Learning Insider-Threat Intervention Policies from Offline Logs

Cyrine Fekih [1] **Ashish Sai** [1] **Adriana Iamnitchi** [1] **Dennis J. N. J. Soemers** [1]

Abstract

Insider threats are among the hardest threats to detect, as malicious behaviour is hidden within legitimate user activity. Unlike classification approaches that label each observation independently, we frame insider threat prevention as a reinforcement learning problem where an agent monitors user behaviour over fixed time windows and decides whether to continue monitoring or block the user. A key property of our setting is that ground-truth labels allow us to simulate the consequence of both continuing and blocking at every time step, enabling us to train a Deep Q-Network variant on a pre-constructed dataset of transitions and Proximal Policy Optimisation on a fully simulated environment. Preliminary results across five user-level folds show detection rates of 42–64% of malicious users, with the agent typically intervening within 0–2 time windows of the first malicious activity. False positives remain a challenge, highlighting the need for improved reward design, state representations, and richer intervention actions.

1. Introduction

Not all cyberattacks originate from outside an organisation. Some are carried out by individuals with authorised access, making insider threats particularly difficult to detect. Malicious behaviour may initially resemble normal user activity and only become distinguishable over time, by which point significant damage may already have occurred (Yuan & Wu, 2021). Given their severity and difficulty of detection, insider threat detection has been the focus of many studies in recent years. Existing work has explored several machine learning and deep learning techniques, from classical methods such as SVMs, decision trees, and Gaussian mixture models to neural approaches including recurrent neural networks, convolutional neural networks, and graph neural networks (Al-Mhiqani et al., 2020; Yuan & Wu, 2021).

Al-Mhiqani et al. (2020) categorise insider-threat detection methodologies mainly into two groups: (1) anomaly detection or classification, where the model learns a normal behaviour and flags any activities that deviate from the baseline; and (2) Signature-based or Rule-based detection, where malicious activity is recognised by a predefined signature. However, in an operational cybersecurity setting, detecting suspicious behaviour is not sufficient on its own. Decisions such as blocking a user have long-term organisational consequences that classification-based approaches do not model. Moreover, the timing of intervention matters: acting too early disrupts legitimate work, while acting too late allows damage to accumulate. This motivates formulating insider threat prevention as a sequential decision-making problem, where a system observes user behaviour over time and decides whether to continue monitoring or intervene. Reinforcement learning (RL) naturally aligns with this setting, as the agent learns a policy by observing states, taking actions, and receiving rewards with the objective of maximising cumulative reward (Sutton & Barto, 2018). Yuan & Wu (2021) also highlight the relevance of using RL for insider-threat detection, given its ability to learn adaptive policies in complex and dynamic environments.

In standard RL, the agent usually improves its policy through repeated interaction with the environment, either by updating the policy from newly collected trajectories or by storing past experiences in a replay buffer and learning from them. However, this learning process requires deploying an untrained agent in a live environment to collect experience, which can be costly and risky, especially in a cybersecurity setting. Offline RL addresses this limitation by training an agent in previously collected data without requiring direct online interaction (Levine et al., 2020). However, one of the main challenges in offline RL is the distribution shift problem, where learning reliable values becomes difficult when some state-action pairs are poorly represented in the training data. A resulting problem is the bootstrapping error, where value estimates for out-of-distribution actions propagate through the Q-function and degrade learning (Levine et al., 2020; Kumar et al., 2019).

In this paper, we introduce an RL framework for insider-

[1]Department of Advanced Computing Sciences, Maastricht University, Maastricht, the Netherlands. Correspondence to: Cyrine Fekih <cyrine.fekih@maastrichtuniversity.nl>.

Accepted at ICML 2026 Workshop on Decision-Making from Offline Datasets to Online Adaptation: Black-Box Optimization to Reinforcement Learning, Seoul, South Korea.

threat prevention. Based on the CERT dataset (Lindauer, 2020), we define a Markov decision process (MDP) in which an agent observes user activity over time and decides whether to continue monitoring or block the user. The agent learns an intervention policy that aims to identify the appropriate time to act, balancing early threat prevention against unnecessary disruption of benign users.

A key property of our setting is that ground-truth labels allow us to determine the outcome of either action at any step. This lets us circumvent the distribution shift and bootstrapping error problems that typically constrain offline RL. We exploit this to train an offline version of DQN (Mnih et al., 2013) on a self-constructed dataset of all possible transitions, and PPO (Schulman et al., 2017) on the resulting simulated environment. We evaluate both approaches across five user-level folds and present preliminary results on detection rates, response delay, and false positive rates.

2. RL Framework for Insider Threat

Insider threats are particularly challenging to detect because insiders already hold authorised access, making their malicious actions difficult to distinguish from normal activity. Insiders are categorised into *traitors*, who misuse their own privileges; *masqueraders*, who impersonate legitimate employees to perform unauthorised actions; and *unintentional perpetrators*, who cause harm through mistakes, or unsafe behaviour (Al-Mhiqani et al., 2020; Yuan & Wu, 2021). Given this, insider threat detection research has largely focused on analysing user behaviour over time.

Al-Mhiqani et al. (2020) classify behavioural evidence into several categories, including biometric, cyber-activity, psychosocial, physical, and combined behaviours. In this work, we focus on cyber-activity behaviour, which is the most widely studied as it directly reflects how users interact with organisational systems, such as login activity, email exchanges and system logs. However, detecting insider threats from behavioural data remains a difficult problem. Malicious activity is extremely rare relative to benign activity, creating a severe class imbalance that challenges standard learning algorithms (Yuan & Wu, 2021). Moreover, insider behaviour evolves gradually over time. A single suspicious action may appear normal in isolation, and it is only the pattern across days or weeks that reveals malicious intent (Al-Mhiqani et al., 2020; Yuan & Wu, 2021).

2.1. Dataset

This work is based on the CERT insider-threat dataset (Lindauer, 2020), a synthetic dataset widely used for insider-threat research that provides multiple sources of user activity including logon/logoff events, device usage, file access, email exchanges, and HTTP activity, along with LDAP and psychometric information (Yuan & Wu, 2021; Homoliak et al., 2019). Among the six released versions, we use r4.2, which is the densest with 70 malicious users and 930 benign users. The malicious users span three insider-threat scenarios, such as employees leaking confidential information or stealing data when leaving the organisation (see Table 3 for details).

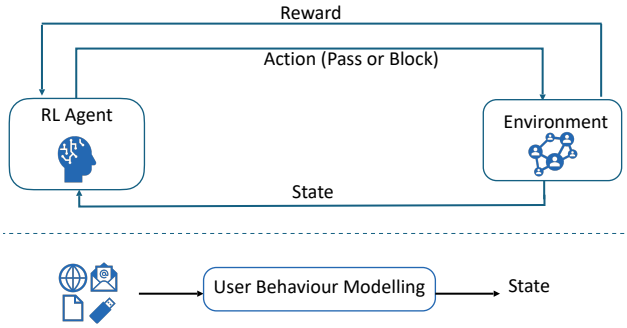

Figure 1. RL framework for insider threat prevention.

2.2. Problem Formulation

Inspired by the real-world setting, we define the insider threat prevention as an MDP where an agent observes user activities and takes an action accordingly. Each episode corresponds to the activity timeline of one user, and each time step corresponds to a fixed time window $\Delta\tau$. At each step t, the agent observes a summary of user activities s_t and chooses to `PASS` or `BLOCK`. Based on the action taken a_t, the environment returns a positive reward if it is a correct block or a correct pass; otherwise, it returns a negative reward. The framework is represented in Figure 1.

Action: The action space is $A = \{\texttt{PASS}, \texttt{BLOCK}\}$ where `PASS` advances the environment from s_t to s_{t+1} and `BLOCK` terminates the episode. More actions can be added like *send warning*, *block access to certain files*, *flag user*, however, in this work we only include two types of action.

State: The state s_t summarises a user's behaviour within a fixed time window through 32 features designed to capture deviations from normal work patterns. These features can be categorised into four groups: (1) Activity counts measure the volume of user actions, such as file access, website visits, emails, and USB usage. (2) Activity ratios normalise these counts by total activity, capturing changes in behaviour composition, such as unusually high external email activity. (3) Temporal features encode when actions occur, including hour-of-day, weekend, and after-hours indicators. (4) Combination features capture co-occurrences that are individually normal but suspicious together, such as USB device usage after hours. The complete list of all 32 features is provided in Appendix B. Since insider behaviour evolves over time, features computed from a single window may not be sufficient to capture deviations from normal

activity. We therefore include rolling deviation features, computed as z-scores over recent windows, to measure how much current behaviour differs from the user's baseline. To extract these features, we aggregate all activity logs from the CERT dataset—file, email, logon, and device records—grouping them by user to obtain individual behavioural timelines. These timelines are then segmented into fixed time windows $\Delta\tau$ where $\Delta\tau \in \{$ 1 hour, 2 hours, ... 1 day ... $\}$. Each segment yields one state vector s_t. This feature extraction component can be seen as a user behaviour modelling module and can in principle be replaced by more sophisticated representations, such as learned embeddings from sequence models (Kong et al., 2025).

Reward: The reward function is designed to reflect the consequences of each action, encouraging early detection of threats and discouraging false positives. The agent receives a positive reward for correctly blocking a malicious user and a small positive reward for correctly continuing to monitor a benign user. In contrast, it receives a penalty for missing an ongoing threat or for disrupting a benign user by blocking them incorrectly. The reward values used in our experiments are defined as follows:

$$r(s_t, a_t) = \begin{cases} +0.1, & \text{if } a_t = \texttt{PASS} \text{ and } y_t = 0, \\ +50, & \text{if } a_t = \texttt{BLOCK} \text{ and } y_t = 1, \\ -30, & \text{if } a_t = \texttt{PASS} \text{ and } y_t = 1, \\ -20, & \text{if } a_t = \texttt{BLOCK} \text{ and } y_t = 0. \end{cases}$$

y_t is defined from the ground truth annotations provided in the CERT dataset. After splitting each user timeline into fixed windows of length $\Delta\tau$, we assign a binary label y_t to each window. A window is labelled malicious ($y_t = 1$) if its time interval overlaps with a ground-truth malicious activity period for the user; otherwise, it is labelled benign ($y_t = 0$).

2.3. Algorithms and Challenges

Modelling insider-threat mitigation as an MDP is useful from an operational perspective because it allows the agent to learn when intervention is justified. However, this formulation also introduces several learning challenges.

First, malicious behaviour is highly sparse and rare. Although balancing the number of benign and malicious users can reduce user-level class imbalance, the number of malicious windows remains limited. The malicious-window ratio represents at most around 12% of the activity windows within malicious-user trajectories, while in most cases, users have less than 5% malicious activity (see Figure 2). Second, the `BLOCK` action is terminal. Once the agent blocks a user, the episode ends. This makes exploration difficult during training, as an early block prevents the agent from observing the rest of the user trajectory. Third, malicious activity often appears late in the user trajectory (see Figure 4). This makes learning more difficult because an agent that blocks too early may never observe the malicious part of the episode, while an agent that waits too long may miss the opportunity for timely intervention. Therefore, the reward function must balance three competing objectives: avoiding unnecessary disruption of benign users, preventing delayed response to malicious activity, and encouraging the agent to gather enough evidence before blocking and avoiding a degenerate safe policy where the agent repeatedly selects `PASS` only to accumulate small positive rewards.

To address these challenges, we use two design choices. First, the reward values are chosen so that, on typical malicious trajectories, the cumulative return of repeatedly selecting `PASS` is lower than the return obtained by blocking at the appropriate time. This prevents the agent from learning a trivial policy that always continues monitoring to collect small positive rewards from benign windows. Second, we train a DQN-based agent (Mnih et al., 2013) using an offline transition dataset generated from the user trajectories and evaluate it in the constructed RL environment. In standard DQN, the agent interacts with the environment, stores transitions in a replay buffer, and updates the Q-function from sampled transitions of the form $(s_t, a_t, r_{t+1}, s_{t+1})$. However, in our setting, online exploration can be problematic because an early `BLOCK` action terminates the episode and may prevent the agent from observing later malicious behaviour. To avoid this issue, we generate transitions for all observed states in the dataset. For each state s_t, we construct both possible action outcomes: selecting `PASS` advances the trajectory to the next time window, while selecting `BLOCK` terminates the episode with a reward determined by the current label. This produces a fixed offline dataset containing both actions for each state, allowing the Q-function to be trained without relying on a limited replay buffer collected through potentially premature exploration.

3. Preliminary Results

To evaluate the proposed formulation, we conduct preliminary experiments with two RL baselines: PPO (Schulman et al., 2017) as on-policy method trained on the simulated environment, and our offline DQN variant as value-based off-policy method trained on generated transitions. The data is split at the user level to avoid information leakage between training and testing trajectories, with 800 users for training (56 malicious) and 200 for testing (14 malicious). During PPO training, episodes are sampled with a balanced ratio of benign and malicious users to reduce user-level class imbalance. To validate the robustness of both approaches, we define five non-overlapping folds, where each fold corresponds to a different arrangement of the train-test split such that every user appears in exactly one test fold.

We first evaluate the learned policies at the user level. A malicious user is counted as detected (true positive) if the agent

Table 1. Offline DQN threat-prevention results over five user-level folds. False Positive-M refers to malicious users blocked before their first labelled malicious window, while False Positive-B refers to benign users incorrectly blocked.

Folds	Detected	Missed	False Positive-M	False Positive-B	True Passes
Fold 1	7 (50.0%)	2	5	57	129 (69.3%)
Fold 2	9 (64.3%)	3	2	46	140 (75.3%)
Fold 3	8 (57.1%)	1	5	38	148 (79.6%)
Fold 4	9 (64.3%)	1	4	44	144 (77.4%)
Fold 5	6 (42.9%)	2	6	23	163 (87.6%)

blocks during a malicious window, and as missed (false negative) if the agent never blocks. A malicious user blocked during a benign window is counted as a false positive. For benign users, blocking represents a false positive, while never blocking represents a true negative. Tables 1 and 2 report the evaluation results of offline DQN and PPO, respectively. Offline DQN detects between 42.9% and 64.3% of malicious users across folds, while PPO detects between 21% and 64%. Averaging across all five folds, offline DQN detects 39 out of 70 malicious users (55.7%) compared to 26 out of 70 for PPO (37.1%). Both models learned to capture malicious patterns, but offline DQN shows better performance than PPO. We also test a standard online DQN model with ϵ-greedy exploration. However, the agent fails to learn a meaningful policy as blocking terminates the episode and random exploration leads to frequent premature terminations that prevent the agent from observing enough of the user's timeline to associate behavioural patterns with rewards. PPO, while also exploring, is less affected, likely due to its ability to automatically adapt its degree of exploration based on state. Our offline DQN variant avoids this problem entirely by training on pre-constructed transitions that cover all the possible transitions at every step. The agent can therefore learn the value of blocking and continuing without needing to explore the environment at all, which may explain its better performance in malicious user detection. However, its false-positive rate remains high, varying from 15% to 33%. This may be attributed to the reward balance between capture reward and disruption cost, which currently encourages blocking, or to limitations in the feature representation for certain benign patterns.

A deeper analysis of offline DQN's detection timing shows that the agent blocks malicious users within 0–2 time windows (median) of the first malicious activity (see Appendix C). This indicates that when the agent does detect a malicious user, it acts at the earliest opportunity.

4. Limitations and Future Work

This work presents an initial framework for proactive insider threat prevention using RL, which has several limitations and directions for improvement. First, the state representation relies on hand-crafted features derived from cyber-activity logs, excluding psychometric indicators and organisational context, which could significantly improve detection of certain threat scenarios. Deep feature extraction methods could help learn more informative representations directly from raw activity data. Second, the temporal context is currently captured through z-score normalisation of certain features, which provides a limited view of behavioural evolution over time. Recurrent architectures such as LSTMs, or attention-based models, could better capture long-range temporal dependencies in user behaviour. Third, the action space is limited to two actions, `BLOCK` or `PASS`, where blocking immediately terminates the episode. A more realistic action space could include intermediate responses such as increasing monitoring, sending a warning, or restricting access, which better reflects real-world security operations and gives the agent finer-grained control. Fourth, the reward function could be refined to more strongly penalise false positives, potentially through adaptive or context-dependent costs that account for the user's role. Finally, the core advantage of formulating insider threat prevention as an RL problem lies in its ability to model the downstream consequences of decisions. For example, blocking a user affects not only that individual but potentially the workflow of their team and the behaviour of other employees. This sequential decision-making perspective is fundamentally what distinguishes the RL approach from deep learning models, which classify each observation independently without considering the broader impact of the resulting action.

Table 2. PPO threat-prevention results over five user-level folds.

Fold	Detected	Missed	False Positive-M	False Positive-B	True Passes
Fold 1	4 (28.5%)	10	6	39	149 (80.1%)
Fold 2	3 (21.4%)	11	6	27	159 (85.5%)
Fold 3	6 (42.9%)	8	0	4	148 (79.6%)
Fold 4	9 (64.3%)	5	0	11	175 (94.1%)
Fold 5	4 (28.5%)	4	10	48	143 (76.9%)

5. Conclusion

We formulated insider-threat prevention as a sequential decision-making problem and showed that labelled detection datasets can be repurposed to construct an MDP. We modelled each user timeline as an episode and evaluated PPO and an offline DQN variant trained on generated transitions from the data-derived environment. Preliminary results across 5 folds show detection rates of 42–64% of malicious users, with the agent typically intervening within 0–2 time windows of the first malicious activity. False positives remain high, indicating that the reward function and state representation require further refinement. This work establishes a foundation for RL-based insider threat prevention. Future work aims to improve state representations, refine the reward function, evaluate the approach on larger datasets, and investigate richer intervention actions beyond blocking.

References

Al-Mhiqani, M. N., Ahmad, R., Zainal Abidin, Z., Yassin, W., Hassan, A., Abdulkareem, K. H., Ali, N. S., and Yunos, Z. A review of insider threat detection: Classification, machine learning techniques, datasets, open challenges, and recommendations. *Applied Sciences*, 10 (15), 2020. ISSN 2076-3417. doi: 10.3390/app10155208. URL https://www.mdpi.com/2076-3417/10/15/5208.

Homoliak, I., Toffalini, F., Guarnizo, J., Elovici, Y., and Ochoa, M. Insight into insiders and it: A survey of insider threat taxonomies, analysis, modeling, and countermeasures. *ACM Comput. Surv.*, 52(2), April 2019. ISSN 0360-0300. doi: 10.1145/3303771. URL https://doi.org/10.1145/3303771.

Kong, K., Liu, D., Jin, X., Li, Z., and Geng, G. Log2sig: Frequency-aware insider threat detection via multivariate behavioral signal decomposition. *arXiv preprint arXiv:2508.05696*, 2025.

Kumar, A., Fu, J., Tucker, G., and Levine, S. *Stabilizing off-policy Q-learning via bootstrapping error reduction*. Curran Associates Inc., Red Hook, NY, USA, 2019.

Levine, S., Kumar, A., Tucker, G., and Fu, J. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv:2005.01643*, 2020.

Lindauer, B. Insider Threat Test Dataset. 9 2020. doi: 10.1184/R1/12841247.v1. URL https://kilthub.cmu.edu/articles/dataset/Insider_Threat_Test_Dataset/12841247.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017. URL http://dblp.uni-trier.de/db/journals/corr/corr1707.html#SchulmanWDRK17.

Sutton, R. S. and Barto, A. G. *Reinforcement Learning: An Introduction*. A Bradford Book, Cambridge, MA, USA, 2018. ISBN 0262039249.

Yuan, S. and Wu, X. Deep learning for insider threat detection: Review, challenges and opportunities. *Computers & Security*, 104:102221, 2021. ISSN 0167-4048. doi: https://doi.org/10.1016/j.cose.2021.102221. URL https://www.sciencedirect.com/science/article/pii/S0167404821000456.

A. Dataset

This appendix provides additional detailed insights into the CERT r4.2 dataset. Table 3 summarises the core types of insider-threat scenarios that are present in the dataset. Figures 2 and 3 provide insight into the ratios of malicious-behaviour windows within malicious user timelines. Figure 4 depicts where malicious-behaviour windows tend to be positioned within malicious user timelines.

Table 3. Summary of CERT r4.2 insider-threat scenarios.

Scenario	Brief description	Number
S1	After-hours login, removable drive use, WikiLeaks upload, and departure.	30
S2	Job-site browsing, competitor contact, high thumb-drive use, and data theft before leaving.	30
S3	Disgruntled administrator installs a keylogger, impersonates a supervisor, sends a mass email, and leaves.	10

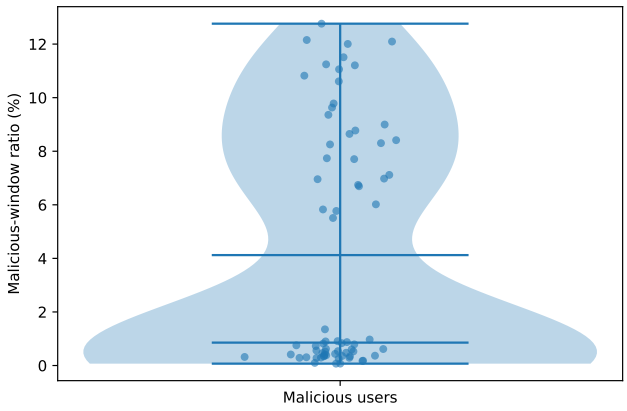

Figure 2. Distribution of malicious-window ratios when using 3-hour windows.

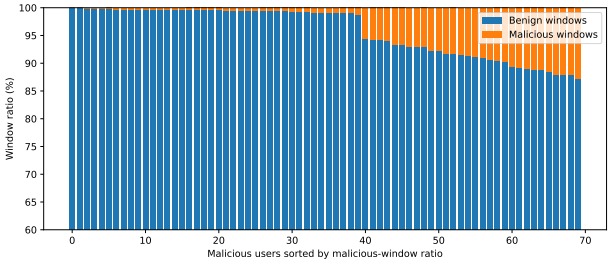

Figure 3. Benign vs malicious activity composition within malicious users (3-hour windows).

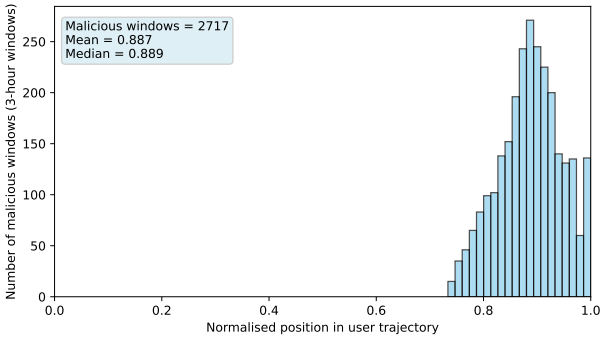

Figure 4. Position of malicious windows in malicious-user trajectories.

B. Feature Representation

Table 4 lists the complete set of 32 features used to represent the state s_t at each time window. Features are grouped into four categories: activity counts, activity ratios, temporal features, and combination features.

- Activity counts capture the volume of user's actions: how many files were accessed, websites visited, and USB devices connected. A sudden spike in file downloads or device usage, for example, may signal data exfiltration. To reduce skewness in these counts, we apply a log transformation $\log(1 + x)$ to all activity count features.
- Activity ratio normalise these counts relative to the user's total activity, highlighting shifts in behaviour composition. For instance, a user who normally sends 5% of their emails externally but suddenly jumps to 40% would produce a high external email ratio, even if their total email count remains unchanged.
- Temporal features encode when activity occurs. The hour of the day is represented using cyclical encoding ($\sin$ and $\cos$ transformations) to preserve the continuity between late night and early morning hours. Binary indicators capture whether the activity occurs on a weekend or outside standard working hours. These are important as some malicious activities are performed outside normal working hours.
- Combination features capture co-occurrences that are individually normal but suspicious together, such as USB device usage after hours.

Rolling deviation features are computed as z-scores over recent windows for some features. All features are computed per user per time window from the aggregated CERT activity logs.

Table 4. Summary of features used to represent user behaviour in each time window.

Activity type	Description
log count	total activities, number of files, documents, archive, executable, size files, exchanged emails, websites, job website, cloud storage, devices and unique devices.
Ratios	Files, emails, external email, attachments, http, job sites, cloud storage and device.
Temporal Features	Hour sin, hour cos, is weekend, is after hour
Combination of features	USB after hours, file device interaction, email after hours.
Rolling features	log of total activities, archive and executable files, device ratio, file ratio

C. Additional Experimental Results

Table 5 provides more detailed insights into the behaviour of Offline DQN. For each fold, the Threats column reports the number of windows that contain malicious activity (note that a single malicious user typically has many windows of malicious activity, if left uninterrupted). The Missed column lists how many of these threats were missed (i.e., not blocked by the agent). Finally, the last two columns list the average and median delay, respectively, in the agent's blocking. Delay is measured as the number of time windows between a malicious user's first instance of malicious behaviour, and that user getting blocked, and is only tracked for malicious users that are in fact blocked at some point by the agent.

Table 5. Response timing of Offline DQN across 5 folds.

Folds	Threats	Missed	Avg delay	Med delay
Fold 1	558	218	14.9	0
Fold 2	372	225	7.9	2
Fold 3	605	98	2.6	0.5
Fold 4	442	75	0.2	0
Fold 5	739	224	11.7	1.5

D. Training setup

Table 6 reports the main training hyperparameters. PPO is trained using environment interaction, while offline DQN is trained on the generated transition dataset.

Table 6. Training hyperparameters used for PPO and offline DQN.

Category	**Hyperparameter**	**Value**
Common	Time window	3 hours
	Network architecture	MLP: [128,128]
	Learning rate	10^{-4}
Offline DQN	Batch size	512
	Number of epochs	500
	Target network update	1000
	Discount factor γ	0.99
PPO	Batch size	128
	Discount factor γ	0.995
	Total timesteps	5×10^5
	Clip range	0.2
	GAE λ	0.95
	Entropy coefficient	0.3