2024 IEEE International Conferences on Internet of Things (iThings) and IEEE Green Computing & Communications (GreenCom) and IEEE Cyber, Physical & Social Computing (CPSCom) and IEEE Smart Data (SmartData) and IEEE Congress

PentraFormer: Learning Agents for Automated Penetration Testing via Sequence Modeling

1st Yunfei Wang
National Key Laboratory of Information Systems Engineering
National University of Defense Technology
Changsha, China
wangyunfei@nudt.edu.cn

2nd Shixuan Liu
Laboratory for Big Data and Decision
National University of Defense Technology
Changsha, China
liushixuan@nudt.edu.cn

3rd Wenhao Wang
National University of Defense Technology
Changsha, China
wangwenhao11@nudt.edu.cn

4th Cheng Zhu✉
National Key Laboratory of Information Systems Engineering
National University of Defense Technology
Changsha, China
zhucheng@nudt.edu.cn

5th Changjun Fan
Laboratory for Big Data and Decision
National University of Defense Technology
Changsha, China
fanchangjun@nudt.edu.cn

6th Kuihua Huang
Laboratory for Big Data and Decision
National University of Defense Technology
Changsha, China
khhuang@nudt.edu.cn

7th Chao Chen
Laboratory for Big Data and Decision
National University of Defense Technology
Changsha, China
chenc1997@nudt.edu.cn

***Abstract*—The exponential growth of computer networks has intensified the requirement for robust security measures, rendering penetration testing—an essential practice that assesses vulnerabilities by simulating attacks—critically important. Automated penetration testing (APT) has evolved from rule-based approaches to intelligent decision-making, such as reinforcement learning (RL), better emulating the adaptability and decision-making process of human penetration testers. However, using RL for APT is challenged by the high cost of real-network interactions, the scarcity of security datasets, and the unique complexities of APT scenarios, which include long decision sequences and delayed rewards, complicating RL efficacy and convergence. Inspired by the Decision Transformer's proficiency in predicting action sequences for offline RL, coupled with its effectiveness in dealing with data efficiency and adaptability, we introduce a novel sequence-based methodology for designing APT agents. This approach is tailored to effectively manage the intricate aspects of data-scarce and complex APT scenarios. Our proposed model, PentraFormer, adeptly addresses these challenges prevalent in APT scenarios. The robustness of our model is attested to through extensive empirical evaluations conducted on APT-dedicated tasks.**

***Index Terms*—sequence modeling, automated penetration testing, decision transformer**

I. Introduction

The proliferation of computer networks has significantly enhanced societal convenience but also accentuated the need to address security issues [1] [2] [3]. Penetration testing, a leading network security assessment method, evaluates potential vulnerabilities and overall security by mimicking cyberattacks on systems [4]. It enables the identification and mitigation of security issues [5]. However, the complex nature of penetration testing, which requires sequential examination and exploitation of numerous vulnerabilities, heavily relies on expert knowledge [6]. The shortage of skilled penetration testers and the outcome's dependence on their expertise underscore the importance of automated penetration testing (APT) in reducing labor pressure and enhancing efficiency [7].

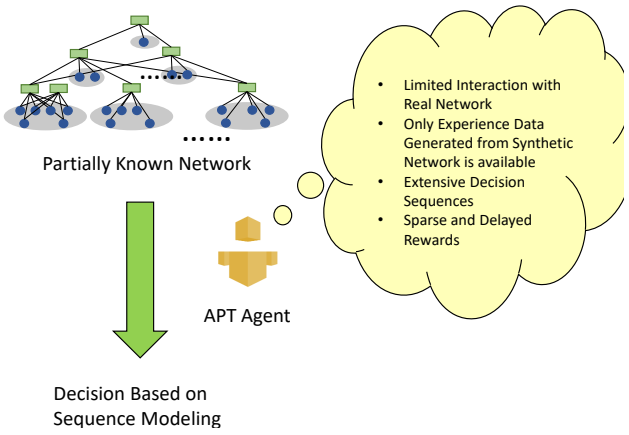

Fig. 1. Challenges arise in automated penetration testing that remains unsolved by traditional reinforcement learning algorithms.

APT mainly relies on decision-making, striving to quickly identify all network vulnerabilities or attain specific intrusion targets, especially in black-box or grey-box settings. The primary challenge is to choose appropriate attack actions and paths from partially-observable network data. Earlier, rule-based methods executed fixed scripts that maps exploitation tools with vulnerability scan outcomes, but these struggled to adapt to evolving environments [8]. Later techniques leveraged intelligent decision-making, including attack trees [9], attack

2836-3701/24/$31.00 ©2024 IEEE

DOI 10.1109/iThings-GreenCom-CPSCom-SmartData-Cybermatics62450.2024.00103

graphs [10], [11], and deterministic planning via Planning Domain Definition Language (PDDL) [12], [13]. However, these methods require complete network environments and had notable scalability limitations. Recent advancements in reinforcement learning (RL) have introduced promising strategies for decision-making in APT. APT is represented as sequential decision-making, with the observed network environment as input, the APT actions as output, and rewards assigned based on the results. This approach more closely resembles real-world, human-led penetration testing [7].

However, implementing RL for APT confronts three primary challenges. Firstly, the cost of interacting with real network environments due to security and privacy concerns is prohibitively high, limiting the efficacy of various RL algorithms which require substantial environmental interactions or offline experience. Secondly, the lack of publicly available network security datasets, requiring extensive sample data for training, compounds the data utilization challenge. Lastly, the typical characteristics of APT scenarios, including extensive decision sequences, sparse rewards, and delayed rewards, can impede RL convergence.

The Transformer model [14], renowned for its ability to model high-dimensional semantic concepts [15] and effective performance in tasks across natural language processing, computer vision, and audio processing [16], serves as the foundation for the Decision Transformer (DT) [15]. DT fuses language/sequence modeling with RL by framing the offline RL problem as a context sequence modeling task, focusing on predicting action sequences based on future expected returns [17]. This approach also enhances data efficiency [15], potentially improving generalization in APT scenarios with limited data by utilizing all trajectories. Additionally, DT's context length allows agents to leverage historical information more effectively, demonstrating stronger long-term credit assignment capabilities. This property aligns well with APT scenarios characterized by long sequences and potential delays in rewards [15]. Lastly, it is capable of transitioning from offline to online learning [17], mitigating the problem of significant performance degradation faced by most RL methods.

In this work, we present PentraFormer, an agent trained via sequence modeling techniques for APT, leveraging the property of DT to overcome decision-making challenges in APT scenarios that are challenging for most reinforcement learning methods. The contributions of this paper could be delineated as follows:

- We are the first to explore APT as a sequence modeling problem. PentraFormer enhances the utilization of sample data, effectively solving challenges associated with long sequences, sparse rewards, and delayed rewards, avoiding the need for regularization or conservatism, and is more suitable for APT scenarios.
- We develop a network generator informed by expert knowledge, producing a diverse set of networks with varying types, shifting attributes, and distinct structures. The network environment, designed around key factors in APT, retains the connectivity, ports, and services of nodes, and utilizes 50 real CVEs to assign attributes to nodes and define the agent's capabilities, thus more accurately depicting the penetration process.
- We evaluate the effectiveness of our model with APT task. Experimental results confirm our model's successful performance in APT scenarios.

The rest of this paper is organized as follows. We discuss related work systematically in Section 2. In Section 3, we introduce PentraFormer in detail, including the MDP formulation and the sequence modelling method. Section 4 presents results for APT experiments, and compares PentraFormer against existing baseline methods. Ultimately, We conclude the paper in Section 5.

II. Related Works

A. Automated Penetration Testing

Penetration testing (PT) serves as a critical approach for verifying and evaluating security vulnerabilities or weaknesses within applications, systems, or networks by simulating cyber attacks. Nevertheless, conducting penetration testing can be complex, particularly when the security team possesses limited expertise. Automated Penetration Testing (APT) has emerged as a method to simplify this process, prompting substantial academic interest.

The prevailing research in APT primarily focuses on two main paths: rule-based and intelligent APT methods. Rule-based APT method entail the extraction of penetration rules derived from expert knowledge, which are then used to organize predefined penetration steps into scripts through a process of rule correlation. However, this line suffers from a lack of adaptability, struggles to accommodate environmental dynamics, and is contingent upon the exhaustiveness of the ruleset as well as the volume of system payloads [8].

On the other hand, intelligent APT methods incorporate techniques such as attack trees [9] and attack graphs [10], [11], which are utilized to encapsulate the sequential decision-making nature of penetration testing and to plan attack paths [7]. This line faces challenges in practical scenarios and large-scale network environments, primarily due to the requirement for extensive network scenario information and the computational complexity associated with path planning. An alternative method for penetration path planning is based on classical planning, where penetration knowledge is translated into the planning domain definition language) and classical planners are employed for problem-solving [12], [13]. This method, typically applied to deterministic planning problems, focuses on path planning under conditions of complete observability and is better suited for white-box PT [18].

Recently, reinforcement learning methods have gained attention in APT [9], [19], [20] as an attempt to to tackle the complex decision-making challenges inherent in APT.

B. Reinforcement Learning for APT

RL methods involve training an agent to make sequential decisions by interacting with an environment, with the objective of maximizing cumulative reward or fulfilling particular

objectives [21]. The essence of reinforcement learning lies in modeling decision problems, commonly through a Markov Decision Process (MDP) for complete observation or a Partially Observable Markov Decision Process (POMDP) [22]. An MDP is characterized by the tuple $M = (S, A, P, R, \gamma)$, where S is the state space, A is the action space, P is the transition probability distribution, R is the reward function, and γ is the discount factor [22]. PT can be abstracted into an MDP model, enabling the use of diverse RL algorithms to training agents to select suitable actions from the discrete and extensive action space [18]. POMDP further models partial observability, i.e., agent's observations are not enough to determine the exact state. Whether reinforcement learning can successfully be applied to APT depends on how to select appropriate penetration actions in this discrete action space.

Researchers, such as Greenwald et al. [23], have employed POMDPs to model PT processes and devise multi-step testing plans, calculating success probabilities for exploitable vulnerabilities and overall attacker success via probabilistic measures. Yousefi et al. [24] utilized MulVAL for multi-host multi-stage vulnerability analysis to generate attack graphs aiding decision-making with Q-learning, while Hu et al. [9] converted network topologies into attack trees to facilitate deep RL decision-making. Gangupantulu et al. [25] incorporated firewall information into attack graph-based planning, introducing cyber-related concepts and utilizing DQN for PT decisions. Yang et al. [26] framed PT as a MORL problem, proposing a Chebyshev decomposition critic to identify various adversary strategies. Despite significant academic engagement, APT research remains in its early stages, with RL-based APT methods facing challenges such as highly abstracted network environments, the expansive state and action spaces in penetration testing, inefficient exploration, sparse rewards, complex interactions, and the scale of network environments.

C. Decision Transformer

The Transformer architecture set new standards in NLP and led to advancements in applications like natural language processing, computer vision, and audio processing [14], [16]. Recently, offline RL with Transformers has improved action prediction. The integration of Transformers with RL has taken two forms. The first apply the Transformer model to parameterize and augment the components (such as value functions and behavior policies) in standard RL algorithms. The alternative methodology casts RL as a sequence modeling task within the Transformer architecture, inspired by the shared statistical nature of offline RL and supervised learning, both involving the training of a general function from sample data. Decision Transformer [15] frames RL as conditional sequence modeling, predicting future actions via an autoregressive model trained on sequences of past states, actions, and returns-to-go. This approach harnesses the causally masked Transformer to determine optimal actions. Its advantage lies in integrating sequence model advancements into reinforcement learning, enabling the use of the Transformer's simplicity, scalability, and the associated language modeling progress like GPT-x and BERT, independent of traditional RL frameworks [27].

III. Method

In this section, we delineate our model PentraFormer , which is composed of three integral components. First, we introduce a simulation network generator that is grounded in a comprehensive expert experience framework. This generator is designed to create realistic scenarios and facilitate the training process. Second, the PT process is characterized as an MDP by incorporating real-world PT experiences. This approach enables the generation of an offline dataset that encompasses a spectrum of policy configurations. Lastly, we describe how the RL decision-making process is transformed into a sequence modeling task, thereby enhancing the resolution of the decision-making process.

A. Computer Network Characterization

The computer network is characterized by a graph $G = (V, E, X)$, with $V = \{1, 2, 3, \ldots, N\}$ symbolizing a set of nodes that constitute a multi-layered heterogeneous network composed of various local networks and architectural layers. These nodes can represent a diverse range of components, such as switches, servers, hosts, firewalls, and routers. $X = \{X_1, X_2, \ldots, X_N\}$ depicts the set of distinct features assigned to each node, with specific attribute features. The feature vector characterizing node i is exemplified as follows:

$$
\begin{aligned}
X_i = \{&'type' :' server',\\
&'lan\_id' : 0,\\
&'system' :' Windows',\\
&'port' : [20, 22, 80, 4459],\\
&'software\_ver' : [('Flink',' 1.11.0'), ('Struts2',' 2.0.0')],\\
&'vuln' : ['CVE-2020-17519','CVE-2019-9193']\}
\end{aligned}
$$

The node feature set, X_i, for a given node i, comprises parameters like node type (server/switch), associated LAN ID, installed systems (Windows/Linux), open ports ID, software and respective versions, and internal Common Vulnerabilities and Exposures (CVE) [28], all reflecting real-life scenarios. Our network simulation generator includes an extendable interface for possible integration of attributes such as executable exploits, payloads, important passwords, among others.

The set E in our network corresponds to edges, representing logical connections established through network protocols or physical links. An edge exists between nodes i and j if they can reciprocally communicate. According to our network configuration, nodes within the same LAN can directly communicate, implying an edge in G, whereas nodes from separate LANs must communicate through intermediary nodes.

To encapsulate the intricate nature of the PT process and to illustrate the challenges traditional RL approaches encounter when navigating large state spaces, we construct a comprehensive network made-up of 100 nodes.

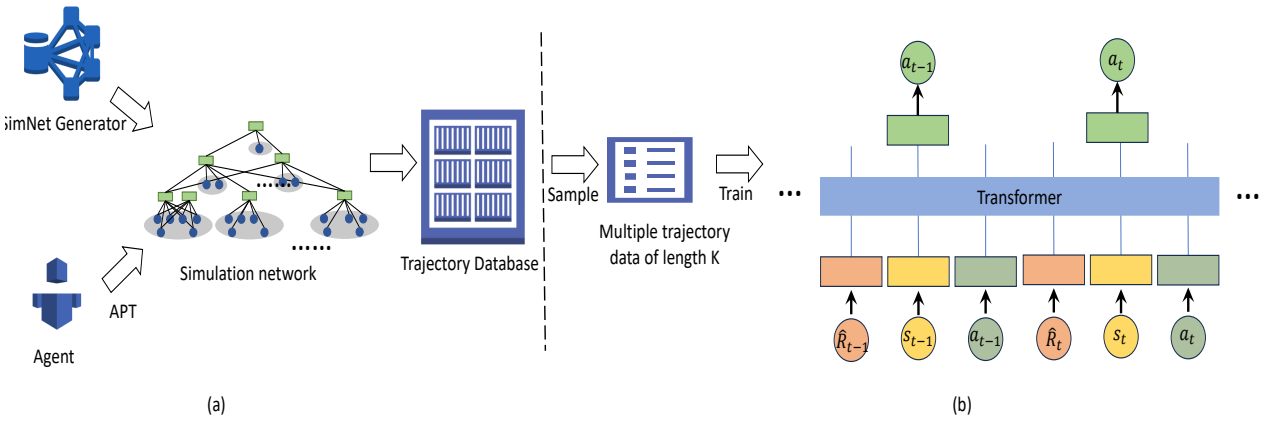

Fig. 2. PentraFormer operates in two key stages: (a) Generating synthetic network scenarios utilizing a network simulator, where naive or online RL-based agents effectively execute automated penetration testing decisions. This process results in the generation of offline trajectory datasets for extensive training. (b) Training DT to predict action based on current observation with sampled trajectories from offline datasets.

B. Markov Decision Process Characterization

In black-box or grey-box penetration testing, testers navigate an entirely or partially unknown environment, using tools like nmap, fscan, and webshell to scan the network and gather information on the active host IPs, open ports, and services [29]–[31]. With this information, combined with their prior experience, they identify vulnerabilities within hosts and, according to predefined objectives, select suitable hosts and vulnerabilities for penetration. Strategies are adjusted based on penetration results, a dynamic akin to the MDP. This resemblance justifies our representation of the PT process as an MDP in obtaining offline datasets, where the agent employs a random strategy for network penetration actions. We now elaborate on the process:

State $\mathcal{S}$: The environmental state is defined by three major components, elaborated as follows:

- Firstly, it contains an $n * n$ adjacency matrix, reflecting communication relationships between nodes, wherein 1 denotes direct communication capability, and 0 otherwise. n denotes the maximum number of nodes that the agent can process concurrently.
- Additionally, an $n * m$ node feature matrix is utilized to symbolize details such as open ports, installed software, versions, and running services. m represents the upper limit of individual node features manageable by the agent. Integer values are directly used for representation, while string values are label-mapped through a fixed dictionary.
- Lastly, the agent's privilege level on a given node is denoted as 1 for presence and 0 for absence.

Insighted by Li et al., the quantities for nodes in the network scenario and the maximum individual features manageable are set to n and m respectively, with nodes ordered by their discovery [7]. Provided that the number of hosts and individual features fall under n and m, this system allows manageable state representation. The state space is defined as $\mathbb{R}^{n*(n+m+1)}$.

Action $\mathcal{A}$: The agent's action space include reconnaissance actions and the capacity to exploit 50 real-world CVEs.

- Reconnaissance actions mimic a tester's application of scanning tools or other strategies to gather single or multiple features of a node. To deepen the richness of the feature data, repeated scanning of the same node is also permissible.
- Utilizing gathered information, the agent strategically chooses which node to attack and determines a specific CVE to exploit after careful consideration.

As the agent undergoes the PT process with partial information, decision-making entails both choosing the node and deciding the specific action, with an aim to maximize the likelihood of successful attacks, expedite the achievement of final objectives, and maximize rewards. Consequently, each action is a tuple (target node, specific operation). For example, $(10, 0)$ represents a reconnaissance action on node 10, while $(10, 13)$ signifies an attack on node 10 using the CVE labeled as 13 (CVE-2020-17519). The action space size is $\mathbb{Z}^{n*(\#CVE+1)}$.

Reward $\mathcal{R}$: Rewards in RL serve as the environment's feedback to agent actions, fostering flexibility consistent with the ultimate objective and directing the agent's exploration and decision-making for comprehensive optimization. Sparse rewards, however, can pose convergence challenges in RL. Rewards can be tailored to align with specific objectives. For instance, to maximize the acquisition of host permissions, one could assign a positive reward for each successful instance of gaining a host permission, while all other actions incur negative rewards. In our setting, we aim for the swift attainment of our penetration target, i.e., the designated target machine. Therefore, regardless of success or failure, every performed action is assigned a reward of -1, with the exception of successful attacks on the specified target machine, for which a reward of 2000 is assigned.

$$R = \begin{cases} 2000, \mathbb{I}_{arrive} \\ -1, else \end{cases}$$

State Transition $\mathcal{T}$: The state transition probability reflects the likelihood of an action succeeding in a particular state. During actual PT, exploiting existing node vulnerabilities may not guarantee success, though the probability improves with diversified penetration scripts—a notion that can be tested experimentally via different probability values. In this study, we set the state transition probability to 1, suggesting a successful attack is assured, provided the exploited vulnerability exists in the target node.

At the onset of penetration, the agent commences from the initial position, selecting a directly communicating node for probing or potential attacks, while decision-making for subsequent actions depends on retrieved information and rewards. The agent begins with an empty observation set, but with deeper penetration and more control over nodes, it accrues more observational data and potential targets for attacks. To optimize returns and rapidly achieve the final objective - reaching the designated node - the agent strategically determines the next attack target and action based on the amassed information.

C. Sequence Modeling

Our model transforms the PT decision-making process into sequence modeling, effectively addressing the challenge of predicting multi-dimensional discrete actions.

Trajectory representation. We present the sequence-based Transformer model with 'returns-to-go' ($\hat{R}_t = \sum_{t'=t}^{T} r_{t'}$), instead of direct rewards, enabling our model to generate actions based on anticipated future returns rather than past rewards. The trajectory, τ, facilitates auto-regressive training and action generation.

$$\tau = (\hat{R}_1, s_1, a_1, \hat{R}_2, s_2, a_2, \ldots, \hat{R}_T, s_T, a_T)$$

During testing, we set the 'returns-to-go' as a designated target value, initiate from the starting state s_0, and utilize our model to create actions. Subsequent to each action, the 'returns-to-go' diminishes by the achieved reward, and this cycle continues until the episode ends.

Architecture. In every time step, we allocate three tokens (returns-to-go, state, action). For each of the 3K tokens across K time steps fed into the model, a unique linear layer is learned to map the original dimension to the hidden layer's dimension, with layer normalization capturing the embedding. This input strategy of K time steps allows the model to evaluate the collective influence across those steps, a contrast to traditional RL methods focused only on the immediate past state, action, and reward. This approach proves more effective in naturally absorbing historical information's influence [32], [33]. All processed tokens employ a GPT-2 setup for the autoregressive prediction of action and state [34]–[36]. Given the action to be predicted is two-dimensional - with one dimension indicating the assault target and the other specifying the operation - we utilize two structurally identical but parametrically distinct models for the prediction, optimizing them simultaneously to overcome the complexity in decision-making offered by discrete multidimensional actions in sequence models [37], [38].

Training. During each training iteration, we batch-sample continuous trajectories, each with a sequence length of K, from the offline trajectory dataset. Training is conducted employing the AdamW optimizer [39], in conjunction with the LambdaLR Learning Rate Scheduler.

IV. Experiment

A. Experimental settings

Generated Computer Network.

We created a network with $n = 100$ nodes, each containing $m = 32$ variables to store requisite information. We incorporated 50 CVEs for attack actions. Drawing on expert insight, we devised network generation rules which include the following principles:

- The network's structure mirrors a layered defense system, composed of multiple interconnected network layers and LANs, with switches serving as the points of connection. Within the same network layer, switches do not interlink, and standby switches are provisioned to illustrate network resilience. Within a single LAN, hosts are directly connected.
- Nodes housed within the same LAN display certain commonalities, particularly in regards to operating systems, installed software, and their versions (e.g., the Windows operating system and Office productivity software). This suggests their system and software vulnerabilities might be identical—a real-world simulation of nodes within the same department sharing similar characteristics. Because user practices vary, some of these vulnerabilities may be mitigated through system patches or software version updates, a dynamic we've additionally simulated in our network.
- Both the counts of nodes within different layers and LANs, and the number of nodes connected by various switches fluctuate, emblematic of the network structure's intrinsic randomness.

Data Preparation. In order to compile offline trajectory datasets, we employed two distinct types of agents for data collection. The first agent performs penetration through randomly choosing actions, whereas the second utilizes Vanilla Policy Gradient (VPG) to learn agents for penetration, preserving the trajectory up until the point of strategy convergence and the test trajectory post-convergence. These combined trajectories subsequently yield three distinct datasets:

- RandomData: A dataset of 500 trajectories collected by the agent performing penetration by randomly selecting actions.
- MulPolData: This dataset merges 500 trajectories from the random-action agent and 1000 trajectories sourced from the VPG agent during training. The assimilation of data during the policy training optimization process

TABLE I
EPISODIC RETURN DURING TEST AFTER TRAINING ON RANDOMDATA, MULPOLDATA AND EXPERTDATA, AVERAGED OVER THREE INDEPENDENT RUNS WITH VARYING SEEDS. THE BOLD/UNDERLINED RESULTS RESPECTIVELY INDICATE THE BEST/SECOND BEST PERFORMANCES FOR EACH RELATION AMONG ALL METHODS.

Dataset	Dataset Performance	PentraFormer	CQL	25%BC	40%BC	60%BC	100%BC
RandomData	-820.5±571.8	**163.1±942.2**	-227.2±754.0	27.6±1096.5	-292.7±693.0	-387.0±783.9	<u>43.6±904.0</u>
MulPolData	-66.8±1471.2	**297.5±691.9**	208.5±591.2	21.7±999.6	-32.0±1044.0	103.0±830.1	<u>213.4±924.3</u>
ExpertData	56.7±1450.0	<u>327.2±881.1</u>	106.9±843.5	-396.2±863.2	-485.3±1189.9	-160.3±810.9	**337.5±694.0**

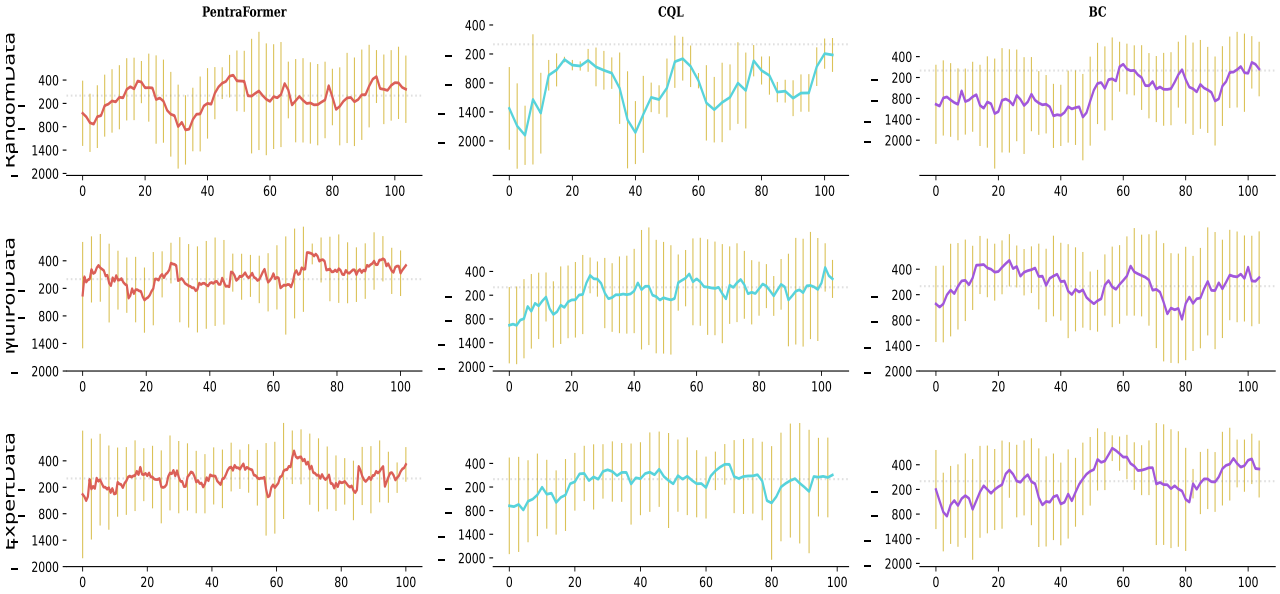

Fig. 3. Episodic return evaluated during training. The error bars denote the variance values among independent runs.

equates to collecting trajectory data through various policies.
- ExpertData: This dataset comprises of 500 trajectories collected with the VPG-trained agent, representing the expert experience datase.

Baselines. We undertake a comparative study involving our proposed model, the Conservative Q-Learning (CQL) [40], and Behaviour Cloning of varying percentile [15].

- CQL represents the state-of-the-art in model-free offline RL, an instantiation of temporal difference learning with value pessimism.
- We additionally include Percentile Behavior Cloning (%BC), where behavior cloning is implemented only on the leading X% of timesteps in the dataset, as ordered by episode returns. This design aims to discern whether PentraFormer can be perceived as conducting imitation learning on a selected penetration experience subset with a specific return. To investigate this, the percentile X% varies between standard BC (where X=100%), which trains on the entire dataset, and the cloning of only the most optimal observed trajectory (when X approaches 0%). This creates a trade-off between enhanced generalization obtained from larger data training and training a specialized model that centers on a favorable data subset. We adopt percentiles from $\{25\%, 40\%, 60\%, 100\%\}$

We conduct experiments on a server with a 64-core CPU, a 256 GB memory and eight 24GB RTX-3090Ti GPUs.

Metrics. We provide the cumulative scores per episode during the testing phase. Each method is independently run over three seeds. The results are displayed as mean ± standard deviation.

B. Experimental Results

In extensive tests across different datasets, we observed distinctly varying episodic returns. The results, presented in Table I, were gathered through averaging over three seeds.

For the RandomData dataset, it was evident that our proposed model was preeminent, exhibiting the highest episodic return with a remarkable mean score of 163.1. This achievement was notably superior relative to the second-place 100%BC model that generated a comparatively lower mean

return of 43.6. Upon applying the same analysis to the MulPolData dataset, the results unveiled parallel trends. Once again, our model consistently remained ahead of the pack, registers the highest overall average return—an impressive 297.5. In a repeated pattern, the 100%BC model managed to anchor itself as the runner-up, albeit with a significant gap, demonstrating an average return of 213.4.

After conducting training using the ExpertData dataset, the performance of the 100%BC model exhibited a marginal and insignificant improvement over our model. The easiness in training this dataset can be attributed to the fact that the data points are sampled by an agent previously trained via the VPG method. Consequently, the task of cloning behaviors becomes less intricate. Furthermore, these outcomes suggest that the model yielding optimal overall performance could fluctuate based on the choice of dataset.

In summary, our model demonstrates superior performance across three datasets.

C. Futher Analysis

In the sparse-reward conditions typical of network penetration, convergence for most RL methods tends to be slow and complex. This is a common challenge that persists irrespective of the varying approaches of different techniques. Specifically, CQL, often implemented in offline RL settings, sometimes fails to converge, leading to efficiency issues and suboptimal performance. These performance gaps represent significant obstacles to conducting competent evaluations.

Relative to two other evaluated techniques, our method stands out due to its marked stability. Our rigorous assessment revealed that it exhibited the lowest variance, indicating consistent and reliable performance. This resilience to fluctuations makes our method particularly robust and attractive for network penetration tasks. Its stability and minimized variance suggest it can effectively tackle the challenges presented by sparse reward scenarios, thereby providing a practical solution in the context of automatic network penetration.

Table II provides a comparative analysis of the time consumed per epoch on different datasets for our model, CQL, and BC. On average, across all datasets, our model required 87.27 seconds per epoch, CQL consumed 46.11 seconds, and BC took 105.50 seconds. CQL, though quickest across all datasets, compromised performance, while BC generally proved slowest. Collectively, this evidence suggests our model delivers an efficient and effective solution for APT.

TABLE II
TIME (SECOND) PER EPOCH ON DIFFERENT DATASETS.

Dataset	PentraFormer	CQL	BC
RandomData	91.23	43.36	93.09
MulPolData	79.61	46.17	116.92
ExpertData	90.97	48.81	106.49
Mean	87.27	46.11	105.50

V. CONCLUSION

We introduce a novel sequence-oriented approach for developing APT agents, drawing inspiration from the Decision Transformer's proficiency in predicting action sequences within offline RL settings, and its effectiveness in addressing data efficiency and adaptibility issues. Our proposed model adeptly navigates the complexities common to APT scenarios. The robust performance of the model is validated through comprehensive empirical analyses conducted on APT tasks.

In our subsequent research, we plan to expand on three key aspects. First, we aim to probe more advanced architectures for effectively modeling states, thereby accurately capturing the complexities of network dynamics and the structure within state spaces [41]–[44], dealing with overly large state and action spaces. Second, we plan to investigate the treatment of out-of-distribution rewards to enhance our model's robustness, thereby enhancing its performance even in diverse and unforeseen situations [45]–[47]. Finally, we intend to conduct more realistic reward modeling designed to resemble actual risk and penetration testing scenarios more closely, including parallel operations of agent and adversarial environments, thus prompting our offline RL agent to better align with the desired objectives and outcomes of APTs [18], [48]–[51].

REFERENCES

[1] Z. Chen, "Research on internet security situation awareness prediction technology based on improved rbf neural network algorithm," *Journal of Computational and Cognitive Engineering*, vol. 1, no. 3, pp. 103–108, 2022.

[2] A. Wani, R. S, and R. Khaliq, "Sdn-based intrusion detection system for iot using deep learning classifier (idsiot-sdl)," *CAAI Transactions on Intelligence Technology*, vol. 6, no. 3, pp. 281–290, 2021.

[3] R. Verma, A. Kumari, A. Anand, and V. Yadavalli, "Revisiting shift cipher technique for amplified data security," *Journal of Computational and Cognitive Engineering*, vol. 3, no. 1, pp. 8–14, 2024.

[4] F. Abu-Dabaseh and E. Alshammari, "Automated penetration testing: An overview," in *The 4th international conference on natural language computing, Copenhagen, Denmark*, 2018, pp. 121–129.

[5] Y. Stefinko, A. Piskozub, and R. Banakh, "Manual and automated penetration testing. benefits and drawbacks. modern tendency," in *2016 13th international conference on modern problems of radio engineering, telecommunications and computer science (TCSET)*. IEEE, 2016, pp. 488–491.

[6] A. Applebaum, D. Miller, B. Strom, C. Korban, and R. Wolf, "Intelligent, automated red team emulation," in *Proceedings of the 32nd Annual Conference on Computer Security Applications*, 2016, pp. 363–373.

[7] Q. Li, M. Hu, H. Hao, M. Zhang, and Y. Li, "Innes: An intelligent network penetration testing model based on deep reinforcement learning," *Applied Intelligence*, vol. 53, no. 22, pp. 27 110–27 127, 2023.

[8] B. Xing, L. Gao, J. SUN, and W. YANG, "Design and implementation of automated penetration testing system," *Application Research of Computers*, 2010.

[9] Z. Hu, R. Beuran, and Y. Tan, "Automated penetration testing using deep reinforcement learning," in *2020 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*. IEEE, 2020, pp. 2–10.

[10] S. Jha, O. Sheyner, and J. Wing, "Two formal analyses of attack graphs," in *Proceedings 15th IEEE Computer Security Foundations Workshop. CSFW-15*. IEEE, 2002, pp. 49–63.

[11] X. Ou, S. Govindavajhala, A. W. Appel *et al.*, "Mulval: A logic-based network security analyzer." in *USENIX security symposium*, vol. 8. Baltimore, MD, 2005, pp. 113–128.

[12] J. L. Obes, C. Sarraute, and G. Richarte, "Attack planning in the real world," *arXiv preprint arXiv:1306.4044*, 2013.

[13] C. Sarraute, "Automated attack planning," *arXiv preprint arXiv:1307.7808*, 2013.

[14] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[15] L. Chen, K. Lu, A. Rajeswaran, K. Lee, A. Grover, M. Laskin, P. Abbeel, A. Srinivas, and I. Mordatch, "Decision transformer: Reinforcement learning via sequence modeling," *Advances in neural information processing systems*, vol. 34, pp. 15 084–15 097, 2021.

[16] T. Lin, Y. Wang, X. Liu, and X. Qiu, "A survey of transformers," *AI open*, vol. 3, pp. 111–132, 2022.

[17] Q. Zheng, A. Zhang, and A. Grover, "Online decision transformer," in *international conference on machine learning*. PMLR, 2022, pp. 27 042–27 059.

[18] k. Chen, H. Lu, B. Fang, Y. Sun, s. Su, and Z. Tian, "Survey on automated penetration testing technology research," *Journal of Software*, pp. 1–21, 2023.

[19] A. Chowdhary, D. Huang, J. S. Mahendran, D. Romo, Y. Deng, and A. Sabur, "Autonomous security analysis and penetration testing," in *2020 16th International Conference on Mobility, Sensing and Networking (MSN)*. IEEE, 2020, pp. 508–515.

[20] F. M. Zennaro and L. Erdődi, "Modelling penetration testing with reinforcement learning using capture-the-flag challenges: Trade-offs between model-free learning and a priori knowledge," *IET Information Security*, vol. 17, no. 3, pp. 441–457, 2023.

[21] D. J. Joshi, I. Kale, S. Gandewar, O. Korate, D. Patwari, and S. Patil, "Reinforcement learning: a survey," in *Machine Learning and Information Processing: Proceedings of ICMLIP 2020*. Springer, 2021, pp. 297–308.

[22] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[23] L. Greenwald and R. Shanley, "Automated planning for remote penetration testing," in *MILCOM 2009-2009 IEEE Military Communications Conference*. IEEE, 2009, pp. 1–7.

[24] M. Yousefi, N. Mtetwa, Y. Zhang, and H. Tianfield, "A reinforcement learning approach for attack graph analysis," in *2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*. IEEE, 2018, pp. 212–217.

[25] R. Gangupantulu, T. Cody, P. Park, A. Rahman, L. Eisenbeiser, D. Radke, R. Clark, and C. Redino, "Using cyber terrain in reinforcement learning for penetration testing," in *2022 IEEE International Conference on Omni-layer Intelligent Systems (COINS)*. IEEE, 2022, pp. 1–8.

[26] Y. Yang and X. Liu, "Behaviour-diverse automatic penetration testing: A curiosity-driven multi-objective deep reinforcement learning approach," *arXiv preprint arXiv:2202.10630*, 2022.

[27] Q. Lin, H. Liu, and B. Sengupta, "Switch trajectory transformer with distributional value approximation for multi-task reinforcement learning," *arXiv preprint arXiv:2203.07413*, 2022.

[28] C. Vulnerabilities, "Common vulnerabilities and exposures," *The MITRE Corporation,[online] Available: https://cve. mitre. org/index. html*, 2005.

[29] A. Orebaugh and B. Pinkard, *Nmap in the enterprise: your guide to network scanning*. Elsevier, 2011.

[30] C. R. Young, "The f-scan system of foot pressure analysis," *Clinics in podiatric medicine and surgery*, vol. 10, no. 3, pp. 455–461, 1993.

[31] T. D. Tu, C. Guang, G. Xiaojun, and P. Wubin, "Webshell detection techniques in web applications," in *Fifth International Conference on Computing, Communications and Networking Technologies (ICCCNT)*. IEEE, 2014, pp. 1–7.

[32] N. Bjorck, C. P. Gomes, B. Selman, and K. Q. Weinberger, "Understanding batch normalization," *Advances in neural information processing systems*, vol. 31, 2018.

[33] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[34] A. Radford, K. Narasimhan, T. Salimans, I. Sutskever *et al.*, "Improving language understanding by generative pre-training," 2018.

[35] X. Liu, Y. Zheng, Z. Du, M. Ding, Y. Qian, Z. Yang, and J. Tang, "Gpt understands, too," *AI Open*, 2023.

[36] L. Floridi and M. Chiriatti, "Gpt-3: Its nature, scope, limits, and consequences," *Minds and Machines*, vol. 30, pp. 681–694, 2020.

[37] D. M. Roijers, P. Vamplew, S. Whiteson, and R. Dazeley, "A survey of multi-objective sequential decision-making," *Journal of Artificial Intelligence Research*, vol. 48, pp. 67–113, 2013.

[38] M. Wen, R. Lin, H. Wang, Y. Yang, Y. Wen, L. Mai, J. Wang, H. Zhang, and W. Zhang, "Large sequence models for sequential decision-making: a survey," *Frontiers of Computer Science*, vol. 17, no. 6, p. 176349, 2023.

[39] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[40] A. Kumar, A. Zhou, G. Tucker, and S. Levine, "Conservative q-learning for offline reinforcement learning," *Advances in Neural Information Processing Systems*, vol. 33, pp. 1179–1191, 2020.

[41] J. Bruna, W. Zaremba, A. Szlam, and Y. Lecun, "Spectral networks and locally connected networks on graphs," *Computer Science*, 2013.

[42] L. P. V. C. G. P. J. Latecki, "Graph convolutional networks based on manifold learning for semi-supervised image classification," *Computer Vision and Image Understanding*, p. 103618, 2023.

[43] S. Liu, Y. Feng, K. Wu, G. Cheng, J. Huang, and Z. Liu, "Graph-attention-based casual discovery with trust region-navigated clipping policy optimization," *IEEE Transactions on Cybernetics*, 2021.

[44] S. Liu, C. Fan, K. Cheng, Y. Wang, P. Cui, Y. Sun, and Z. Liu, "Inductive meta-path learning for schema-complex heterogeneous information networks," *arXiv preprint arXiv:2307.03937*, 2023.

[45] H. Li, X. Wang, Z. Zhang, and W. Zhu, "Out-of-distribution generalization on graphs: A survey," *arXiv preprint arXiv:2202.07987*, 2022.

[46] F. Che, G. Yang, D. Zhang, J. Tao, and T. Liu, "Self-supervised graph representation learning via bootstrapping," *Neurocomputing*, 2021.

[47] Z. Hu, Y. Dong, K. Wang, K.-W. Chang, and Y. Sun, "Gpt-gnn: Generative pre-training of graph neural networks," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2020, pp. 1857–1867.

[48] P. Herzog, "Open-source security testing methodology manual," *Institute for Security and Open Methodologies (ISECOM)*, 2003.

[49] O. Group *et al.*, "Information systems security assessment framework," *Open Information Systems Security Group*, 2006.

[50] P. Team *et al.*, "The penetration testing execution standard documentation," 2017.

[51] E. M. Hutchins, M. J. Cloppert, R. M. Amin *et al.*, "Intelligence-driven computer network defense informed by analysis of adversary campaigns and intrusion kill chains," *Leading Issues in Information Warfare & Security Research*, vol. 1, no. 1, p. 80, 2011.