Towards Automated Penetration Testing Using Inverse Soft-Q Learning

Dongfang Song Yuhong Li Ala Berzinji Elias Seid
Department of Computer and Systems Sciences, Stockholm University
Stockholm, Sweden
Email: {yh2025, alabe, elias.seid}@dsv.su.se

***Abstract*—Penetration testing (pentesting), a proactive defensive practice for identifying vulnerabilities and supporting cybersecurity management, has traditionally been conducted manually due to its heavy reliance on specialized knowledge of human experts. In this paper, we propose PT-ISQL, an automated PenTesting approach based on Inverse Soft-Q Learning (ISQL), an imitation learning algorithm that enables efficient policy learning from expert demonstrations. PT-ISQL trains an agent to take optimal actions when interacting with the pentesting environment by effectively mimicking expert behavior. Our evaluation shows that PT-ISQL achieves high performance using significantly fewer expert demonstrations compared with generative adversarial imitation learning approaches. Furthermore, it demonstrates faster convergence, improved stability, and reduced training overhead. These results suggest that PT-ISQL is a promising and practical solution for scalable, automated penetration testing.**

Keywords—penetration testing; deep reinforcement learning; imitation learning; inverse soft-Q learning; PT-ISQL.

I. INTRODUCTION

Penetration testing, commonly referred to as pentesting, is a proactive cybersecurity measure that involves simulating real-world attacks on computer systems, networks, or applications to identify vulnerabilities before they can be exploited by malicious actors. By mimicking the tactics, techniques, and procedures of actual attackers, pentesters can uncover weaknesses in systems and applications, providing insights to mitigate them and helping organizations prioritize their security strategies before real attacks occur.

Pentesting is one of the most essential cybersecurity controls. It is not a one-time activity but a continuous process that organizations must conduct regularly. The frequency of testing is typically determined by risk assessments and the organizations' operational structure. Traditionally, pentesting is a highly manual process, requiring skilled and experienced professionals to plan, execute, and adapt attacks based on system reconnaissance and responses. The manual nature of this process, combined with the increasing complexity and scale of modern IT infrastructures, makes frequent and comprehensive testing both costly and time-consuming.

As a result, researchers have begun investigating methods to automate pentesting, with the goal of increasing testing speed, reducing dependence on skilled professionals, and making the process easy to conduct. Recent work can be broadly categorized into two main approaches.

One involves the use of Large Language Models (LLMs), such as GPT-based systems [1] and deep learning agent-based systems [2][3], to use the extensive domain knowledge inherent in LLMs to automate pentesting. Although LLM-based pentesting approaches have proven highly effective in reducing manual intervention and enhancing automation, they still face notable challenges and inefficiencies inherent to LLMs, such as limited pentesting knowledge, context loss [1], unstructured data generation and efficiency [4].

The other category uses Reinforcement Learning (RL) to discover novel attack paths and adapt to dynamic environments. Among these approaches, one class, such as [5]-[7], relies on attack graphs. However, applying attack graphs to real-world, dynamic pentesting scenarios is challenging, as they require comprehensive and often unavailable system knowledge. In contrast, the other class, such as [8], uses exploitable machines to train a deep reinforcement learning model to automate the pentesting process. Nevertheless, deep RL methods often face challenges related to large state spaces and high-dimensional discrete action spaces, which complicate the training process in pentesting scenarios [9]. Moreover, the use of random exploration during the early stages of training can further introduce instability, potentially causing the model to fail to converge.

Recently, Imitation Learning (IL) has been used in automating pentesting [4] [10] [11]. IL[12], a special form of reinforcement learning, infers the reward function by modeling expert behaviour rather than relying on direct feedback from the environment. The agent learns a policy through expert demonstrations. The approaches presented in [4] [10] [11] have shown that IL can improve the performance of automated pentesting by incorporating expert knowledge. However, these approaches often face challenges in agent training, either too complex or requiring a vast amount of expert data, which is hard to gain in practice.

In this paper, we proposed PT-ISQL, an automated pentesting approach based on Inverse Soft-Q Learning (ISQL), which simplifies the process of IL by learning a soft Q-function that implicitly captures both the reward and the policy without using the complex adversarial training process. Our contributions are as follows:

- We propose an architecture for realizing automated pentesting based on ISQL;
- We implement a method for encoding pentesting tasks and actions, demonstrating that ISQL can be effectively used in automating pentesting;
- We conduct thorough experiments in a simulated network to evaluate the proposed PT-ISQL approach, and provide an in-depth analysis to the results.

The remainder of the paper is organized as follows. In Section II, we review the related work, focusing on comparing our approach with state-of-the-art approaches for pentesting based on reinforcement and imitation learning. In Section III, we present the proposed PT-ISQL approach, including the system architecture and detailed methodological steps. In Section IV, we elaborate the experiments and provide an analysis of the results. We conclude the paper and describe the future work in Section V.

II. Related Work

As mentioned above, LLMs have recently been used to automate pentesting. For example, PentestGPT [1] uses three modules, Reasoning, Generation, and Parsing to represent the specific roles typically found within penetration testing. It introduces a Pentesting Task Tree (PTT) derived from the cybersecurity attack tree to encode the ongoing status of tests and guide the subsequent actions. To overcome limitations such as limited pentesting knowledge and insufficient automation, PentestAgent [2] was proposed, which uses multi-agent collaboration to cover all phases of the pentesting lifecycle, thereby greatly reducing the need for human intervention. Although these studies have shown strong potential for automating penetration testing tasks, they also suffer from critical limitations inherent to LLMs. These include the need for vast amounts of high-quality training data, shallow task understanding, limited context windows, and lack of persistent memory. Such constraints pose huge risks in security-critical domains like pentesting. Particularly, the inability to perform long-term planning and retain stateful knowledge may hinder performance in complex tasks such as multi-step exploit chaining and privilege escalation.

We chose to use IL with limited amount of expert knowledge to automate pentesting, aiming to build compact, robust and reliable pentesting systems. In the following sections, we focus on state-of-the-art approaches using RL and IL to automate the pentesting process.

A. Pentesting based on Reinforcement Learning (RL)

In RL, an agent learns to make decisions by interacting with an environment, making it well-suited for pentesting which requires evaluating the current situation and then taking appropriate actions. As a result, many studies have applied RL to automate pentesting, such as [5]-[8]. In these approaches, the system under test is modeled as the Environment, and the pentester is the Agent. The interaction of the tester and the system is considered as the Action and results in the state change. Various techniques, including deep RL, have been used to address the complexity of RL problems for pentesting.

For example, [5] presented a method for identifying optimal attack path using a Deep Q-Network (DQN), based on a network topology generated from Shodan data and an attack tree constructed using MulVAL [13]. The traditional attack tree representation was improved by transforming it into a transition matrix, which was then used for DQN training. However, in real-world scenarios, the topology of the target network is often unknown or only partially accessible, limiting the applicability of such approaches.

Deep Exploit [14] is a pentesting tool that uses an advanced deep RL algorithm, Asynchronous Advantage Actor Critic (A3C), to exploit vulnerable servers automatically. In [8], a pentesting framework was developed based on Deep Exploit, and the influence of the number of neural networks on exploitation success rates was evaluated. However, real-world pentesting environments and complex network systems often involve a large, discrete action space, posing challenges for the training and convergence of deep RL models. For instance, algorithms like DQN select the action with the highest predicted value as the optimal choice. Unlike environments such as games, where actions are relatively deterministic and limited in scope, pentesting involves greater uncertainty and a more complex, discrete set of actions and outcomes. Furthermore, in large action spaces, multiple actions may have similar or identical values, leading to ambiguity and suboptimal decisions [15]. These limitations hinder the effective use of RL for pentesting, especially in realistic and dynamic environments.

B. Pentesting based on Imitation Learning (IL)

IL [12] is a specialized form of RL. Traditional RL relies on trial and error, with the agent receiving feedback from its environment in the form of rewards or penalties. In contrast, IL enables an agent to learn a policy directly from expert demonstrations by modeling expert behavior and inferring the underlying reward function being implicitly optimized. IL is particularly useful when it is easier for an expert to demonstrate the desired behavior than to define a reward function that would lead to the same behaviour, or when learning the policy from scratch is difficult. This makes IL especially well-suited for complex tasks such as pentesting.

A Generative Adversarial IL (GAIL) method was proposed in [16], where the reward function is learnt by measuring the similarity between an agent’s and an expert’s behavior. GAIL-PT [9], which combines GAIL and Deep Exploit, was developed to build an automatic pentesting framework. It addresses the challenge of high-dimensional action space by using the GAIL algorithm. GAIL-PT performs well in small-scale network environments (with or without honeypots), and large-scale networks, showing the potential of using IL for automated pentesting. However, the training process of GAIL requires careful tuning of hyperparameters and techniques, such as gradient penalization. Moreover, GAIL-PT is prone to overfitting with expert trajectory distributions and may not generalize well to different environments.

In the framework (i.e., DQfD-AIPT) [11], a method using expert knowledge is proposed. It combines transformed abstract expert knowledge with collected pentesting traces over various network scenarios. It provides a different method for solving the overfitting problem. However, despite using a less complex algorithm, this method requires a large amount of expert data to build the expert database.

Compared with these studies, our approach uses a more efficient way to train a pentesting agent, which requires far fewer expert demonstrations while converges more quickly.

III. Pentesting based on Inverse Soft-Q Learning

A. System Architecture

IL has developed into three main methods, Behaviour Cloning (BC), Direct Policy Learning (DPL) and Inverse Reinforcement Learning (IRL). BC is the simplest form of

IL, using supervised learning on expert data. It is widely used but can lead to cascade errors. DPL requires the presence and interaction with an expert. IRL, on the other hand, aims to infer the environment's reward function from expert demonstrations and then uses RL to discover the optimal policy—one that maximizes the inferred reward function.

ISQL is a recent method for implementing IRL, incorporating soft Q-Learning into the inverse learning process. ISQL has been shown to require less expert demonstration data to achieve comparable performance. Moreover, it considers stochastic or noisy expert actions. Thus, we chose ISQL to automate pentesting. Figure 1 illustrates the basic architecture of our automated pentesting framework based on ISQL: PT-ISQL. It consists of three main components. The Pentesting Environment represents the network environment where vulnerabilities are assessed by using the automated pentesting approach. It is the environment with which the RL agent and human experts interact during the pentesting. The environment returns the corresponding state information after each action taken by the agent or expert.

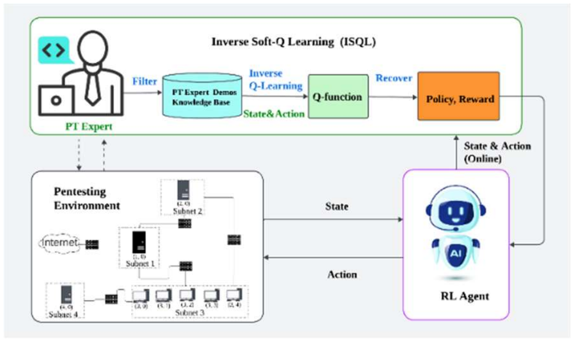

Figure 1. System architecture for PT-ISQL.

The ISQL component is the core of our system, implementing the ISQL algorithm tailored for pentesting. It interacts with the Pentesting Environment and generates the information used by the RL Agent. Expert demonstrations are first collected, consisting of recorded sequences of actions taken by human security experts or earlier runs of an RL agent during attempts to discover and exploit vulnerabilities in the Pentesting Environment. These demonstrations capture high-quality, goal-directed behaviours, such as exploiting services, vulnerability detection, and escalating privileges. Then, the resulting state-action pairs are used in the training process to get a soft Q-function, which enables the model to infer both the reward function and the policy that best explain the experts' behaviours. This process allows the RL agent to imitate expert strategies by learning from their demonstrations, even when the experts' behaviours are stochastic or suboptimal.

Using the learned policy, the RL Agent component interacts with the Pentesting Environment, navigating networks and selecting actions, such as scanning, exploiting, escalating privileges, or exfiltrating data, much like a fully autonomous AI red team agent. Since ISQL allows for stochastic behaviour (a soft policy), the agent can randomize attack sequences, adapt to defensive changes, such as patched systems or Intrusion Detection Systems (IDS), and make context-aware decisions that evolve over time. In addition, because the reward function captures underlying intent (e.g., reaching high-value targets, or staying undetected), the agent can generalize its knowledge to unseen network topologies or adapt to different vulnerabilities.

Through the interaction of these three components, fully automated pentesting can be achieved.

B. *Inverse Soft-Q Learning for Pentesting*

The PT-ISQL process consists of three main steps:

Step 1: process expert demonstrations. For each trajectory τ in the set of expert demonstrations D_{expert}, the state-action pairs (s, a) are extracted. These pairs are then used as inputs for the reward and Q networks.

Step 2: conduct iterative training via ISQL. Instead of learning a policy from a reward function, ISQL simplifies the process of the IRL by learning a Q-function that implicitly captures both the reward and the policy without using the complex adversarial training process. The goal of this step is to learn rewards and Q-values that align with expert behaviour. The target of the Q-function is computed based on the current reward $r(s,a)$, the expected value of the next state's Q-values and a soft entropy term $-\alpha\log \pi(a|s)$, where $\pi(a|s)$ is the policy derived from the Q-values using a softmax function, scaled by the entropy temperature α. This reflects the fact that experts not only try to perform well (i.e., high rewards) but also act stochastically and robustly, avoiding always picking the single "best" action. It balances the reward maximization and exploration through entropy. Namely

$$Q_{target}(s,a) = r\,(s,a) + \gamma\, \mathrm{E}_{a'}\,[Q(s',a')] - \alpha \log \pi(a'|s') \quad (1)$$

$$\pi(a|s) = \mathrm{softmax}\,(Q(s,a)/\,\alpha) \quad (2)$$

Figure 2 shows the pseudocode of this step.

```
Algorithm 1 Inverse soft Q-Learning (ISQL) for Pentesting
Require:
    Input: Expert trajectories $\mathcal{D}_{\text{expert}} = \{(s, a, s')\}$, states $\mathbf{s}$, Actions $\mathbf{a}$, discount factor $\gamma$, entropy temperature $\alpha$, learning rate $\eta$, total iterations $T$
        //Expert trajectories $\mathcal{D}_{\text{expert}}$ are from pentesting environment;
        //states $\mathbf{s}$, such as configuration, vulnerability information of all hosts (open ports, access level...) ;
        //Actions $\mathbf{a}$, such as scans, exploits, or privilege escalations etc. similar to expert behavior.
Ensure:
    Output: Learned Q-function $Q_\theta(s, a)$ from which policy $\pi(a|s) \propto \exp(Q_\theta(s, a)/\alpha)$ can be derived
1: Initialize Q-function $Q_\theta(s, a)$ with parameters $\theta$
2: for $t = 1$ to $T$ do
3:     Sample batch $\{(s, a, s')\} \sim \mathcal{D}_{\text{expert}}$
4:     Compute soft values: $V(s') \leftarrow \alpha \cdot \log \sum_{a'} \exp(Q_\theta(s', a')/\alpha)$
5:     Compute reward: $\hat{r} \leftarrow (Q_\theta(s, a) - \gamma V(s'))$
6:     Compute loss:
           $\mathcal{L} \leftarrow -E[\hat{r}] + E[Q_\theta(s, a) - \gamma V(s')] + \frac{1}{4\alpha} E[\hat{r}^2]$
7:     Update Q-function parameters: $\theta \leftarrow \theta - \eta \nabla_\theta \mathcal{L}$
8: end for
```

Figure 2. Pseudocode of the ISQL training for pentesting.

Step 3: agent execution. The trained agent can now use the learned policy to act in the environment autonomously, performing pentesting tasks, such as discovering attack paths, exploiting systems, and chaining exploits toward high-value targets, which enables realistic and adaptive red teaming.

IV. Evaluation and Analasys

A. *Experiment Setup*

We implemented the proposed approach in a virtual machine running Kali Linux. MiniConda (Version 24.1.2) was used to set up the Python virtual environment for building the three components of PT-ISQL. We used the NetworkAttack Simulator (NASim, Version 0.12.0) [17] to simulate the Pentesting Environment. In order to compare our work with [9], we have chosen the "small-honeypot" network.

The topology of our experimental network is shown in Figure 3. The network consists of four subnetworks with a total of eight hosts, one of which is a honeypot. The hosts run two types of operating systems (Linux and Windows) and three types of services (HyperText Transfer Protocol-HTTP, Secure SHell-SSH, and File Transfer Protocol-FTP). Hosts (2, 0) and (4, 0) are valuable assets (i.e., sensitive hosts) in the network, each assigned a reward value of 100. Node (3, 2) is a honeypot machine with a value of -100. The RL agent is expected to avoid exploiting honeypots. Firewalls filter specific types of services between subnets, and each action makes a cost for the agent.

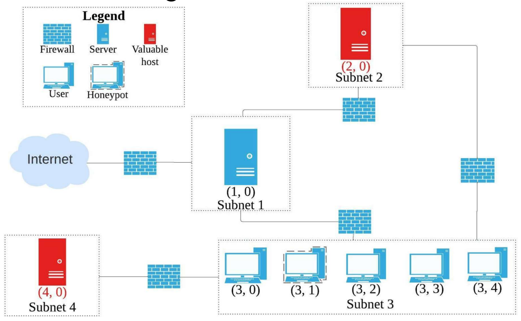

Figure 3. Topology of the experimental network.

The agent aims to maximize its score by reaching the sensitive hosts while minimizing cost. There are four types of scanning actions for getting information from each host: OSScan, Service Scan, ProcessScan, and SubnetScan. In addition, the agent can perform exploitation and privilege escalation actions for specific services and processes. The total number of actions is 72 and states is 24576. The available actions are listed in Table I, with each action associated with a different cost. Moreover, each action has a probability of success, indicating the difficulty of exploiting certain vulnerabilities. However, whether an action succeeds depends not only on this probability but also on factors such as firewall rules, the network topology, and the host's configuration.

TABLE I. Actions that can be taken by agents

Action name	OS	Cost	Probability	Access
SSH-EXP	Linux	3	0.9	User
FTP-EXP	Windows	1	0.6	User
http-EXT	/	2	0.9	User
Tomcat-PE	Linux	1	1	Root
Daclsvc-PE	Windows	1	1	Root
Subnet-Scan	/	1	1	/
OS-Scan	/	1	1	/
Service-Scan	/	1	1	/
Process-Scan	/	1	1	/

Table II lists the hyperparameters used for expert demonstration data generation using RL and agent training. A three-layer SimpleQ Network model was chosen for ISQL. A total of 1000 expert trajectories were extracted. To control the quality of the expert demonstrations, a reward threshold was used to filter the expert demonstrations data.

TABLE II. Hyperparameters for expert data generation and agemt training

Hyperparameter	Value
Learning rate	0.0001
Batch size	64
Discount factor,	0.9
Hidden layer size	128
Replay memory size	1000000
Initial memory size	10000
Target network update frequency	4
Initial temperature parameter	1
Max steps per episode	1000

B. *Evaluation Metrics*

We evaluated the performance of PT-ISQL from the perspectives of both imitation learning and automated pentesting. We first discuss the key factors that influence the performance of the ISQL algorithm in the context of pentesting in Sections C and D, then evaluate the proposed PT-ISQL approach according to the following three metrics.

Honeypot invasion probability. It is the likelihood that a pentesting agent is deceived into interacting with a honeypot. It serves as an indicator of the pentesting approach's stealth and precision. A high probability suggests that the pentesting approach cannot well distinguish real targets from decoys, while a low probability indicates more accurate reconnaiss-ance and smarter exploitation strategies. Frequent hitting of honeypots implies a low ability to uncover real vulnerabilities. In our tests, a honeypot is considered invaded if the returned reward is less than -100. In such cases, the invasion probability is set to 1.0 (100%); otherwise, it is 0. The average honeypot invasion probability is computed over 10 episodes (i.e., pentesting rounds) for each evaluation.

Average reward. In ISQL, an agent that receives a high cumulative reward is likely following expert-level strategies, taking efficient and goal-directed actions while avoiding risky or low-value behaviors. Reward accumulation directly reflects several aspects of performance: success rate (i.e., whether the goal is reached), efficiency (i.e., fewer steps to reach the goal) and stealthiness (i.e., fewer alerts triggered or honeypots invaded). Therefore, we use the average cumulative reward of 10 episodes to measure the performance of the proposed PT-ISQL approach.

Goal-reached probability. The goal of our tests in the simulated network is to reach the valuable hosts (2, 0) and (4, 0). Whether the goal is reached or not is recorded after each episode. If the goal is reached, the probability is set to 1.0 (100%); otherwise, it is 0. The goal-reached probability is calculated as the average value over 10 episodes.

In our tests, the learning steps are set to 20000 as default, and the evaluation interval is set to 200 steps. But for the tests in Section E, the numbers are increased to 100000 and 1000, respectively, to accommodate the extended training requirements of deep reinforcement learning. We describe our experiments and analyze the results in Sections C to E below.

C. Influence of the Threshold of Expert Demonstrations

To study the influence of expert demonstration quality on the performance of PT-ISQL, we use a threshold to filter the expert demonstration data. We observed that when the threshold is reduced to 21, the agent's mean reward is considerably lower than that of the expert demonstrations. To illustrate the influence of this threshold, we compared the results using two values: 21 (low threshold) and 100 (high threshold). For each threshold, varying numbers of expert demonstrations were used to train the agent. The average reward and standard deviation are calculated after 2000 steps, when all rewards had converged.

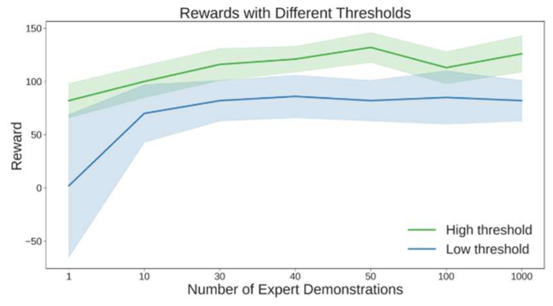

Figure 4. Rewards under different thresholds (solid lines-average values).

Figure 4 shows the rewards with different numbers of expert demonstrations under two threshold settings. When the reward threshold is low (21), the average reward and standard deviation of all the experiments are 98.80 and 41.14, respectively. When the threshold is high (100), the average reward and the standard deviation improved to 134.86 and 21.23, respectively. These results indicate that higher-quality expert data (i.e., higher threshold) leads to both higher average rewards and more stable performance, regardless of the number of demonstrations used. However, even in the high-threshold case, the agent's mean reward remains lower than that of the expert data.

D. Number of Reruired Expert Demonstrations

Due to the difficulty of obtaining expert data in practice, the minimum required number of expert demonstrations is an important factor affecting the usability of IL algorithms. To investigate this in the context of our PT-ISQL, we measured the reward when using different numbers of expert demonstrations: 1, 10, 30, 40, 50, 100, and 1000, under both low and high threshold settings. We observe the minimum number of expert demonstrations when the reward reaches a stable required value.

As shown in Figure 5, the rewards converge around 2000 steps in all settings. When the number of expert demonstrations is 1, the rewards are low and fluctuate heavily. Table III presents the average reward with standard deviation for different numbers of expert demonstrations after convergence (2000 steps). The results show that when the number of expert demonstrations exceeds 30, the performance becomes stable and consistent.

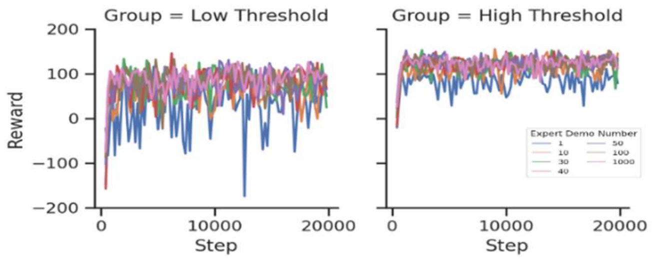

Figure 5. Rewards of different number of expert demonstrations.

TABLE III. AVERAGE REWARD WITH STANDARD DEVIATION UNDER DIFFERENT NUMBER OF EXPERT DEMONSTRATGIONS

No. Expert demo	Rewards - low threshold	Rewards - high threshold
1	2 ± 67	82 ± 16
10	70 ± 27	100 ± 15
30	82 ± 19	116 ± 15
40	86 ± 20	121 ± 12
50	82 ± 19	132 ± 14
100	85 ± 25	113 ± 15
1000	82 ± 19	126 ± 17

To further analyze convergence speed, we examined the relationship between the number of episodes completed and the number of training steps. After convergence (around 2000 steps), a higher number of episodes within a fixed number of training steps indicates faster convergence and thus a shorter duration for completing the automated pentesting task.

As shown in Figure 6, under the low-threshold setting, using 50 expert demonstrations results in a convergence speed nearly identical to that achieved with 100 or even 1000 expert demonstrations. Under the high-threshold setting, the number of expert demonstrations can be reduced to 40 while still achieving the convergence speed of the 100 and 1000 demonstration cases. Additionally, when completing 100 episodes, the time required with 30 expert demonstrations under the high-threshold condition is shorter than that under the low-threshold condition.

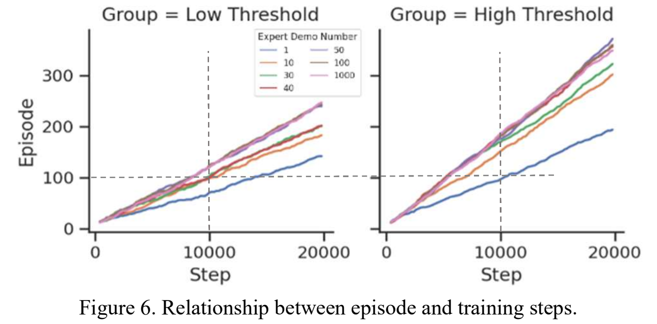

Figure 6. Relationship between episode and training steps.

In contrast, GAIL-PT requires 5000 expert demonstrations to reach the minimum number of training rounds in the same simulated network with a honeypot [6]. This shows that the number of required expert demonstrations in our proposed PT-ISQL is greatly fewer than that in GAIL-PT, which relies on generative adversarial learning and is also dependent on expert data. With our PT-ISQL, 30 expert demonstrations are enough to achieve good training performance in the simulated network with a "small honeypot". Furthermore, increasing the number or the quality threshold of expert demonstrations can further increase the converging speed of training.

E. *Comparison of ISQL with Simple Q-Learning*

To demonstrate the advantages of the ISQL algorithm in our PT-ISQL approach, we compared the pentesting performance using ISQL with that using Simple Q-Learning (a reinforcement learning method). The three pentesting metrics were examined across varying training steps. In this experiment, the high-threshold expert data was used, with 50 expert demonstrations provided. For each algorithm (with ISQL denoted as iq, and Simple Q-Learning as rl), five runs were conducted, and the results were averaged for the comparison of each metric.

Honeypot Invasion Probability

Figure 7 illustrates the honeypot invasion probability of five runs. The solid line represents the mean value, with the shaded area denoting the standard deviation. The results show that using ISQL greatly reduces the probability of honeypot invasion compared with deep reinforcement learning. This finding aligns with the results reported in DQfD-AIPT [11], where using expert demonstrations also led to significantly fewer interactions with honeypots. Nevertheless, in their study, the agent interacted with the honeypot during the early stages, i.e., before convergence.

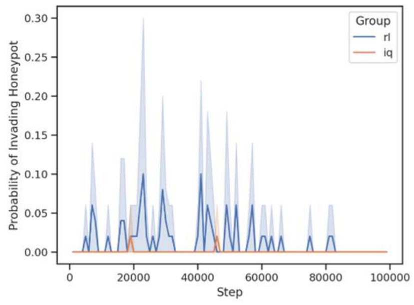

Figure 7. Probability of honeypot invasion of ISQL and Simple Q-Learning.

Average Reward

As shown in Figure 8, the rewards obtained using ISQL are both high and stable from the early stage of training. In addition, the reward is consistent across runs as indicated by the small shaded area. In contrast, when using the reinforcement learning algorithm, the reward is highly unstable across different runs. In fact, some runs fail to converge even after 100000 steps. This trend is consistent with the results reported in [9] and [11].

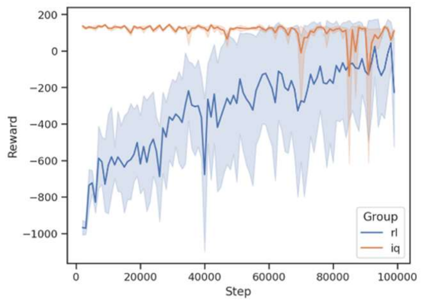

Figure 8. Rewards of ISQL and Simple Q-Learning.

Goal-Reached Probability

Figure 9 shows the average goal-reached probability with standard deviation (shaded area) over five runs using both algorithms. The results demonstrate that ISQL achieves much higher goal-reaching performance and requires far fewer training steps compared with Simple Q-Learning. ISQL achieves high goal-reaching performance even from the early stage of training, as it can quickly learn effective strategies from expert demonstrations.

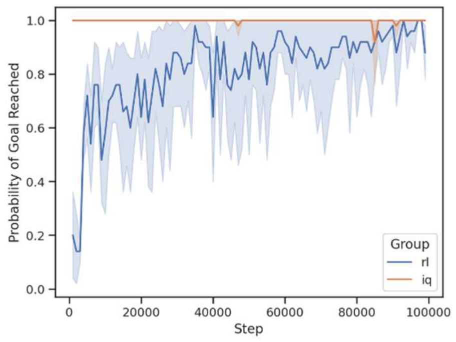

Figure 9. Goal reached probability of ISQL and Simple Q-Learning.

However, the performance of ISQL may degrade with excessive training. As seen in Figure 8 and Figure 9, after around 20000 steps, the reward begins to fluctuate more, and the goal-reaching rate declines. In contrast, Simple Q-Learning shows slower and less stable learning. Even after 100000 steps, the algorithm has not fully converged: the continued upward trend in the goal-reaching probability indicates that learning is still in progress.

V. CONCLUSIONS AND FUTURE WORK

In this paper, we proposed an automatic pentesting approach based on ISQL. Our approach uses soft Q-Learning to infer the reward function implicitly optimized by human experts, while requiring significantly less expert data compared with other reinforcement learning methods based on expert demonstrations. Evaluation results show that our PT-ISQL approach is much faster than that of the general deep reinforcement learning method, such as Simple Q-Learning. The performance of the trained PT-ISQL agent is comparable to that of human experts. The required number of expert demonstrations is largely reduced compared with GAIL-PT (50 vs 5000), making PT-ISQL a more data-efficient and practical solution for automated pentesting.

However, the experiments conducted in the paper are limited in a simulation environment with a small network, and the trained agent's transferability to different situations has not been assessed. Future work is to evaluate PT-ISQL in a more realistic simulation environment and to test it in real-world networks. This includes training agents on real expert demonstrations data, and integrating PT-ISQL with frameworks such as Deep Exploit, with the goal of making PT-ISQL a fully functional and deployable automated pentesting tool. Qualitative comparisons with LLM-based agents, such as PentestGPT, is also a future work.

REFERENCES

[1] G. Deng et al., “PentestGPT: Evaluating and Harnessing Large Language Models for Automated Penetration Testing,” In proc. of 33rd USENIX Security Symposium (USENIX Security 24), pp.847–864.

[2] X. Shen et al., “Pentest Agent: Incorporating LLM Agents to Automated Penetration Testing”, arXiv:2411.05185v1 [cs.CR], Nov. 7, 2024.

[3] A. Happe and J. Cito, “Getting pwn’d by AI:penetration testing with large language models,” In Proc. of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, pp.2082–2086.

[4] H. Kong et al., “VulnBot: Autonomous penetration testing for a multi-agent collaborative framework”, arXiv:2501.13411v1 [cs.SE] 23 Jan. 2025

[5] Z. Hu, R. Beuran, and Y. Tan, “Automated penetration testing using deep reinforcement learning,” in 2020 IEEE European Symposium on Security and Privacy Workshops (EuroSPW), pp. 2-10, IEEE, 9 2020.

[6] I. Jabr, Y. Salman, M. Shqair, and A. Hawash, “Penetration testing and attack automation simulation: deep reinforcement learning approach,” An-Najah University Journal for Research, Apr. 2024, pp. 7-14. DOI:10.35552/anujr.a.39.1.2231

[7] J. Yi and X. Liu, “Deep reinforcement learning for intelligent penetration testing path design,” Applied Sciences, vol. 13, p. 9467, Aug. 2023.

[8] L. V. Hoang et al., “Leveraging deep reinforcement learning for automating penetration testing in reconnaissance and exploitation phase,” in Int. Conf. on Computing and Communication Technologies, pp. 41-46, IEEE, 12 2022.

[9] J. Chen, S. Hu, H. Zheng, C. Xing, and G. Zhang, “Gail-PT: An intelligent penetration testing framework with generative adversarial imitation learning,” Computers Security, vol. 126, p. 103055, 3 2023.

[10] F. M. Zennaro and L. Erdödi, “Modelling penetration testing with reinforcement learning using capture-the-flag challenges: tradeoffs between model-free learning and a priori knowledge” IET Information Security, vol.17, pp.441-457, 5 2023.

[11] Y. Wang et al., “Dqfd-aipt: An intelligent penetration testing framework incorporating expert demonstration data,” Security and Communication Networks, vol. 2023, pp. 1-15, 5. 2023.

[12] M. Zare, P. M. Kebria, A. Khosravi, and S. Nahavandi, “A survey of imitation learning: Algorithms, recent developments, and challenges,” IEEE Transactions on Cybernetics, vol.54, pp. 7137-7168, Dec. 2024.

[13] X. Ou, S. Govindavajhala and A. W. Appel, “Mulval: A logic-based network security analyzer,” in Proc. of USENIX security symposium, vol. 8, pp. 113-128, 2005.

[14] T. Isao, “Deep exploit,” 2018. https://github.com/13o-bbr-bbq/machine_learning_security/blob/master/DeepExploit/README.md / [retrieved: Sept. 2025]

[15] G. Farquhar et al., “Growing action spaces,” in Proc. of the 37th International Conference on Machine Learning, vol. 119, pp. 3040-3051, PMLR, 5 2020.

[16] J. Ho and S. Ermon, “Generative adversarial imitation learning,” Advances in neural information processing systems, vol. 29, pp. 4572-4580, Dec. 2016.

[17] J. Schwartz and H. Kurniawati, “Autonomous penetration testing using reinforcement learning,” CoRR, vol. abs/1905.05965, 2019.