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, Sweden

Contact email: yh2025@dsv.su.se

1. Aims and Contributions

We aimed at:

- Building a compact, robust and reliable automated penetration testing (pentesting) system based on Inverse Soft-Q Learning (ISQL)

Contributions of our study:

- 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.

2. SOTA—Focus on RL & IL for Pentesting

❑ Pentesting with Reinforcement Learning (RL)

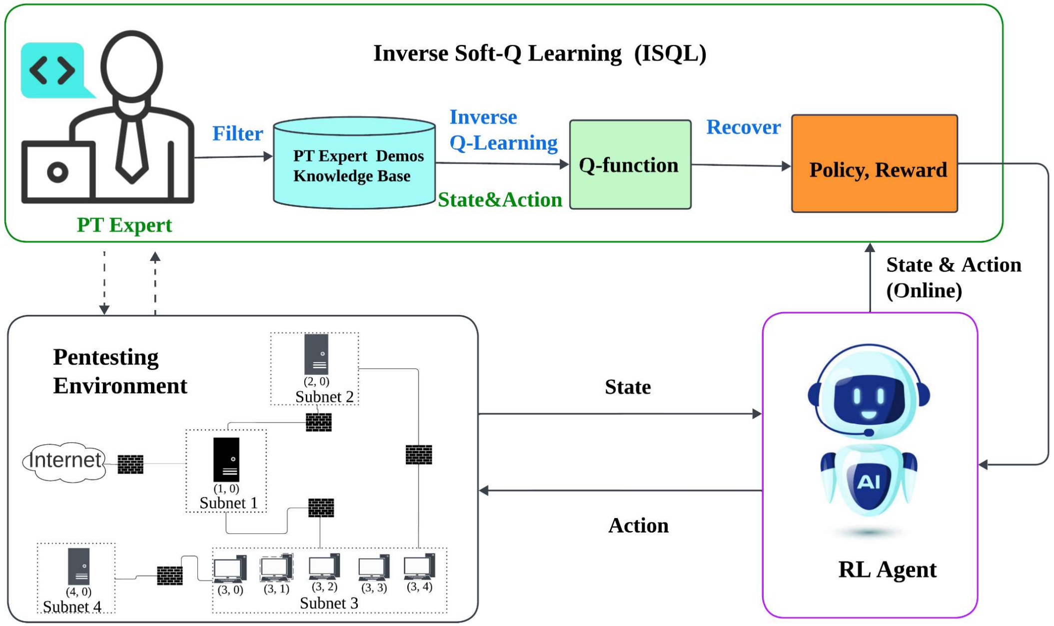
- **Core idea**: Agent learns via environment interaction (Agent → Action → State Change → Reward).
- **Examples**:
  - Deep Exploit (A3C algorithm): Automates server exploitation but struggles with large discrete action spaces.
  - DQN-based attack path optimization: Relies on known network topology (unrealistic for real-world use).
- **Key limitations**: RL's trial-and-error leads to instability; large state/action spaces slow convergence.

❑ Pentesting with Imitation Learning (IL): a specialized form of RL

- **Core idea**: Agent learns from expert demonstrations (infers reward/policy).
- **Examples**:
  - GAIL-PT: Combines GAIL + Deep Exploit but needs hyperparameter tuning and 5000 expert demos.
  - DQfD-AIPT: Reduces overfitting but requires massive expert data.
- **Key limitation**: large amount of expert data are needed to build the expert database

❑ **Our Advantage**: PT-ISQL uses ISQL to avoid complex adversarial training while minimizes the needed expert data.

3. PT-ISQL —System Architecture



Three Core Components

- **ISQL Component (Core)**
  - Input: Expert demonstrations (state-action pairs: e.g., scan, exploit, privilege escalation).
  - Output: Learned soft Q-function (implicitly captures reward + policy).
- **RL Agent**
  - Uses ISQL-learned policy to act autonomously (e.g., discover attack paths, avoid honeypots).
  - Adapts to dynamic defenses (e.g., patched systems, IDS).
- **Pentesting Environment**
  - Returns state info (host config, vulnerabilities) after agent/expert actions.
  - In our paper, we used a simulated network (e.g., NASim "small-honeypot" topology: 4 subnets, 8 hosts, 1 honeypot).

3. PT-ISQL —ISQL Workflow (1)

Step 1: Process Expert Demonstrations

- Extract state-action pairs (s, a) from expert trajectories (D_{expert}).
- Filter demos via reward threshold (e.g., high threshold = 100 for quality control).

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 γ, entropy temperature α, learning rate η, 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

Initialize Q-function $Q_\theta(s, a)$ with parameters θ

for $t = 1$ to T **do**

  Sample batch $\{(s, a, s')\} \sim \mathcal{D}_{\text{expert}}$

  Compute soft values: $V(s') \leftarrow \alpha \cdot \log \sum_{a'} \exp(Q_\theta(s', a')/\alpha)$

  Compute reward: $\hat{r} \leftarrow (Q_\theta(s, a) - \gamma V(s'))$

  Compute loss:

$$\mathcal{L} \leftarrow -E[\hat{r}] + E[Q_\theta(s, a) - \gamma V(s')] + \frac{1}{4\alpha} E[\hat{r}^2]$$

  Update Q-function parameters: $\theta \leftarrow \theta - \eta \nabla_\theta \mathcal{L}$

end for

3. PT-ISQL —ISQL Workflow (2)

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 γ, entropy temperature α, learning rate η, 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

```
Initialize Q-function Q_θ(s, a) with parameters θ
for t = 1 to T do
    Sample batch {(s, a, s')} ~ D_expert
    Compute soft values: V(s') ← α · log Σ_{a'} exp(Q_θ(s', a')/α)
    Compute reward: r̂ ← (Q_θ(s, a) − γV(s'))
    Compute loss:
        L ← −E[r̂] + E[Q_θ(s, a) − γV(s')] + (1/4α) E[r̂²]
    Update Q-function parameters: θ ← θ − η∇_θ L
end for
```

$$\mathcal{L} \leftarrow -E[\hat{r}] + E[Q_\theta(s, a) - \gamma V(s')] + \frac{1}{4\alpha} E[\hat{r}^2]$$

Step 2: Iterative ISQL Training

- **Goal**: Learn a soft Q-function that aligns with expert behavior.
- **Key Equations**:

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

  r = reward, γ = discount factor, α = entropy temperature, π = policy

  Policy: $\pi(a|s)$ = softmax $(Q(s,a)/\ \alpha$

- **Training Loop:**
  - Initialize Q-function (Q_θ).
  - Sample batch from D_{expert}.
  - Compute loss (MSE between predicted/ target Q-values).
  - Update Q_θ via gradient descent.

Step 3: Agent Execution

- Agent uses learned π to perform pentesting tasks (e.g., reach sensitive hosts, avoid honeypots).

4. Evaluation —Experiment Setup

Network, tools and evaluation metrics

Environment & Tools

Platform: Kali Linux VM + **MiniConda** (Python 3.x).

Simulator: NASim v0.12.0 (“small-honeypot” network).

Network Topology:

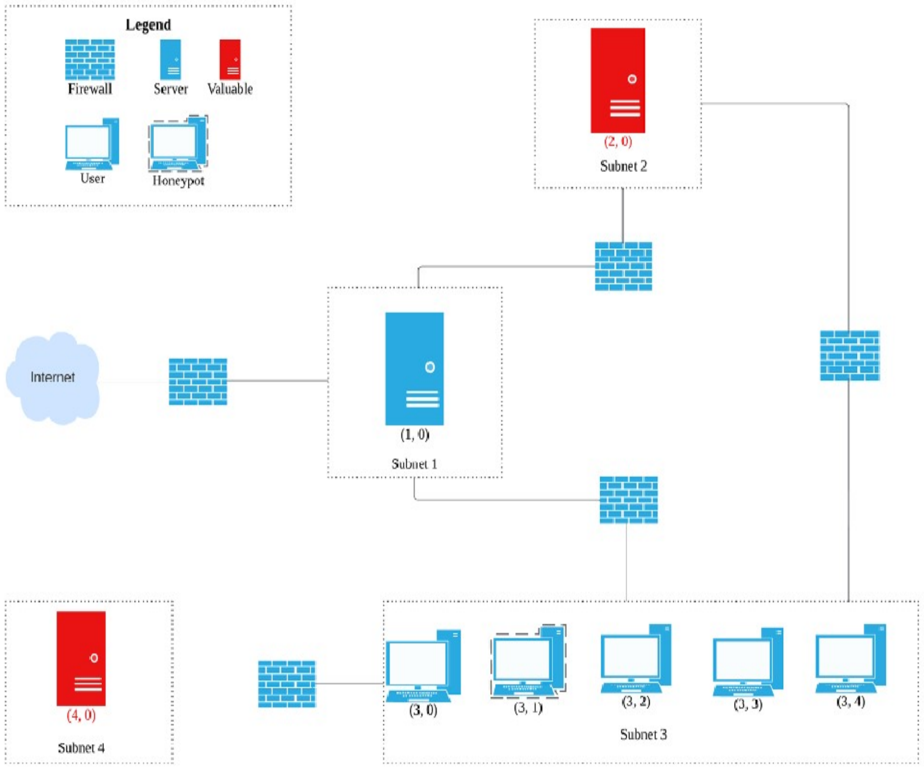
- 4 subnets, 8 hosts (Linux/Windows), 3 services (HTTP, SSH, FTP).
- Sensitive hosts: (2,0) and (4,0) (reward = 100).
- Honeypot: (3,2) (penalty = -100).

Evaluation Metrics

Honeypot Invasion Probability: Likelihood of agent interacting with honeypot (lower = better).

Average Cumulative Reward: Reflects efficiency/stealth (higher = better).

Goal-Reached Probability: Likelihood of reaching sensitive hosts (higher = better).



4. Evaluation —Key Result (1)

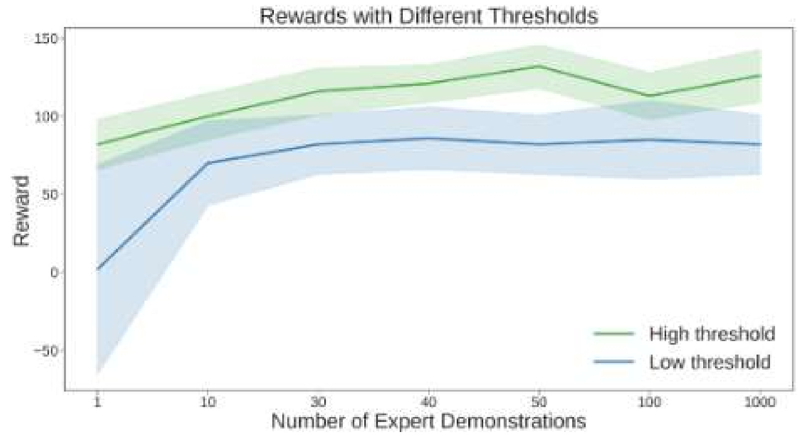
Expert Demo Quality & Quantity

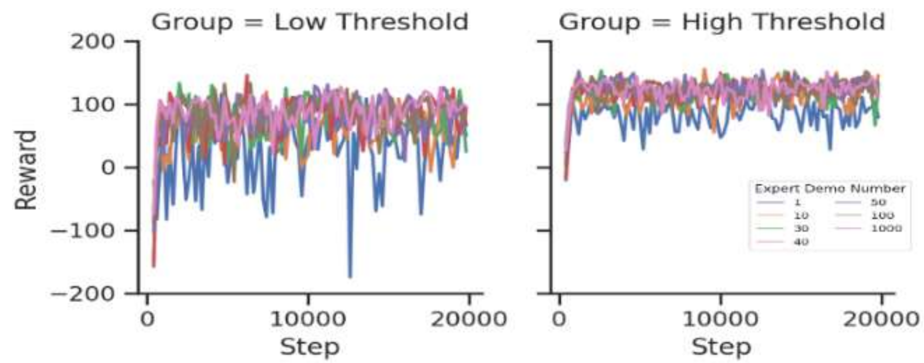
❑ Influence of Demo Threshold

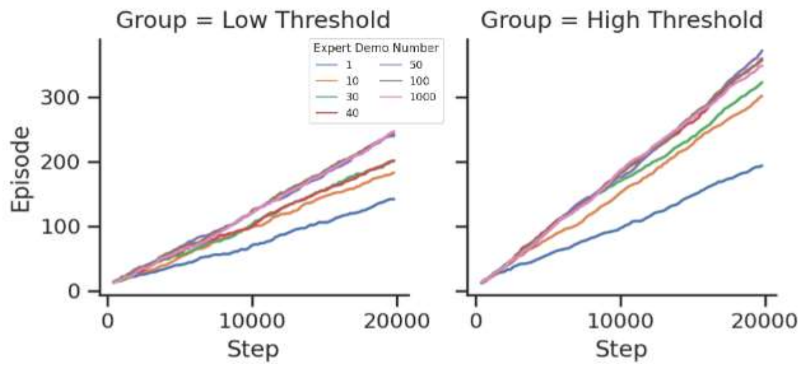
- **Low Threshold (21)**: Avg. reward = 98.80 ± 41.14 (high variance).
- **High Threshold (100)**: Avg. reward = 134.86 ± 21.23 (higher reward, lower variance).
- **Conclusion**: Higher-quality demos (high threshold) improve stability.

❑ Minimum Demo Quantity

- **Stability Threshold**: ≥30 demos (high threshold) → stable reward (116 ± 15).
- **Comparison to GAIL-PT**: PT-ISQL needs 30 demos vs. GAIL-PT's 5000.
- **Conclusion**: PT-ISQL is highly data-efficient.



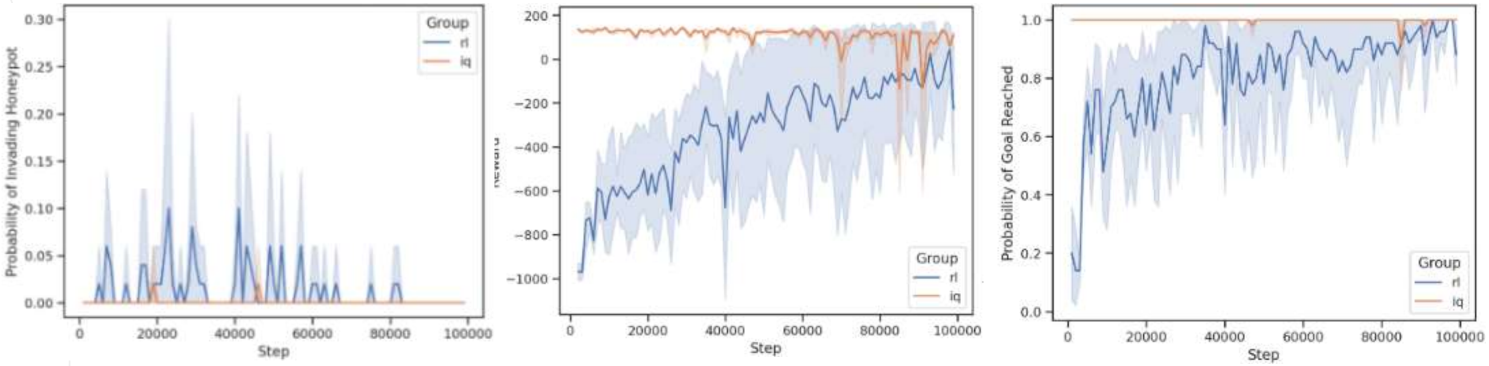




4. Evaluation — Key Result (2)

ISQL vs. Simple Q-Learning

- ❑ **Metric comparisons**
- ❑ **Convergence Speed**
  - PT-ISQL converges by 20,000 steps; RL still learns at 100,000 steps.



Honeypot probability, Average reward, Goal probability vs. Training steps

Metric	PT-ISQL (ISQL)	Simple Q-Learning (RL)
Honeypot Invasion Prob.	~0.05 (early convergence)	~0.25 (unstable)
Avg. Reward	132 ± 14 (stable)	Fluctuates (some runs fail)
Goal-Reached Prob.	~0.9 (by 20,000 steps)	~0.6 (after 100,000 steps)

5. Conclusion and Future Work

Conclusion:

- PT-ISQL uses ISQL to automate pentesting with minimal expert data.
- Outperforms RL in convergence speed, stability, and task performance.
- Reduces honeypot interactions and improves goal achievement.

Future work:

- Validate PT-ISQL in larger simulated networks and real-world environments.
- Integrate PT-ISQL with tools like Deep Exploit for deployable use.
- Conduct qualitative comparisons with LLM-based agents.

References (1)

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.

References (2)

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.