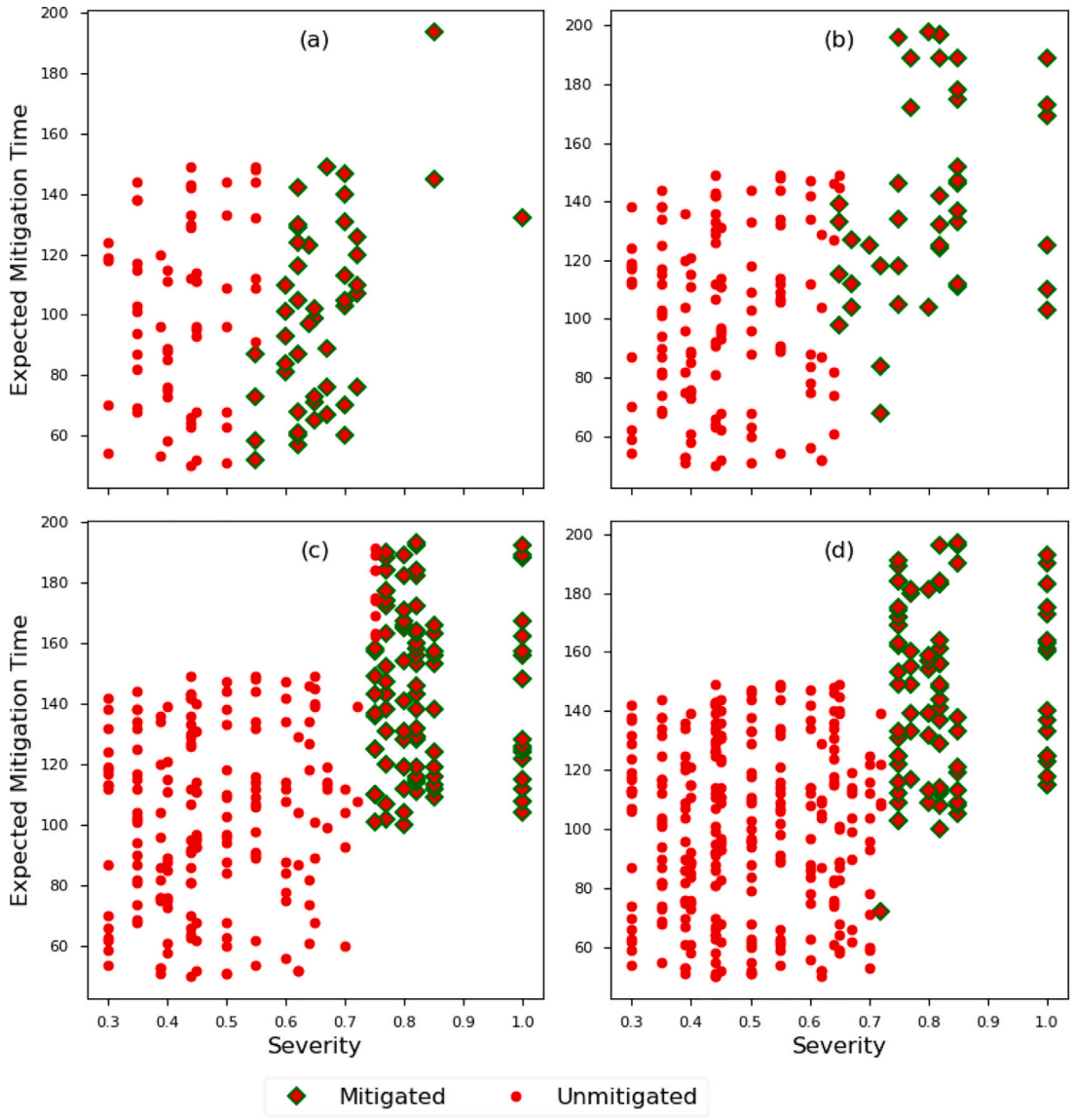

Fig. 9. Comparison among vulnerability severity scores and mitigation time for weeks 1–4 (a–d).

when a larger number of vulnerabilities arrived. As a result, the total number of *critical* vulnerabilities that were prioritized for mitigation throughout an episode was significantly lower than that obtained with case 1 (presented in our approach). This behavior of the DRL-agent can be observed in Fig. 10(a) and Fig. 10(d) by comparing the difference in resources allocated when the average number of *critical* vulnerabilities is larger in the first and last weeks of the month. Next, we present the conclusions from this study and future directions.

6. Conclusions, summary of meta-principles and future directions

The paper presented a novel cyber vulnerability management framework, Deep VULMAN, to identify and prioritize important vulnerabilities for mitigation in the wake of stochastic vulnerability arrivals in a resource-constrained environment. The problem of effective vulnerability management is posed as a sequential decision-making problem and solved using a DRL-based policy gradient method along with a mathematical optimization model. We first trained a state-of-the-art DRL agent using a simulated CSOC operations environment, which was built using real-world CSOC data, to learn the near-optimal policy of allocating resources for selecting vulnerabilities for mitigation. Next, a mathematical model for vulnerability prioritization and selection was formulated and solved using integer programming to obtain the prioritized set of important vulnerabilities selected for mitigation. We conducted our experiments on simulated and real-world vulnerability data for one year.

Below we present a summary of meta-principles obtained from the experiments conducted in this research:

1. The Deep VULMAN approach outperforms all other vulnerability management approaches by selecting the maximum number of vulnerabilities for mitigation from the four evaluation factors considered in this study. The factors included: (a) high-value assets, (b) machines with lower levels of protection, (c) organizationally relevant machines (such as web and database servers), and (d) machines identified in the IDS alerts.
2. The DRL-enabled method provided an intelligent and flexible approach by generating a near-optimal resource distribution strategy to respond to future uncertainty in the number and types of vulnerability arrivals compared to the inflexible static optimization methods.
3. The Deep VULMAN framework is able to identify machines that are easier targets for adversaries by integrating the IDS alert information.

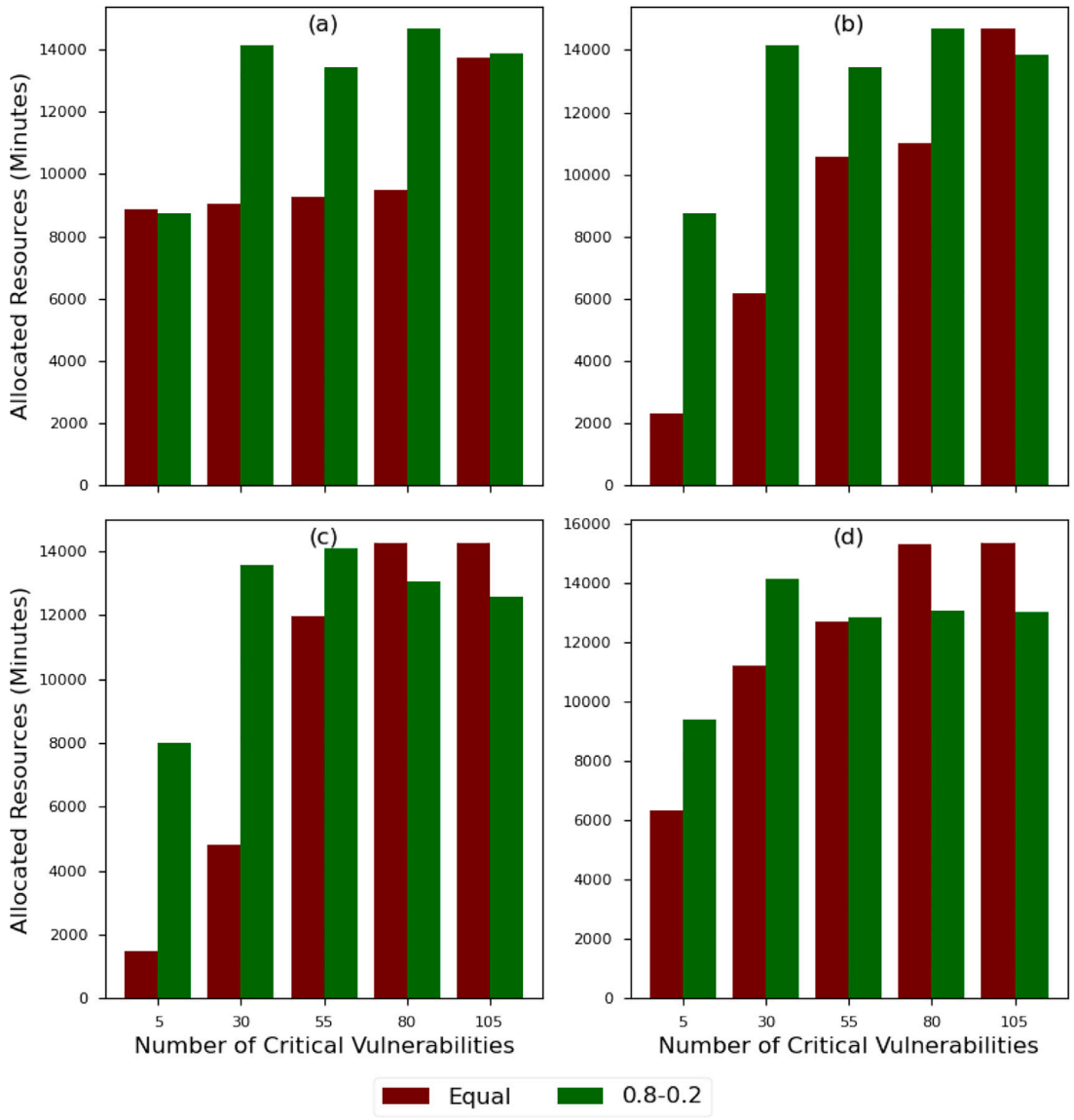

Fig. 10. Comparison of the average number of hours allocated and the average number of critical vulnerability arrivals for case 1 and case 2 in (a) week 1, (a) week 2, (a) week 3, and (a) week 4.

4. The novel reward function and the effective objective function in the framework are able to guide the AI agent in identifying and prioritizing the critical vulnerabilities for mitigation month after month.

Our DRL-enabled cyber vulnerability management framework can strengthen the security posture of an organization by generating robust policies in stochastic and resource-constrained real-world environments. In this study, we determined the (near-)optimal allocation of a limited number of resources available in a CSOC across different time steps. An interesting follow-up work or a future research direction can include the development of data–driven models to determine the optimal number of security personnel needed to achieve the performance goal of a vulnerability management team. Furthermore, a trade-off research study can be investigated comparing the impact of budget on staffing and performance of the vulnerability management team.

CRediT authorship contribution statement

Soumyadeep Hore: Methodology, Investigation, Software, Validation, Data curation, Writing – original draft. **Ankit Shah:** Conceptualization, Methodology, Investigation, Software, Data curation, Writing – original draft, Writing – review & editing, Supervision, Funding acquisition. **Nathaniel D. Bastian:** Conceptualization, Methodology, Writing – review & editing, Supervision.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

The data that has been used is confidential.

Acknowledgments

The authors would like to thank the CSOC team for providing their vulnerability data to compare the various methods used in this research. Further, this work was supported in part by the U.S. Military Academy (USMA) under Cooperative Agreement No. W911NF-22-2-0045, and the U.S. Army Combat Capabilities Development Command

(DEVCOM) C5ISR Center under Support Agreement No. USMA21056. The views and conclusions expressed in this paper are those of the authors and do not reflect the official policy or position of the U.S. Military Academy, U.S. Army, U.S. Department of Defense, or U.S. Government.

References

Ahmed, Z., Le Roux, N., Norouzi, M., & Schuurmans, D. (2019). Understanding the impact of entropy on policy optimization. In *International conference on machine learning* (pp. 151–160). PMLR.

Allodi, L., & Massacci, F. (2014). Comparing vulnerability severity and exploits using case-control studies. *ACM Transactions on Information and System Security*, *17*(1), 1–20.

Bogyrbayeva, A., Jang, S., Shah, A., Jang, Y. J., & Kwon, C. (2021). A reinforcement learning approach for rebalancing electric vehicle sharing systems. *IEEE Transactions on Intelligent Transportation Systems*.

Carvalho Melo, L., & Omena Albuquerque Máximo, M. R. (2019). Learning humanoid robot running skills through proximal policy optimization. In *2019 Latin American Robotics Symposium (LARS), 2019 Brazilian Symposium on Robotics (SBR) and 2019 Workshop on Robotics in Education* (pp. 37–42). http://dx.doi.org/10.1109/LARS-SBR-WRE48964.2019.00015.

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

Dondo, M. G. (2008). A vulnerability prioritization system using a fuzzy risk analysis approach. In *IFIP international information security conference* (pp. 525–540). Springer.

Dulac-Arnold, G., Mankowitz, D., & Hester, T. (2019). Challenges of real-world reinforcement learning. arXiv preprint arXiv:1904.12901.

Farris, K. A., McNamara, S. R., Goldstein, A., & Cybenko, G. (2016). A preliminary analysis of quantifying computer security vulnerability data in "the wild". *9825*, In *Sensors, and command, control, communications, and intelligence (C3I) technologies for homeland security, defense, and law enforcement applications XV* (p. 98250T). International Society for Optics and Photonics.

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*, *21*(4), 1–28.

FIRST. org (2020). Common Vulnerability Scoring System version 3.1: Specification Document. https://www.first.org/cvss/specification-document [Online; accessed 18-May-2022].

Fruhwirth, C., & Mannisto, T. (2009). Improving CVSS-based vulnerability prioritization and response with context information. In *2009 3rd International symposium on empirical software engineering and measurement* (pp. 535–544). IEEE.

Gallon, L. (2010). On the impact of environmental metrics on CVSS scores. In *2010 IEEE second international conference on social computing* (pp. 987–992). IEEE.

Haldar, K., & Mishra, B. K. (2017). Mathematical model on vulnerability characterization and its impact on network epidemics. *International Journal of Systems Assurance Engineering and Management*, *8*(2), 378–392.

Holm, H., Ekstedt, M., & Andersson, D. (2012). Empirical analysis of system-level vulnerability metrics through actual attacks. *IEEE Transactions on Dependable and Secure Computing*, *9*(6), 825–837.

Holm, H., Sommestad, T., Almroth, J., & Persson, M. (2011). A quantitative evaluation of vulnerability scanning. *Information Management & Computer Security*.

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*.

Kakade, S., & Langford, J. (2002). Approximately optimal approximate reinforcement learning. In *In Proc. 19th international conference on machine learning*. Citeseer.

Kirtas, M., Tsampazis, K., Passalis, N., & Tefas, A. (2020). Deepbots: A webots-based deep reinforcement learning framework for robotics. In *IFIP international conference on artificial intelligence applications and innovations* (pp. 64–75). Springer.

Kuypers, M., & Paté-Cornell, E. (2016). Center for International Security and Cooperation, Stanford, CA., https://Cisac.fsi.stanford.edu/sites/default/files/doe_cyber_security_incidents.pdf.

Liang, H. (2020). A precision advertising strategy based on deep reinforcement learning. *Ingénierie Des Systèmes D'Information*, *25*(3).

Lin, X., Wang, Y., Olkin, J., & Held, D. (2020). Softgym: Benchmarking deep reinforcement learning for deformable object manipulation. arXiv preprint arXiv:2011.07215.

McQueen, M. A., McQueen, T. A., Boyer, W. F., & Chaffin, M. R. (2009). Empirical estimates and observations of 0day vulnerabilities. In *2009 42nd Hawaii international conference on system sciences* (pp. 1–12). IEEE.

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

Mell, P., Scarfone, K., Romanosky, S., et al. (2007). A complete guide to the common vulnerability scoring system version 2.0. *1*, In *Published By FIRST-Forum of incident response and security teams* (p. 23).

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., et al. (2016). Asynchronous methods for deep reinforcement learning. In *International conference on machine learning* (pp. 1928–1937). PMLR.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., et al. (2013). Playing atari with deep reinforcement learning. arXiv preprint arXiv:1312.5602.

NIST (2022). National vulnerability database. https://nvd.nist.gov/vuln [Online; accessed 1-May-2022].

Schulman, J., Levine, S., Abbeel, P., Jordan, M., & Moritz, P. (2015). Trust region policy optimization. In *International conference on machine learning* (pp. 1889–1897). PMLR.

Schulman, J., Moritz, P., Levine, S., Jordan, M., & Abbeel, P. (2015). High-dimensional continuous control using generalized advantage estimation. arXiv preprint arXiv:1506.02438.

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

Shah, A., Farris, K. A., Ganesan, R., & Jajodia, S. (2019). Vulnerability selection for remediation: An empirical analysis. *The Journal of Defense Modeling and Simulation*, Article 1548512919874129.

Shah, A., Ganesan, R., Jajodia, S., & Cam, H. (2018). A two-step approach to optimal selection of alerts for investigation in a CSOC. *IEEE Transactions on Information Forensics and Security*, *14*(7), 1857–1870.

Sharma, R., Sibal, R., & Sabharwal, S. (2021). Software vulnerability prioritization using vulnerability description. *International Journal of Systems Assurance Engineering and Management*, *12*(1), 58–64.

Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., & Riedmiller, M. (2014). Deterministic policy gradient algorithms. In *International conference on machine learning* (pp. 387–395). PMLR.

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT Press.

Van Hasselt, H., Guez, A., & Silver, D. (2016). Deep reinforcement learning with double q-learning. *30*, In *Proceedings of the AAAI conference on artificial intelligence*. (1).

Wang, Z., Schaul, T., Hessel, M., Hasselt, H., Lanctot, M., & Freitas, N. (2016). Dueling network architectures for deep reinforcement learning. In *International conference on machine learning* (pp. 1995–2003). PMLR.

WhiteHouse (2021). Executive Order on Improving the Nation's Cybersecurity (Presidential Actions, May 12, 2021). https://www.whitehouse.gov/briefing-room/presidential-actions [Online; accessed 1-May-2022].

Xu, M., Schweitzer, K. M., Bateman, R. M., & Xu, S. (2018). Modeling and predicting cyber hacking breaches. *IEEE Transactions on Information Forensics and Security*, *13*(11), 2856–2871.