VPnet: A Vulnerability Prioritization Approach Using Pointer Network and Deep Reinforcement Learning

Zhoushi Sheng, Bo Yu(✉), Chen Liang, and Yongyi Zhang

National University of Defense Technology, ChangSha, China
{shengzhoushi12,yubo0615,liangchen16,zhangyongyi}@nudt.edu.cn

Abstract. Vulnerability prioritization is becoming increasingly prominent in vulnerability management. The contradiction between mountains of vulnerability scan results and limited remediation resources is so stark that using severity scores and crude heuristics to prioritize vulnerabilities is overwhelmed. To implement better vulnerability management, this paper proposes a vulnerability prioritization approach using a pointer network and deep reinforcement learning, called VPnet. In VPnet, the objective of vulnerability prioritization is maximizing the total risk reduction in the target environment under limited resources. First, we transform vulnerability scan reports into a matrix. Each item in the matrix consists of a vulnerability risk and cost value. The former is quantified by combining severity, threat, impact, and asset criticality factors, and the latter is an estimate of the time required to patch a vulnerability. Then, we construct a pointer network that takes the matrix and a constraint value as inputs to output a priority vulnerability remediation plan. Furthermore, we use deep reinforcement learning to train the pointer network model parameter, since obtaining pointer network labels is computationally expensive. A novel method integrating imitation learning and autonomous learning is also devised to speed up the training process and produce a better model. The proposed approach VPnet is evaluated by generating simulated scenarios. Results show that our approach develops nearly optimal solutions in seconds under different scale scenarios and constraints, and achieves a 22.8% performance improvement in a practical example, indicating that our approach is effective while exhibiting flexibility and efficiency.

Keywords: vulnerability prioritization · vulnerability management · risk · pointer network · deep reinforcement learning

1 Introduction

With the rapid development of information technology, the number of vulnerabilities is growing explosively, and tens of thousands of vulnerabilities are discovered and disclosed to the public every year. As a result, a steady stream of

Supported by the Natural Science Foundation of China (61902416, 61902412).

S. Goel et al. (Eds.): ICDF2C 2022, LNICST 508, pp. 307–325, 2023.
https://doi.org/10.1007/978-3-031-36574-4_18

security alerts is generated when firms scan systems with vulnerability detection software, and it is impractical to fix all vulnerabilities due to limited processing capacity. Hence, considering vulnerability remediation priorities in the practice of vulnerability management is critical.

The Common Vulnerability Scoring System (CVSS) has long been a de facto standard in the community when prioritizing vulnerabilities [15]. However, the CVSS aims to measure the severity levels of vulnerabilities, not the risks that people truly care about. According to Tenable Research [10], 75% of vulnerabilities with CVSS scores of 7 or higher have never been exploited, which means that vulnerabilities with high severity scores are not necessarily high-risk vulnerabilities. When the CVSS is used alone to prioritize remediation efforts, a great number of resources are wasted on patching nonhigh-risk vulnerabilities; meanwhile, massive potential risks still exist in the target network.

Increasing research has been invested in ranking and prioritizing vulnerabilities to change the status quo ([1,8,15,19,20]). Particularly, there are many methods that have been proposed to measure risk factors in addition to severity, such as threat forecasting and context impact estimation. As people's perceptions of risk are maturing, security vendors cannot wait to release various risk-based vulnerability management products, such as vulnerability prioritization rating(VPR) [23] and Vulnerability Management Services(VMS) [7], which undergo continuous improvement.

However, these products are far from satisfying customer expectations. There are two challenges that this paper seeks to address. On the one hand, the risk calculation methods of these products are opaque, poorly interpretable, and require powerful real-time data-driven, which is difficult for ordinary enterprises to implement. On the other hand, vulnerability prioritization directly on the risk score of individual vulnerabilities is not sufficient, the optimal allocation of resources should also be addressed due to limited resources. As we know, the available resources in the process of vulnerability remediation are limited. For instance, enterprises limit the total vulnerability remediation time during a vulnerability remediation campaign to ensure that other businesses are not affected. Because the time costs required for patching various vulnerabilities are different, the developed vulnerability remediation priority strategy is likely suboptimal when optimal allocation is not considered.

In this paper, we formulate the vulnerability prioritization problem as a combinatorial optimization problem, that is maximizing the total risk reduction of the target network given limited resources. The problem is NP-hard [17]. In order to address this problem efficiently, we propose a vulnerability prioritization approach using a pointer network and deep reinforcement learning, called VPnet. First, we transform vulnerability scan reports into a matrix. Each item in the matrix consists of a vulnerability risk and cost value. The former is quantified by combining severity, threat, impact, and asset criticality factors, and the latter is an estimate of the time required to patch a vulnerability. Then, we construct a pointer network that takes the matrix and a constraint value as

inputs to output a priority vulnerability remediation plan. Furthermore, we use deep reinforcement learning to train the pointer network model parameter.

The main contributions of this paper are summarized as follows:

- VPnet, a new approach for vulnerability prioritization, is proposed; VPnet uses a pointer network and deep reinforcement learning to develop a optimal vulnerability prioritization plan.
- A simple to implement and interpretable risk quantification formula is proposed; we quantify risk score by considering severity, threat, impact, and asset criticality factors.
- A novel method that integrates imitation learning and autonomous learning to improve the performance of model training is proposed.
- Our approach is validated by experiments over various scenarios to demonstrate the effectiveness of VPnet.

2 Related Literature

Various vulnerability prioritization methods have recently been proposed both in academia and industry.

In academics, [20] presented a testable stakeholder-specific vulnerability categorization (SSVC) method that avoids some problems faced by the CVSS. SSVC takes the form of decision trees for different vulnerability management communities. [9] produced the first open, data-driven framework for predicting the probability that a vulnerability will be exploited within 12 months following public disclosure; this framework is called the exploit prediction scoring system (EPSS). This system adopts a logistic regression technique that is transparent, intuitive, and easily implemented, and the output score provides useful assessments of the threat of a given vulnerability. [1] proposed an automated context-aware vulnerability risk management (AC-VRM) methodology to prioritize vulnerabilities for remediation based on organizational context rather than severity only. The proposed solution considers multiple vulnerability databases to attain great coverage of known vulnerabilities and to determine vulnerability rankings.

In industry, IBM Security X-Force Red Vulnerability Management Services (VMS) [7] prioritize the most critical vulnerabilities exposing those systems and remediate those vulnerabilities in a systematic, "light lifting" fashion. Prioritization Scan results are inputted into X-Force Red's hacker-built automated ranking engine, which enriches and prioritizes findings based on weaponized exploits and key risk factors. Tenable has authored a new research technology called the Vulnerability prioritization Rating (VPR) [23]. The VPR aims to help prioritize mitigation efforts and help the organization understand the likelihood a given vulnerability will be exploited by using a combination of machine learning and threat intelligence. Delve Security develops and provides a vulnerability management solution leveraging machine learning to automate vulnerability management scanning and prioritization [5], a key element is machine-learning Contextual Prioritization, which evaluates 40-plus factors for each vulnerability on an enterprise network.

It can be seen from the above that the current research trend is to properly measure risks. [3] recognized that risk is a combination of the"threat" faced by the target system (the ability and intention of the threat actor), the "vulnerability" (weakness or exposure) of the system, and the "impact" (consequence or destruction) of a successful exploiting of vulnerability on the organization. Guided by [3], this paper quantifies risk by combining recent works on threats and the CVSS standard. However, considering the risks of vulnerabilities alone is not sufficient for vulnerability prioritization. Some vulnerabilities present high risks, but their repair costs are higher, and it is not cost-effective to repair such vulnerabilities. To minimize the total risk with limited resources, the cost factor should be considered, and the vulnerability prioritization problem is formulated as a combinatorial optimization problem as follows:

$$G = \max_{x_i \in \{0,1\}} \sum_{i=1}^{I} r_i * x_i \quad \text{subject to } \sum_{i=1}^{I} c_i * x_i \leq C \tag{1}$$

where i is the vulnerability instance (VI) index, $1 \leq i \leq I$, I is the total number of VIs in the scan. x_i is a binary 0-1 indicator variable with value 1 if the corresponding VI is selected for remediation and 0 otherwise. r_i and c_i are the risk and cost score of VI i.

Nonetheless, the problem is NP-hard [17]. With the scale of the problem increasing, the time and space complexity grow rapidly,it cannot be efficiently solved with a traditional algorithm ([12,14]), such as a dynamic program algorithm [14]. Motivated by recent advancements in deep learning techniques, neural combinatorial optimization was proposed to address combinatorial optimization problems [2], this approach can effectively adapt to different scale scenarios. Therefore, we take advantage of the recent progress regarding neural combinatorial optimization and develop a vulnerability prioritization approach.

3 Materials and Methods

3.1 Overall Framework of VPnet

The overall framework is designed as shown in Fig 1. The framework mainly consists of two core modules: a data pre-processing module and a remediation plan generation module. In the data pre-processing module, we quantify the risk and cost of each VI from the original scanner reports, and generate an input matrix for a point network. Specifically, the risk score is a combination of four factors: severity, threat, impact, and asset criticality, while the cost score is based on the repair suggestions. The remediation plan generation module consists of offline training and online deployment. Its main task is to train a pointer network model offline by using deep reinforcement learning, and testing or applying the model online. This trained pointer network model can output an optimal priority vulnerability remediation plan given the input.

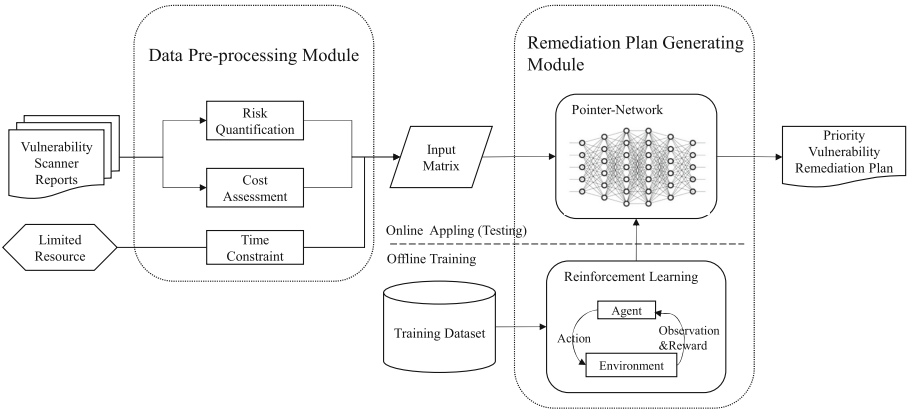


Fig. 1. The overall framework of VPnet

3.2 Data Pre-processing Module

Collecting Raw Data. We use a Nessus tool to scan the enterprise network, comprehensively collecting raw vulnerability scanner reports. Table 1 displays a sample of what the vulnerability scanner data obtained via Nessus look like, the actual IP is replaced by the host number to protect privacy and some unnecessary columns are ignored, such as port. Each vulnerability instance is measured as a unique tuple $< v, h, p >$. such as the first item in Table 1, which can be measured as <CVE-2019-0708, Host1, Windows_xp> .

Computation of Risk Score. The risk score is a fundamental concept in VPnet, It is a quantitative value that offers security teams the ability to focus their efforts on the greatest risk vulnerabilities. However, no unified standard is available for the computation of risk scores, as this task is still undergoing continuous research. Many security vendors have diverse risk calculation methods that are not transparent, such as Kenna Security Vulnerability Risk Score [11] and Rapid7's Real Risk Score [18]. Guided by the literature [3], we propose a feasible risk calculation method that is a combination of severity, threat, impact, and asset criticality factors, which is easy to implement. It can be formulated as:

$$r_i = q_i * (\alpha * S_i + \beta * T_i + \gamma * I_i) \tag{2}$$

-i is the VI index, which match a unique VI tuple $< v, h, a >$.

-S_i is the severity score of the VI obtained from the base CVSS score. The base CVSS score reflects the severity of a vulnerability by capturing its intrinsic characteristics, ranging from 1 to 10, where 1 represents the least severe score and 10 represents the most severe score. S_i is normalized to a scale between 0 and 1, and the normalization formula is $S_i = baseScore/10$.

-T_i is the threat score of the VI obtained from the EPSS. The EPSS characterizes the probability that a software vulnerability will be exploited in the

Table 1. Example of data collected via scan tool

HOST	Program	CVE	Description	Solution
Host1	Windows_xp	CVE-2019-0708	A remote code execution vulnerability exists in Remote Desktop Services when an unauthenticated visitor connects to the target system using RDP and sends specially crafted requests.	Disable Remote Desktop Services if they are not required.
Host2	Apache-httpserver	CVE-2017-7679	In Apache httpd 2.2.x, mod_mime can read one byte past the end of a buffer when sending a malicious Content-Type response header	Oracle strongly recommends that customers apply CPU fixes as soon as possible

wild in the next 30 days and produces a probability score between 0 and 1. The higher the score is, the greater the vulnerability threat. The EPSS score can be obtained via an online API(https://www.first.org/epss/api).

-I_i is the impact score of the VI obtained from the CVSS impact score. The CVSS impact score reflects the actual outcome of exploiting the vulnerability, and its value ranges from 1 to 10. I_i is normalized to a scale between 0 and 1, and the normalization formula is $I_i = impactScore/10$.

-q_i is the asset criticality score specified by the system operators. An asset criticality assessment(ACA) identifies and ranks the most critical assets in the target network, helping security teams focus efforts where they are most needed. In practice business, the score can be obtained through an ACA tool, usually ranging from 0 to 1, and the higher the value, the higher the criticality. In our experiment, we classify the criticality of assets into three levels: general, medium, and critical, corresponding to the values of 0.5, 0.75, and 1, respectively.

-$\alpha, \beta,$ and γ are manually assigned weight values, which are determined by the actual enterprise needs. In our experiments, we assigned the same 1/3 value to these three parameters, in the absence of specific background knowledge.

Assessing Vulnerability Remediation Cost. Accurately valuating the resource consumption of each vulnerability is difficult due to the various practical factors that should be considered. Fortunately, time cost of remediating a vulnerability is relatively easy to quantify, while a vulnerability remediation activity possesses a total time constraint. This paper draws on the research in [6] and assesses the vulnerability remediation cost according to the time factor.

We first divide the cost values into three categories based on the difficulty of remediation: 1 to 3 h, 3 to 6 h, and 6 to 9 h. If the solution description provided by Nessus contains keywords such as “default password,” “configuration changes,” “weak cipher,” or “password update”, it indicates that the remediation is relatively simple; therefore, the cost value falls into the first category. If a version upgrade is required in the solution, it is classified into the second category. If a system update is needed, it is classified into the third category. Next, the exact cost value is randomly generated from the corresponding period and normalized to a scale between 0 and 1. In addition, if no solution is available

for the vulnerability, we set a far greater cost value than those described above because such vulnerabilities are too costly for ordinary enterprises to patch.

Constructing the Input Matrix. After the calculation of the risk and cost value, a vulnerability instance can be represented by the node (r_i, c_i), for instance, VI_1 can be represented by $(0.422058, 0.1)$. Thus, we can construct a matrix M by combining all the scanned VI nodes. M belongs to $R^{N\times 2}$ where N is the total number of VIs in the scan. Together with a constraint value C, M is used as the input of the pointer network, where the final input belongs to $R^{(N+1)\times 2}$. Specifically, the last node is the constraint value while carrying a duplicate redundant value, and before is the M (Table 2).

Table 2. Data pre-processing

VI_ID	Host	Program	CVE_ID	Risk	Cost
1	Host1	Windows_xp	CVE-2019-0708	0.422058	0.1
2	Host2	Apache_httpserver	CVE-2016-2161	0.290605	0.3

3.3 Remediation Plan Generating Module

Pointer Network. The essence of vulnerability prioritization is to select and determine the remediation of order vulnerabilities from all the scanned VIs, and this can be formalized as a combinatorial optimization problem. Motivated by [24], we design a pointer network to solve this problem. Figure 2 presents the architecture of the pointer network. This network comprises two sub-network modules, called encoder and decoder.

The encoder network first utilizes a d-dimensional embedding layer to process the input matrix generated by data preprocessing, and each VI node is converted to a d-dimensional vector φ^T. Here, the role of the embedding layer is to embed 2-dimensional VIs into a continuous d-dimensional vector, so as to better train the pointer network model and ensure the model is inductive. Second, a recurrent neural network that consists of long short-term memory (LSTM) cells reads the embedded sequence, one node at a time, and the i-th time step outputs a latent memory state e_i, where $e_i \in R^d$. In addition, the total constraint cost value C is transferred to the decoder network for further judgment.

The decoder network also adopts LSTM cells to construct a recurrent neural network and maintains the latent memory state d_i at each step i, where $d_i \in R^d$. G is a d-dimensional vector, which is the input of the first decoding step and is treated as a trainable parameter of the neural network. Then, the decoder iteratively selects VIs for remediation until the total cost exceeds the constraint C. A pointing mechanism is the core for selecting VIs; at each step i, it produces a probability distribution over the VIs, and we can select a VI to patch under the probability of each node. A selected VI is taken as the input of the next

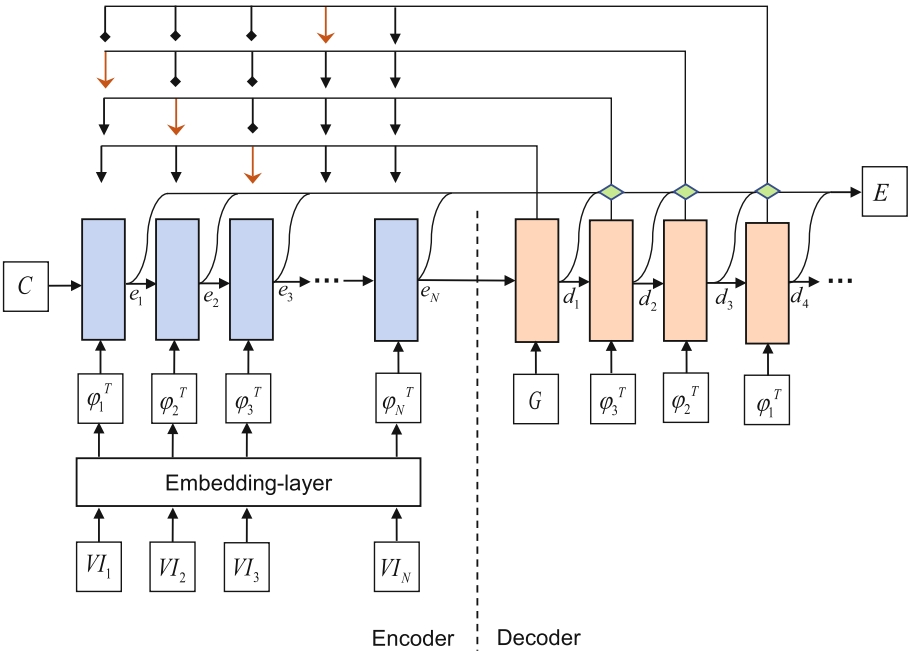


Fig. 2. Proposed pointer network for vulnerability prioritization. A tan-colored arrow suggests that the pointing node has the highest probability of being selected, and a rhombus arrow suggests that the pointing node has been selected before.

decoder step, and it will never be selected in all subsequent decoding steps. The probability distribution p is calculated by the following formulas:

$$u^i{}_j = v^T \tanh(W_1 e_j + W_2 d_i) \qquad j \in (1, ..., N) \tag{3a}$$

$$p = \text{softmax}(u^i) \qquad u^i = \{u^i{}_1, ..., u^i{}_N\} \tag{3b}$$

Where $v, W_1,$ and W_2 are learnable parameters, e_j and d_i are latent memory states, and the softmax function is a function that turns u^i into a vector of N real values that sum to 1, which can represent the probability distribution. In addition, the probability of a VI is set to $-\infty$ if this VI has been selected before.

The pointer network design is very suitable for dealing with the vulnerability prioritization problem. Due to its pointing mechanism, we can apply a neural network model trained on small-size training data to larger scenarios, which is beyond the reach of other network structures. For example, the sequence-to-sequence model [21], which is highly popular in AI, needs to train a separate model for each N , which makes deployment difficult. More details will be shown in the following chapter.

Deep Reinforcement Learning. This paper follows the deep reinforcement learning (DRL) paradigm to train the pointer network model parameters based

on full research ([2,13,16]). While supervised learning seems to be a more common way to train a neural network [4], it is not suitable for the vulnerability prioritization problem. On the one hand, obtaining optimal labels via a problem solver is expensive because the problem is NP-hard, on the other hand, the problem may be revised with a deepening understanding of vulnerability prioritization, while constructing a new problem solver is very difficult. However, it is relatively easy to find feasible solutions under resource constraints, and we can provide a reward function for verifying the quality of a set of feasible solutions. Hence, we follow the DRL paradigm to tackle the above problem, precisely speaking, we adopt a Monte Carlo policy gradient approach to solve the problem [22].

Reward Function. If there is no reward function incentive, the vulnerability priority solution output through the untrained pointer network is random rather than optimal, then what kind of solution is optimal? Obviously, our goal is to first patch the most serious vulnerabilities to minimize the overall system security risk under limited resources. Given an input matrix $S = \{VI_i\}_{i=1}^{N}$ and contraint value C, where each $VI_i = (r_i, c_i)$, we are concerned with finding a vulnerability remediation solution π that patches the vulnerabilities in sequence until the constraints are exceeded and has the best effect in reducing risk. Therefore, we design a reward function as follows:

$$L(\pi \,|S, C\,) = \sum_{i=0}^{n-1} \gamma^i r_{\pi(i)} \tag{4}$$

where γ is a discount factor that is normally set to a value in the range $[0.9, 1]$, and n is the total numbers of vulnerabilities in solution π. The practical significance of this design is not only to minimize the total risk but also to give priority to repairing higher-risk vulnerabilities. Eventually, driven by this reward function and using reinforcement learning to continuously optimize the parameters of the pointer network, the pointer network model can output the optimal vulnerability prioritization solution instead of a random one.

Optimization the Pointer Network Parameter with Policy Gradients. We adopt a Monte Carlo policy gradient approach to optimize the parameters of a pointer network denoted by θ. Our training goal is to maximize the expected return given an input matrix S and contraint value C, which is defined as:

$$J(\theta \,|S, C\,) = E_{\pi \sim p_\theta(.|s\,)} L(\pi \,|S, C\,) \tag{5}$$

The gradient of the above objective function can be formulated as follows:

$$\nabla_\theta J(\theta \,|S, C\,) = \mathbb{E}_{\pi \sim \ p_\theta(.|s\,)} \left[(L(\pi \,|S, C\,) - b(S, C)) \nabla_\theta \log p_\theta(\pi \,|S, C\,) \right] \tag{6}$$

where $b(S, C)$ is a baseline function which is important for training, we will discuss it further in the following paragraph, and $p_\theta(\pi \,|S, C\,)$ is a stochastic policy

that assigns high probabilities to high-quality solutions. It can be calculated by a chain rule:

$$p_\theta(\pi\,|S,C\,) = \prod_{i=1}^{n} p_\theta(\pi(i)|\pi(<i),S,C) \tag{7}$$

During the training phase, by sampling cases $(s_1,c_1),(s_2,c_2),...,(s_B,c_B)$ from the training data D and a solution $\pi_i\,|s_i,c_i$ per case, the gradient (6) can be approximated with Monte Carlo sampling as follows, and it can be trained with an Adam optimizer [12].

$$\nabla_\theta J(\theta) = \frac{1}{B}\sum_{i=1}^{B}(L(\pi|\mathrm{s}_i,c_i) - b(\mathrm{s}_i,c_i))\nabla_\theta \log p_\theta(\pi|\mathrm{s}_i,c_i) \tag{8}$$

Baseline Function. Constructing a good baseline function $b(\mathrm{s}_i,c_i)$ is significant for the policy gradients. Adding a baseline function does not change the expectation of the objective function, but it can effectively reduce the variance and improve the stability of the training process. On the other hand, not all solutions can be sampled due to the presence of a large number of feasible solutions. Some low-quality solutions will receive positive rewards once sampled, while some high-quality solutions will receive zero rewards if they are not sampled. As a result, the neural network cannot learn good strategies. Adding a baseline function that makes the rewards both positive and negative values can prevent this problem, and bad solutions are gradually phased out. Currently, two popular baseline functions are available, and they are described as follows.

(1) Exponential Moving Average. A simple baseline $b(\mathrm{s}_i,c_i)$ is an exponential moving average (EMA) of the rewards, reflecting the fact that the given policy improves with training over time. The EMA is calculated by the formula below:

$$EMA_t = \begin{cases} reward_0 & t=0 \\ reward_t * (\frac{\xi}{1+t}) + reward_{t-1} * (1-\frac{\xi}{1+t}) & t>0 \end{cases} \tag{9}$$

where t represents the number of training time steps. $reward_i$ is the average reward value of the i-th training step. ξ is a smoothing factor, and the most common choice is $\xi = 2$. If the smoothing factor is increased, more recent observations have more influence on the EMA.

(2) Actor-Critic Training. Another popular choice for the baseline function is to construct an auxiliary neural network called the critic network. This neural network parameterized by θ_v aims to learn the expected reward values $E_{\pi\sim\ p_\theta(.|s)}L(\pi|s)$. Given an input (s_i,c_i), the objective of the critic network is to minimize the loss between its prediction $b_{\theta_v}(s_i,c_i)$ and the actual reward $L(\pi|s_i,c_i)$; therefore, the loss function is formulated as:

$$\mathcal{L}(\theta_v) = \frac{1}{B}\sum_{i=1}^{B}\|b_{\theta_v}(s_i,c_i) - L(\pi_i|s_i,c_i)\|_2^2 \tag{10}$$

Besides, the actor network is the pointer network parameterized by θ, the complete algorithm is shown in Algorithm 1.

Algorithm 1: Actor-Critic Training

Data: training data D, total number of training steps T, and batch size B

Result: θ,θ_v

Initialize the pointer network params θ,the critic network params θ_v, training step $t = 0$

while $t < T$ **do**

  Sample instance $(s_i, c_i) \sim D$ for $i \in (1, ..., B)$

  Sample solution $\pi_i \sim p_\theta(.|s_i, c_i)$ for $i \in (1, ..., B)$

  Calculate $b_i = b_{\theta_v}(s_i, c_i)$ for $i \in (1, ..., B)$ with Critic

  Calculate $\nabla_\theta J(\theta) = \frac{1}{B} \sum_{i=1}^{B} (L(\pi|\mathrm{s}_i, c_i) - b(\mathrm{s}_i, c_i)) \nabla_\theta \log p_\theta(\pi|\mathrm{s}_i, c_i)$

  Calculate $\mathcal{L}(\theta_v) = \frac{1}{B} \sum_{i=1}^{B} \|b_{\theta_v}(s_i, c_i) - L(\pi_i|s_i, c_i)\|_2^2$

  Update $\theta \leftarrow Adam(\nabla_\theta J(\theta))$

  Update $\theta_v \leftarrow Adam(\mathcal{L}(\theta_v))$

  Update $t \leftarrow t + 1$

end

However, these two options for the baseline function perform not well in experiments, and the possible reasons for this will be analyzed in Sect. 4. To achieve improved performance, we propose a novel method that is a combination of a strong heuristic and the actor-critic algorithm, which the former is like imitation learning, while the latter is similar to autonomous learning.

(3) Our Method. Intuitively, if a vulnerability presents a high risk but is easy to repair, it should be remediated first because of cost performance, so we can find an easy yet strong heuristic algorithm. The algorithm takes the VIs ordered by their risk-to-cost ratios until the total cost exceeds the imposed constraint C. We believe that a strong heuristic algorithm can hasten the learning process of the neural network, and a novel method is proposed in this paper based on the above cognition. Our method first takes the reward of a solution provided by the strong heuristic as the baseline so that the pointer network can quickly learn good strategies and then applies the actor-critic algorithm to better conduct autonomous learning. The details is shown in algorithm 2.

Algorithm 2: Our Method

Data: training data D, number of training steps for phase1 T_1, total number of training steps T, and batch size B

Result: θ,θ_v

Initialize the pointer network params θ,the critic network params θ_v, training step $t = 0$

Phase1: Imitation learning

while $t < T_1$ **do**

  Sample instance $(s_i, c_i) \sim D$ for $i \in (1, ..., B)$

  Sample solution $\pi_i \sim p_\theta(.|s_i, c_i)$ for $i \in (1, ..., B)$

  Generate solution $\pi_{Si} \sim Strong\_Heuristic(s_i, c_i)$

  Calculate $b_i = L(\pi_{Si}|s_i, c_i)$ for $i \in (1, ..., B)$

  Calculate $\nabla_\theta J(\theta) = \frac{1}{B} \sum_{i=1}^{B} (L(\pi_i|s_i, c_i) - b_i) \nabla_\theta \log p_\theta(\pi_i|s_i, c_i)$

  Update $\theta \leftarrow Adam(\nabla_\theta J(\theta))$

  Update $t \leftarrow t + 1$

end

Phase2: Autonomous Learning

Update $\theta_v \leftarrow \theta$

while $t < T$ **do**

  Update $\theta_v, \theta \leftarrow Actor\_Critic(\theta_v, \theta)$

  Update $t \leftarrow t + 1$

end

Sampling on trained model. After the training process of the pointer network model is completed, we apply this model to output a solution, that is, a priority vulnerability remediation plan. However, the process of constructing a solution is stochastic; it requires iteratively selecting VIs to patch under the probability of each VI, which is given by the trained policy $p_\theta(\pi|S, C)$. Therefore, we sample solutions given an input case to develop a final solution, and the two sampling strategies detailed below are considered in this paper.

(1) Repeat Sampling. Our first strategy is simply to repeat an input case many times. Multiple candidate solutions are produced when given repeated input cases using the pointer network model, and we choose the final solution with the greatest reward.
(2) Shuffle Sampling. Consider the fact that if we shuffle the input sequence without changing the feature value of each VI, the optimal solutions should be

the same. At the same time, we deem that the shuffling procedure increases exploration and improves the possibility of finding an optimal procedure. Therefore, our second strategy randomly shuffles the input sequence many times, and we choose the best sequence from the multiple candidate solutions.

In addition, a temperature hyperparameter T_{softmax} plays an important role when choosing a VI in the decoding phase and influences the sampling procedure. The temperature parameter controls the softness of the probability distribution. This is realized by rewriting formula(3b) as $p = \text{softmax}(u^i/T_{\text{softmax}})$. When $T_{\text{softmax}} > 1$, the probability distribution is smoother, leading to the candidates being more divergent. The optimal temperature parameter value is determined by experimentation.

4 Experiments

Before starting the experiments, we list the table comparing VPnet with other works. From the Table 3, we see that our work is an extension of works such as CVSS [15] and EPSS [9], ensuring measuring risk is comprehensive. Some commercial tools such as VPR [23] and VMS [7] measuring risk may be more comprehensive due to richer data sources, but they are not transparent, so it is difficult to set up experimental comparisons, at the same time, these tools do not consider combination optimization problems (COP) which is important for vulnerability prioritization, and we think that is our advantage. The most comparable work is VULCON [6], which also abstracts the vulnerability prioritization problem into a combinatorial optimization problem, but it takes a traditional mixed-integer programming method that spent 3 min to complete a task, and such efficiency obviously cannot support real-time requirements, since continuous scanning makes the recommended solution to constant change. Our work shifts the computational burden to the offline training phase, while in the applying phase, we adapt to real-time requirements by a well-trained solver that can output solutions in seconds.

Table 3. Comparison of different vulnerability prioritization works

Works	Quantitative	Risk measure	Transparent	Use AI	COP
CVSS3.0 [15]	Yes	Part(Severity)	No	No	No
EPSS [9]	Yes	Part(Theats)	Yes	No	No
SVCC [20]	No(Decsion)	No	Yes	No	No
AC-VRM [1]	Yes	Part(Context-aware)	No	No	No
VMS [7]	Yes	Comprehensive	No	Yes	No
VPR [23]	Yes	Comprehensive	No	Yes	No
VULCON [6]	Yes	Comprehensive	Yes	No	Yes
VPNET	Yes	Comprehensive	Yes	Yes	Yes

In order to prove the effectiveness of VPnet, there are two main aspects to consider, on the one hand, whether the VPnet can train a good pointer network solver, the evaluation criterion is the effectiveness of reinforcement learning in the training phase, if the reward value in the training iteration can increase and finally stabilize at the highest value or near the highest value, then it means that my model is effective. On the other hand, how well the trained pointer network solver performs, and the main concerns are how much the time is spent, whether the optimal solution can be obtained, and how scalable it is. Guide by the above reasons,we conduct experiments to investigate the behavior of VPnet.

4.1 Experimental Details

Experimental Environment Configuration. The software and hardware configuration of the experimental environment is shown in Table 4.

Table 4. Experimental environment configuration

Experimental environment	Experimental parameters
OS	Windows 10
CPU	Intel(R) Core(TM) i7-10510U
GPU	NVIDIA GeForce MX250 @ 2 GB
RAM	16 GB
Programming tool	Python3.7
Deep learning framework	Pytorch1.10.1+cu102

Experimental Datasets. During the training phase, we randomly generate three training datasets, VP30, VP50 and VP100, consisting of ten thousand instances with items' risks and costs drawn uniformly at random from $[0, 1]$, and a constraint value is generated randomly in $[0, sum(c_i)]$ for each instance, such as VP30, it belongs to $R^{B\times 31\times 2}$, where B is the instances number. During the testing phase, we use the vulnerability scan reports drawn from scenarios of different sizes as raw data, and we preprocess the raw data and set different constraint values as final inputs for our model.

Experimental Setup. In order to obtain a well-trained model of a pointer network and verify the performance of our approach. We first train the pointer network models as vulnerability prioritization solvers using different methods. Next, we compare two sampling strategies and test the impact on the sampling process of a sensitive parameter T_{softmax}. Then, we test the performance of our solver by setting different numbers of VIs and various constraint values. Finally, we demonstrate the vulnerability prioritization plan provided by VPnet through a practical example.

4.2 Results

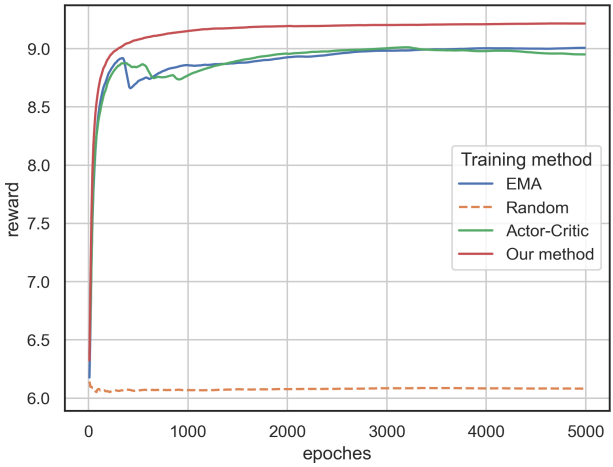
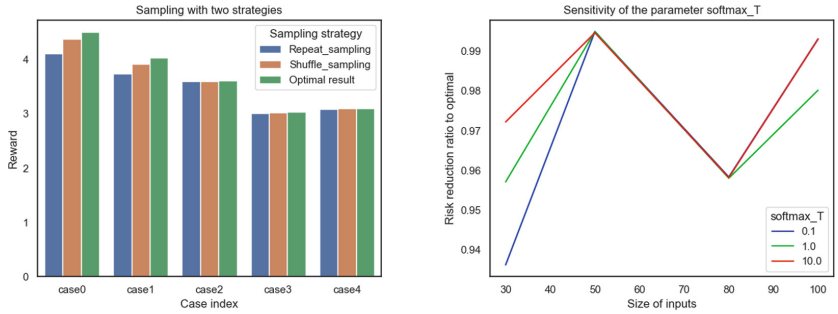


Fig. 3. The results obtained with different training methods.

The Results of Training Models. We first train VP30 solvers with two popular policy gradient algorithms, the EMA and actor-critic algorithm; however, the performances of such solvers are not satisfactory when we test them. After conducting an in-depth analysis, we think the main reason for this is that the number of feasible solutions satisfying the constraint is countless, so it is difficult to learn an optimal by strategy starting from a random state, and the objective function is likely to fall into local optima when updating the gradient. To overcome this situation, we propose a new method, which is detailed in Algorithm 2, and the result is presented in Fig 3. We can see from Fig 3 that the red curve is very steep in the early stage and quickly stabilizes at a higher reward value than those of the other two curves, indicating that our method speeds up the training process and trains better policy parameters.

The Results of the Sampling Solutions. After a well-trained model provides a policy for vulnerability prioritization, we also need to perform sampling to develop a final solution. In Fig 4(a), we construct five test cases to verify the performance of the two strategies. From the figure, we can see that among all the test cases, the result of sampling by shuffling the input is closer to the optimal result than that of repeated sampling, where the optimal result is calculated by a dynamic programming algorithm. In Fig 4(b), we test the sensitivity of the parameter T_{softmax} to inputs with different scales, and the result shows that the best-performing value for the parameter T_{softmax} may be related to the sizes of the inputs. In addition, the number of samples has a certain impact on the results; the more samples there are, the better the results. However, as

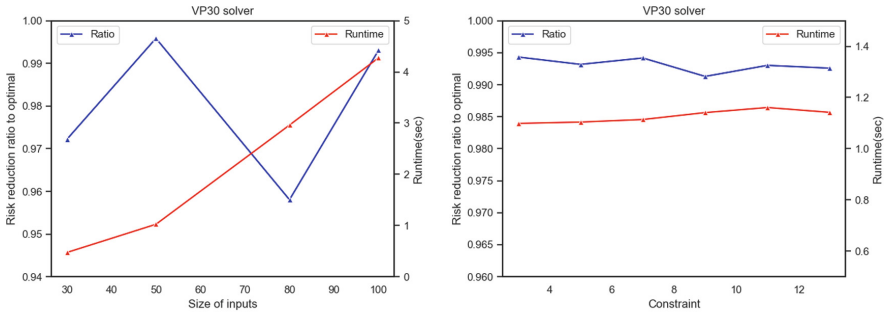
the number of samples increases, the performance improvement becomes slight, while the computational cost grows linearly, so we compromise to form a balance between obtaining better results and incurring fewer costs. The number of samples is generally set to 64 in our paper.



(a) (b)

Fig. 4. The results of sampling solutions.

The Performance of Our Solver. In Fig. 5, we use a solver trained by the VP30 dataset and a shuffling sampling strategy to develop a final vulnerability prioritization plan, and we use the risk reduction ratio to optimal indicator to evaluate the performance of VPnet, the indicator value is between 0 and 1, with larger values indicating that our method is closer to the optimal solution. As displayed in Fig 5(a), near-optimal solutions are obtained by a VP30 solver with different size inputs, and even the worst performance reaches 95.8% ratio to the optimal, while the time consumption is within a few seconds. In Fig 5(b), we assign different constraint values to a simulate instance, results show that the risk reduction ratio to the optimal all exceeded 99% while the time consumption is relatively constant at about 1 s. Both pictures indicate that our model that is trained by small-size data can be applied to the scenarios of different scales, and our approach is effective with the advantage of flexibility and efficiency.



(a) (b)

Fig. 5. The performance of our solver.

Showing Results Through a Practical Example. We test VPnet in a practical scenario. In the scenario, 53 vulnerability instances are scanned using the Nessus tool, the risk and cost value of each VI are obtained by the data preprocessing module, and the total given patch time is 30 h, corresponding to a constraint value of 3. As a result, VPnet recommends that 9 of the 53 vulnerabilities should be patched first, as shown in Table 5. These results are relatively consistent with expert perceptions, and the top-ranked vulnerabilities are recognized as the riskiest vulnerabilities by the public. For example, CVE-2019-0708, which is also known as "BlueKeep", is very popular in the hacking community. In Table 6, we show some key indicators. The baseline is the reward for a vulnerability remediation plan, which is ranked directly based on the observed risk without considering the cost. The results show that our approach achieves a 22.8% performance improvement and takes only 1.2 s.

Table 5. The priority vulnerability remediation plan provided by VPnet

Rank	Host	Program	CVE_ID
1	Host9	ProFTPD 1.3.5	CVE-2015-3306
2	Host1	Windows_xp	CVE-2019-0708
3	Host6	Apache Log4j2	CVE-2021-44228
4	Host4	Quest NetVault	CVE-2017-17653
5	Host4	Parse-server	CVE-2022-24760
6	Host7	Novell	CVE-2011-3176
7	Host7	Redis	CVE-2021-41099
8	Host0	Windows_7	CVE-2017-0143

Table 6. Key indicators

VPnet reward	Baseline	Improvement ratio	Runtime(Sec)
4.575062	3.723954	0.228549	1.257781

5 Limitations and Discussion

We concede that our current work has some limitations.

From an internal perspective of VPnet, there is no ground truth to compare in the vulnerability prioritization, and different ways of measuring risk bias the results. In addition, it is sometimes inappropriate for us to treat each vulnerability as an isolated instance. Some vulnerabilities that exist in the same software may be patched together through software updates. Such vulnerabilities should be combined into one instance to be fixed.

From a broader perspective, the actual situation is more complex than we currently consider since stakeholders in vulnerability management scenarios are

diverse, and enterprises also have different practical needs. Vulnerability prioritization may be a multiobjective optimization problem when enterprises consider a combination of economic concerns, business requirements, security, and other impacts, and research on vulnerability prioritization still needs to go deeper.

However, we believe that our approach will be inspiring to both enterprises and researchers. This paper applies a pointer network to vulnerability prioritization for the first time and demonstrates its great potential through experiments. Until now, the pointer network has achieved great success in the fields of combinatorial optimization and nature language processing (NLP) since it was first proposed. In particular, it outperforms many other neural networks in performing the summarization task of NLP. Seeing the essences of these problems through phenomena, the common denominator of these problems in different fields is ranking and choosing the most important options from the candidate nodes; this is also the heart of the pointer network. Although this problem may be revised with an evolving understanding of vulnerability prioritization and the fact that enterprises' practices for estimating risk vary to suit specific needs, we can still use a pointer network to develop a priority vulnerability remediation plan, as long as we properly redesign the input format and reward function. Therefore, we believe that using pointer networks for vulnerability prioritization is a large step in the right direction, and we will continue to augment and improve VPnet in future work.

6 Conclusion

This paper deeply discusses various factors regarding vulnerability prioritization and recognizes that the essence of vulnerability prioritization is a combinatorial optimization problem. Based on the above understanding, we present VPnet, a vulnerability prioritization approach using a pointer network and deep reinforcement learning. We evaluate VPnet through experiments, results show that our approach develops nearly optimal solutions in seconds under different scale scenarios and constraints, and achieves a 22.8% performance improvement in a practical example. We also present a new training method that integrates imitation learning and autonomous learning to improve model training performance. Additionally, we implement an easy-to-implement vulnerability risk calculation formula, which combines severity, threat, impact, and asset criticality factors.

References

1. Ahmadi, V., Arlos, P., Casalicchio, E.: Normalization of severity rating for automated context-aware vulnerability risk management. In: 2020 IEEE International Conference on Autonomic Computing and Self-Organizing Systems Companion (ACSOS-C), pp. 200–205. IEEE (2020)
2. Bello, I., Pham, H., Le, Q.V., Norouzi, M., Bengio, S.: Neural combinatorial optimization with reinforcement learning. arXiv preprint arXiv:1611.09940 (2016)
3. Blank, R.M.: Guide for conducting risk assessments (2011)

4. Caruana, R., Niculescu-Mizil, A.: An empirical comparison of supervised learning algorithms. In: Proceedings of the 23rd International Conference on Machine Learning, pp. 161–168 (2006)
5. Delve Security: Ai-driven vulnerability management solution (2020). https://www.secureworks.com/products/taegis/vdr. Accessed 07 Jul 2020
6. Farris, K.A., Shah, A., Cybenko, G., Ganesan, R., Jajodia, S.: Vulcon: a system for vulnerability prioritization, mitigation, and management. ACM Trans. Priv. Secur. (TOPS) **21**(4), 1–28 (2018)
7. IBM: X-force red vulnerability management services (2021). https://www.ibm.com/security/services/vulnerability-management. Accessed 14 Aug 2021
8. Jacobs, J., Romanosky, S., Adjerid, I., Baker, W.: Improving vulnerability remediation through better exploit prediction. J. Cybersecur. **6**(1), tyaa015 (2020)
9. Jacobs, J., Romanosky, S., Edwards, B., Adjerid, I., Roytman, M.: Exploit prediction scoring system (EPSS). Digit. Threats Res. Pract. **2**(3), 1–17 (2021)
10. Aboud, J.: Why you need to stop using CVSS for vulnerability prioritization (2020). https://www.tenable.com/research. Accessed 27 Apr 2020
11. Kenna Security: Vulnerability scores and risk scores (2021). https://www.kennasecurity.com/blog/vulnerability-scores-and-risk-scores. Accessed 16 Oct 2021
12. Kingma, D.P., Ba, J.: Adam: a method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)
13. Li, Y.: Deep reinforcement learning: an overview. arXiv preprint arXiv:1701.07274 (2017)
14. Martello, S., Pisinger, D., Toth, P.: Dynamic programming and strong bounds for the 0–1 knapsack problem. Manag. Sci. **45**(3), 414–424 (1999)
15. Mell, P., Scarfone, K., Romanosky, S.: Common vulnerability scoring system. IEEE Secur. Priv. **4**(6), 85–89 (2006)
16. Mnih, V., et al.: Asynchronous methods for deep reinforcement learning. In: International Conference on Machine Learning, pp. 1928–1937. PMLR (2016)
17. Papadimitriou, C.H.: On the complexity of integer programming. J. ACM (JACM) **28**(4), 765–768 (1981)
18. Rapid7: Quantifying risk with insightvm (2020). https://www.rapid7.com/products/insightvm/features/real-risk-prioritization/. Accessed 20 Sep 2020
19. Sharma, R., Sibal, R., Sabharwal, S.: Software vulnerability prioritization using vulnerability description. Int. J. Syst. Assur. Eng. Manag. **12**(1), 58–64 (2021)
20. Spring, J.M., Hatleback, E., Householder, A., Manion, A., Shick, D.: Prioritizing vulnerability response: a stakeholder specific vulnerability categorization. Technical report, CARNEGIE-MELLON UNIV PITTSBURGH PA PITTSBURGH United States (2019)
21. Sutskever, I., Vinyals, O., Le, Q.V.: Sequence to sequence learning with neural networks. Adv. Neural Inf. Process. Syst. **27** (2014)
22. Sutton, R.S., McAllester, D., Singh, S., Mansour, Y.: Policy gradient methods for reinforcement learning with function approximation. Adv. Neural Inf. Process. Syst. **12** (1999)
23. Tenable: Vulnerability prioritization rating (2019). https://www.tenable.com/sc-dashboards/vulnerability-priority-rating-vpr-summary. Accessed 11 Feb 2019
24. Vinyals, O., Fortunato, M., Jaitly, N.: Pointer networks. Adv. Neural Inf. Process. Syst. **28** (2015)