RESTRAIN: Reinforcement Learning-Based Secure Framework for Trigger-Action IoT Environment

Md Morshed Alam [1,†], Lokesh Chandra Das[2,*,†], Sandip Roy[1,3], Sachin Shetty[3], and Weichao Wang[4]

[1]School of Cybersecurity, Old Dominion University, Norfolk, USA
[2]School of Computing, Wichita State University, Wichita, USA
[3]Center for Secure & Intelligent Critical Systems, Old Dominion University, Norfolk, USA
[4]Department of Software and Information Systems, University of North Carolina at Charlotte, Charlotte, USA
{m2alam, sroy, sshetty}@odu.edu, lokesh.das@wichita.edu, wwang22@charlotte.edu

2025 International Wireless Communications and Mobile Computing (IWCMC) | 979-8-3315-0887-6/25/$31.00 ©2025 IEEE | DOI: 10.1109/IWCMC65282.2025.11059593

***Abstract*—Internet of Things (IoT) platforms with trigger-action capability allow event conditions to trigger actions in IoT devices autonomously by creating a chain of interactions. Adversaries exploit this chain of interactions to maliciously inject fake event conditions into IoT hubs, triggering unauthorized actions on target IoT devices to implement remote injection attacks. Existing defense mechanisms focus mainly on the verification of event transactions using physical event fingerprints to enforce security policies to block unsafe event transactions. These approaches are designed to provide offline defense against injection attacks. The state-of-the-art online defense mechanisms offer real-time defense, but extensive dependency on the inference of attack impacts on the IoT network limits the generalization capability of these approaches. In this paper, we propose a platform-independent multi-agent online defense system, namely RESTRAIN, to counter remote injection attacks at runtime. RESTRAIN allows the defense agent to profile attack actions at runtime and leverages reinforcement learning to optimize a defense policy that complies with the security requirements of the IoT network. The experimental results show that the defense agent effectively takes real-time defense actions against complex and dynamic remote injection attacks and maximizes the security gain with minimal computational overhead.**

***Index Terms*—Deep Recurrent Q-Network, Internet of Things, Multi-Agent System, Reinforcement Learning, Remote Injection Attack, Trigger-action Platform.**

I. Introduction

Trigger action capabilities in Internet of Things (IoT) platforms make it easier for users to set up customized rules and let IoT hubs enforce these rules to automate network tasks. For example, a homeowner could set up a rule that instructs an IoT hub to ask a smart sprinkler system to open the water valve when a thermostat measure is 130°F, considering there is a fire hazard. Users can create functional dependencies and leverage causal relationships among IoT devices to set up rules in a *rule engine* [1]. Smart hubs implement these rules in real-time and invoke actions in IoT devices based on the occurrence of prerequisite event triggers defined in the rules [2]. Note that a smart hub is the central entity in an IoT network through which IoT devices communicate with each other. Typically, IoT devices report their cyber states to the hubs notifying recent event transactions. The hubs verify the physical states of the reporting devices and invoke actions in corresponding IoT devices based on predefined rules [1].

* Corresponding author, † Equal Contribution

An event transaction contains the details of an *event condition* which acts as a *trigger* for the relevant actions in other devices. The execution of user-defined rules creates a *chain of interactions* incorporating a sequence of triggers and corresponding actions [3]. The hubs utilize the chain of interactions to automate network tasks and report device changes to users. However, an attacker can exploit the chain of interactions to maliciously inject fake event conditions into the hubs to carry out *remote injection attacks* [4]. The objective of the attacker is to instantiate invalid actions in IoT devices and compromise the functionality of a goal node (see Section II-A) [5]. This type of attack is also referred to as *event spoofing attack* since the attacker injects spoofed event conditions into the hub to implement the attack [6].

Existing research mainly focuses on offline anomaly detection systems (i.e., event verification systems) to mitigate IoT environment vulnerabilities [6], [7], [8], [9], [10]. These systems first verify the occurrence of event conditions based on *physical event fingerprints* [7] captured by sensors and utilize rule-based or machine learning (ML)-based approaches to ensure that the event transactions comply with predefined security policies, thereby blocking the unsafe transactions. These security systems do not offer real-time protection against active remote injection attacks. Additionally, many of them require source code modification of IoT applications that limits the generalization capability of defense solutions [9]. To address the limitations of offline defense mechanisms, some researchers propose single-agent based online security solutions to provide real-time defense against agile remote injection attacks [4], [11]. These approaches rely on the impact of malicious activities in the network to infer the attack sequences, thereby applying relevant defense actions. However, the impact of malicious activities may not accurately reflect how attackers intrude the IoT environment. Therefore, the defense policy may not be adequate to effectively counter dynamic and complex attack activities in real-time.

In this paper, we introduce RESTRAIN, a reinforcement learning (RL)-based online security framework that simulates a realistic attack-defense scenario and trains a smart defense agent to guard the IoT environment against dynamically changing remote injection attacks. In RESTRAIN, a defense agent and an attack agent are trained independently to opti-

mize a defense policy and an attack policy, respectively.

We make the following contributions in this paper:

- We propose an RL-based multi-agent defense system, called RESTRAIN, that allows a defense agent to model attack activities at runtime and take optimal defense actions to secure IoT networks against progressing remote injection attacks.
- We design novel reward functions for both attack and defense agents, enabling them to achieve maximized attack and security gains through the optimal selection of attack and defense actions.
- We implement RESTRAIN using TensorFlow [12] and train attack and defense agents using Deep Recurrent Q-Network (DRQN) [13].
- We conduct an extensive experiment to evaluate the performance of RESTRAIN. Simulation results exhibit that RESTRAIN effectively defends against agile remote injection attacks with minimal computational overhead.

The rest of the paper is organized as follows. In section II, we provide an overview of a remote injection attack and introduce a relevant threat model. In section III, we describe the proposed defense system in detail. The experimental details and the simulation results are presented in section IV. Finally, in section V, we conclude the paper and highlight some future research directions.

II. Attack Overview

To implement a remote injection attack exploiting the chain of interactions, an attacker must inject fake event conditions into an IoT network and invoke relevant actions in target IoT devices. For example, suppose the owner of a smart home has set a customized rule: "*If the smart thermostat detects a temperature* $\geq$ 120°*F, then unlock the smart lock on the front door and open the smart window*". In this scenario, the attacker could inject a fake event that reports the temperature as 120°F, thereby maliciously triggering the *unlock()* and *open()* commands for the smart lock and smart window. Fig. 1 represents the example scenario.

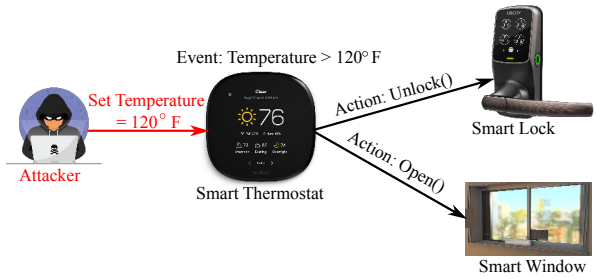


Fig. 1. Remote injection attack scenario.

We assume that the attacker injects a set of event conditions $C = \{c_i\}, 1 \leq i \leq n$, where n is the number of event conditions required to compromise a goal node. The attacker performs injection actions through a set of exploits $\xi = \{e_j\}, 1 \leq j \leq m$, where m is the number of exploits, and $m \leq n$. Each exploit e_j represents an interaction between an attacker and an IoT hub that reports an event condition c_i at time t. The attacker adopts an attack strategy to select a sequence of optimal exploits ξ to apply them to the IoT hub [4].

A. Attack Characterization

A remote injection attack can be modeled using a Directed Acyclic Graph (DAG), $\mathcal{G} = \{C, \xi\}$ demonstrating causal relationships between event conditions and corresponding actions [14]. Hence, event conditions act as graph *nodes* while actions (i.e., exploits) act as *edges*. The root of the graph represents the starting point of the attack injection and one of the leaf nodes represents the goal node. Note that the goal node is not fixed and can be different from time-to-time. When a defender generates a DAG for the purpose of analyzing the security of an IoT network, the defender must enforce the *monotonicity property* [14] in the DAG to avoid the *state explosion problem* [15]. This monotonicity property ensures that the prior nodes in the DAG have no impact on the successor nodes. The enforcement of this property keeps the graph reasonably small to perform additional security analysis. We assume that the defender generates attack graphs using available security tools (e.g., Topological Vulnerability Analysis (TVA) [16]) and conducts security analysis on constructed graphs.

B. Threat Model

The attacker performs active reconnaissance to discover attack vectors in the network and remotely injects fake event conditions using software called *ghost devices* [3]. These ghost devices are capable of simulating the behavior of real IoT devices. The attacker discovers device credentials from public forums and manufacturer's websites and utilizes these credentials to impersonate legitimate IoT devices. In addition, the attacker can capture network traffic using sniffers and extract real-time event traffic information by performing packet analysis using different IoT network utilities [17], [18]. The attacker can profile defense actions and have the ability to perform *opportunistic attacks* [7].

In an online defense system, the defender profiles ongoing attack actions and takes optimal defense actions to minimize the impact of attack actions in the network. We presume that the attacker cannot compromise the defense system and always takes minimal injection actions to evade detection by the defense system. Furthermore, we assume that the attacker is unable to compromise the sensors which are used to collect verifiable physical evidence and the IoT hub through which all IoT communications occur.

III. RESTRAIN: Reinforcement Learning-based Secure Framework for Trigger-Action IoT Environment

We present RESTRAIN as a multi-agent RL framework where a defense agent maximizes its security gain by optimizing a defense policy and an attack agent learns strategies to take the best injection actions. Fig. 2 shows the system architecture of RESTRAIN. The attack agent observes the IoT environment to get the current environment states as well as profiles the defense actions to estimate the latest defense action applied to the environment. The attack agent exploits these information to take current optimal action to the environment. Similarly, the defense agent observes the up-to-date security state of the environment and profiles the attack actions to determine the best defense policy. Unlike

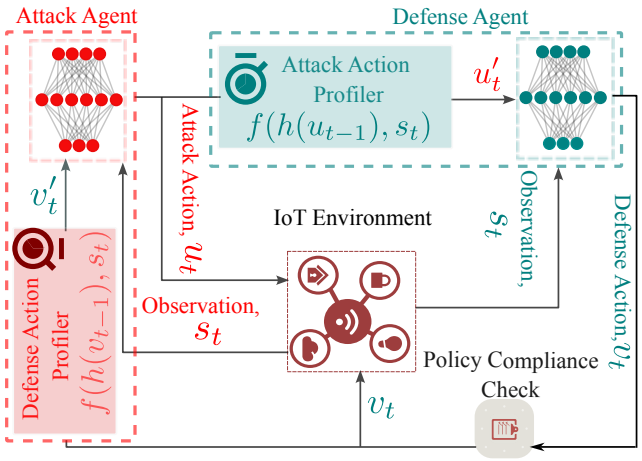


Fig. 2. An overview of RESTRAIN. The attack agent and the defense agent take u_t and v_t actions, respectively, at current time-step based on the observation s_t from the environment. The agents have the capability of profiling opponents' actions as their observations. The defense action goes through the *policy compliance check* before applying to the IoT environment.

the attack agent, the defense agent recommends the defense action to the IoT hub rather than directly applying to the IoT environment. Then, the hub runs a *policy compliance check* against predefined security policies for the defense action. Finally, the hub applies the defense action to the IoT environment if it passes the policy compliance check.

A. System Environment

We consider a smart home network as the system environment where the security states of the IoT devices, a set of attack actions, and a set of defense actions are encoded in a *finite state machine*. The state machine consists of $\mathcal{N}_\mathcal{S}$ unique states, $S = \{s_i\}, 1 \leq i \leq \mathcal{N}_\mathcal{S}$, $\mathcal{N}_\mathcal{U}$ unique attack actions, $U = \{u_j\}, 1 \leq j \leq \mathcal{N}_\mathcal{U}$, and $\mathcal{N}_\mathcal{V}$ unique defense actions, $V = \{v_k\}, 1 \leq k \leq \mathcal{N}_\mathcal{V}$. Hence, s_t represents the security state of the environment at time t after both the attack agent and the defense agent perform the actions u_{t-1} and v_{t-1}, respectively, at time $t-1$. We define s_t as the security state of the environment once the event condition c_t is reported to the hub. Note that the state s_t represents the input for the RL algorithms used for optimizing the policies for the defense agent and the attack agent. The environment transitions from s_{t-1} to s_t with probability of $T(s_{t-1}, u_{t-1}, s_t) = Pr(s_t|s_{t-1}, u_{t-1})$ due to the attack action u_{t-1}. If the transition occurs due to the defense action v_{t-1}, it occurs with probability $T(s_{t-1}, v_{t-1}, s_t) = Pr(s_t|s_{t-1}, v_{t-1})$. For a successful transition at time t, the environment returns r_{u_t} and r_{v_t} as rewards to the attack agent and the defense agent, respectively.

B. Attack Agent

The attack agent performs the following two types of actions (i.e., $\mathcal{N}_\mathcal{U} = 2$) to implement a remote injection attack: 1) u_r: performing *active reconnaissance* to discover attack vectors in the network; and 2) u_i: performing *event injection* to report fake event conditions to the hub to invoke unauthorized actions in target IoT devices. The attack agent infers the latest defense action, v'_t by conducting opponent modeling $v'_t = f\big(h(v_{t-1}), s_t\big)$ using an LSTM-based Recurrent Neural Network [19], where $h(v_{t-1}) = < v_1, v_2, ..., v_{t-1} >$. The objective is to optimize a policy $\pi^*_{u_t}$ to take an optimal sequence of actions (e.g., u_r, u_i) maximizing the overall discounted reward $\sum_{t=0}^{T} \gamma_u^t r_{u_t}$, where $\gamma_u \in [0, 1)$ determines how much the future reward should be discounted from the current time step t.

1) Reward Function: The objective of the attack reward function is to take optimal actions. We design the attack reward function in such a way that the attack agent takes minimal injection actions so that it does not expose itself to the defense agent by taking aggressive injections. In other words, the attack agent prefers more reconnaissance actions. At time t, the attack agent selects the action u_t following the reward function as given in equation (1):

$$r_{u_t} = \begin{cases} r_{u_r} - \lambda log(\kappa_u) & \text{if } u_t = u_r, \\ r_{u_i} + log(n_i) - \lambda n_i + r_g & \text{otherwise.} \end{cases} \quad (1)$$

where r_{u_t} represents the reward that the attack agent receives from the environment for taking the action u_t at time t. r_{u_r} represents the reward for taking the reconnaissance action, u_r. On the other hand, r_{u_i} represents the reward for injecting a fake event condition using the action u_i into the hub. r_g is the reward the agent receives if it is successful in injecting an event condition that helps invoke an undesired action in the goal node. At any given time step, the parameter κ_u denotes the number of consecutive reconnaissance actions, u_r performed by the attack agent. The parameters λ denotes *attack proximity factor* that signals a defense agent how close the attack agent is to the goal node, and n_i represents the number of injection actions performed thus far. The attack agent is incentivized to minimize both these parameters, as the larger values of these parameters make the attack actions increasingly more suspicious to the defense agent. We compute $\lambda \in [0, 1)$ as a ratio between the attack agent's current position and the goal node's position in the attack sequence.

C. Defense Agent

The objective of the defense agent is to obtain an optimal *defense policy* to recommend real-time defense actions to the smart hub maximizing the overall security gain. The defense agent performs two types of actions (i.e., $\mathcal{N}_\mathcal{V} = 2$): 1) v_a: *assessing* the security state of the environment, and 2) v_b: *blocking* unsafe trigger conditions. The defense agent takes v_a to assess how many event conditions $\{c_i\}$ the attack agent already injected into the hub. On the other hand, the defense agent performs action v_b to set a lower bound of the injection actions required to compromise a node by the attack agent. The defense agent incorporates opponent's modeling in its decision process and infers the latest attack action. In order to infer the latest attack action, the defense agent utilizes a history of attack actions as follows: $u'_t = f\big(h(u_{t-1}), s_t\big)$, where $h(u_{t-1}) = < u_1, u_2, ..., u_{t-1} >$. Like the attack agent, the defense agent optimizes defense policy $\pi^*_{v_t}$ yielding a sequence of v_a and v_b maximizing $\sum_{t=0}^{T} \gamma_v^t r_{v_t}$, where $\gamma_v \in [0, 1)$ is the discount factor.

1) Reward Function: The reward function is designed to motivate the defense agent to take optimal actions to provide RESTRAIN with active defense against the attack agent without hurting the availability of the network devices. To select an optimal action v_t at time t, the defense agent utilizes the reward function as given in equation (2):

$$r_{v_t} = \begin{cases} r_{v_a} - \omega_d log(\sigma \kappa_v) + \lambda\sigma & \text{if } v_t = v_a, \\ r_{v_b} - \sigma - log(n_b) & \text{otherwise.} \end{cases} \quad (2)$$

where r_{v_t} is the reward the defense agent receives for taking the action v_t. r_{v_a} and r_{v_b} represents the reward the defense agent receives from the environment for the security assessment actions (v_a) and the reward given for the block actions (v_b), respectively. The parameter κ_v denotes the number of consecutive v_a actions performed by the defense agent, and the parameter $\sigma \in (0, 1]$ denotes *injection threshold* that quantifies the tolerance level of the defense agent against injection actions, u_i. We define σ using the following formula: $\sigma = 1 - \lambda$. When σ is closer to 1, the defense agent becomes lenient against injection actions and prefers to take v_a actions more often. Once $\sigma \leq \lambda$, the defense agent takes block actions v_b aggressively to prevent the attack agent from reaching the goal node. The parameter n_b in equation (2) represents the number of block actions taken by the defense agent thus far, which is required to minimize to maximize the reward r_{v_t}. The parameter ω_d is a weighting factor which is empirically chosen to control the impact of injection threshold in the reward function.

IV. Simulation Results

A. Experiment

We develop an IoT environment using OpenAI Gym [20] to implement RESTRAIN. In particular, a trigger-action IoT platform is developed complying with the OpenAI Gym structure to simulate real-time attack-defense strategies using reinforcement learning. An RL algorithm called Deep Recurrent Q-Network (DRQN [13]) is implemented in Python leveraging the TensorFlow [12] framework. We train the RL models for both the defense agent and the attack agent using a machine equipped with Apple M3 Pro chip and 18GB RAM running on macOS Sonoma 14.4.

We utilize the PEEVES [7] dataset to extract event conditions to define the network structure of RESTRAIN. This dataset records IoT event transactions from 12 different event sources and physical evidence measured by 48 sensors. Event transactions and physical evidence share causal relationships, and therefore, we use the measured physical evidence to verify the occurrence of those event transactions. We translate the verifiable event conditions into network states and encode them into an IoT system environment. We encapsulate reward functions into the environment to provide feedback to agents for their actions.

We employ a *function approximator* for each agent and conduct hyperparameter tuning to figure out an optimal set of hyperparameters that maximizes the agent's objective. Especially, we perform a grid search [21], [22] over a range of hyperparameters including *learning rate*, *batch size*, *exploration rate* ϵ, and *network complexity*, etc. Table I enlists the optimal set of hyperparameters obtained through the grid search approach that we use to train our agents. As we model our IoT trigger action platform as a multi-agent learning system, we use the same set of parameters to facilitate the fair competition between the attack agent and the defense agent. The neural network consists of four hidden layers. The first hidden layer is a Dense layer with 64 neurons followed by two LSTM layers. Each LSTM layer has 32 neurons. Finally, the output of the final LSTM layer is passed through a Dense layer of 16 neurons before passing through the output layer. We train our *function approximators* using a stochastic gradient descent based optimization algorithm named Adam [23] optimizer and Mean Squared Error (MSE) as a loss function.

TABLE I
OPTIMAL HYPERPARAMETERS

discount factor, γ	0.99
batch size	32
learning rate, α	0.001
$(\epsilon, \epsilon_{decay}, \epsilon_{min})$	$(1.0, 9995, 0.005)$
target network update frequency, C	10 episodes
replay buffer size	$5k$
activation function for Dense layers	*ReLU*
activation function for LSTM layers	*Tanh*
weighting factor, ω_d	0.01

In our simulation, agents utilize the ϵ-*greedy* policy method defined in equation (3) to accelerate the learning process [24], [25].

$$\pi^{\epsilon}(a|s) = \begin{cases} 1 - \epsilon_t + \frac{\epsilon_t}{|A|} & \text{if } a = \text{argmax}_{a' \in A} Q_t(s, a') \\ \frac{\epsilon_t}{|A|} & \text{otherwise} \end{cases} \quad (3)$$

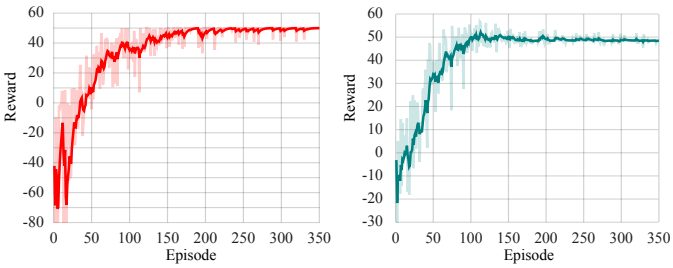
where $a \in \{u, v\}$ represents actions for both the attack agent and the defense agent. In ϵ-*greedy* policy, π^{ϵ} chooses a random action from the action space A with the probability of $\epsilon_t \in [0, 1]$; otherwise, it exploits a greedy action according to the Q_t.

B. Performance Evaluation

We run the simulation for 350 episodes, where each episode performs 50 gradient updates. We evaluate RESTRAIN in terms of total reward, number of injection actions taken by the attack agent, and the number of block actions employed by the defense agent to safeguard the IoT network. We also consider *attack injection threshold* and *attack proximity factor* for modeling the attack-defense dynamic of RESTRAIN. Finally, we assess the computational overhead of our proposed approach in terms of time incurred on the network environment.

1) Attack-Defense Reward: The reward of the attack agent represents the accumulated attack gain received by the attack agent from the environment for taking injection actions in all time steps in a certain episode. The environment provides a discrete reward to the attack agent computed using equation (1) for each attack action. The attack agent tries to discern an optimal attack policy to maximize the accumulated reward that indicates the agent's capability to perform optimal injection actions. Fig. 3(a) shows the learning trend of the attack agent.

At early stage of learning, the reward exhibits an unstable



(a) Attack agent reward. (b) Defense agent reward.

Fig. 3. The reward graphs for the attack agent and the defense agent.

pattern, which means that the agent does not learn an optimal policy. As training progresses, the agent learns to take optimal actions. More specifically, the reward graph shows more stable pattern after $\approx$ 150 episodes converging to the maximized cumulative reward.

Similarly, the defense agent achieves security gain as a form of reward according to equation (2) from the environment at every time step for each action it takes and tries to maximize the accumulated reward to determine the optimal defense policy in an episode. The defense agent also exhibits the unstable pattern of the accumulated reward like the attack agent at the initial stage of learning. However, the defense agent figures out the optimal defense policy more faster than the attack agent. Particularly, the defense agent achieves stable reward pattern at $\approx$ 100 episodes and shows a convergence trend afterwards as shown in Fig. 3(b).

2) Attack-Defense Dynamic: As we discussed in Section III, the attack agent and the defense agent make changes in the environment states by taking *injection* and *block* actions, respectively. To model the attack-defense dynamic, we take into account how an agent reacts to the opponent agent's action for which a state transition occurs. Fig. 4 depicts the attack-defense dynamic.

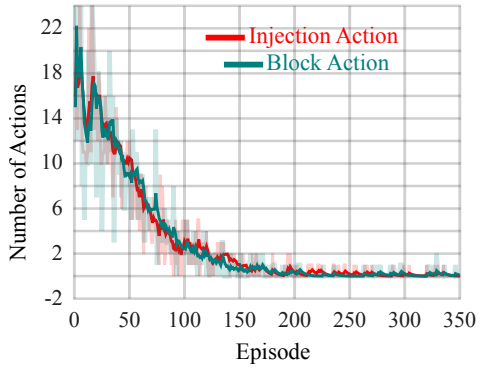


Fig. 4. Injection-Block actions taken by attack and defense agents.

When the attack agent exploits the network by taking more injection actions, the defense agent tries to secure the network by taking more block actions. However, as the more injection actions increase the chances of exposing the attack agent to the defense agent and the more block actions constraints the network availability, both agents try to take optimal injection and block actions as required in order to reach the goal node or protect the goal node. From Fig. 4, it can be observed that at the beginning of the training, when the attack agent exploits the network with more injection actions, the defense agent also takes more block actions to compete against the attack agent's aggressiveness. In contrast, the agents prefer taking *reconnaissance* and *assessing* actions more frequently to *injection* and *block* actions.

3) Defense Tolerance Level vs Attack Proximity Factor: We define the *attack proximity factor* as a metric to determine how close the attack agent is to the goal node whereas the *tolerance level* to represent the injection threshold that tells a defense agent when to become aggressive against attack activities. In RESTRAIN, the increase in attack proximity factor leads to the decrease of tolerance level and vice versa. When the attack agent starts performing injection actions aggressively, the defense agent starts reducing its tolerance level so that it can take more aggressive defense actions to obstruct the attack progression. Fig. 5 displays the result. It is clearly evident from Fig. 5 that the attack proximity factor increases at the initial stages of learning. This is because the defense agent allows the attack agent to take more injection actions so that it can learn the attack dynamic and gradually reduce its tolerance level. The defense agent starts taking aggressive defense actions (i.e., *block* actions) when the tolerance level becomes smaller or equal to the attack proximity factor. Once the defense agent blocks the crucial triggers from the IoT environment, it becomes infeasible for the attack agent to progress towards the goal node even though the tolerance level is increased a lot. This phenomenon is evident in Fig. 5. When the tolerance level increases after $\approx$ 110 episodes, the attack agent cannot increase the attack proximity factor beyond 0.5, demonstrating the fact that it becomes harder for the attack agent to inject malicious triggers into the environment to invoke invalid actions to reach the goal node.

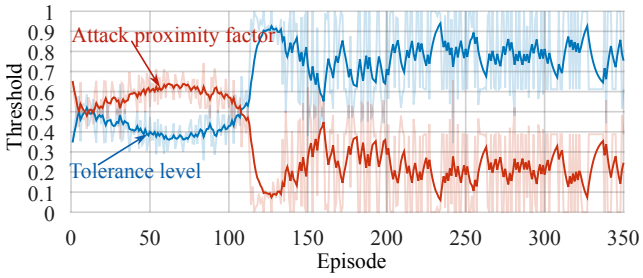


Fig. 5. The defense tolerance level vs the attack proximity factor.

4) Computational Overhead: The enforcement of security mechanisms always incurs computational burdens on systems and networks. RESTRAIN also adds some computational overhead to the IoT environment. To quantify this overhead, we compute the time (in seconds) the reinforcement learning algorithm takes in each episode to train the agents. Fig. 6 represents the computational overhead of RESTRAIN over episodes. It is evident that RESTRAIN learns to stabilize the computational overhead (e.g., $\leq$ 6.5s) after a certain number of episodes (e.g., after $\approx$ 75 episodes). This stability

in computational overhead signifies the applicability of the proposed system in real IoT settings.

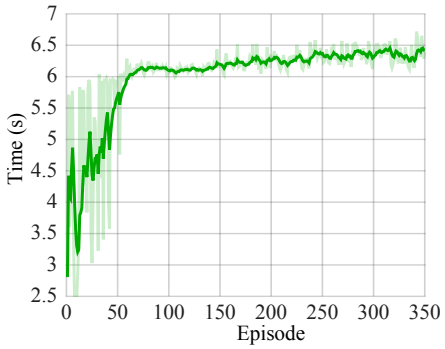


Fig. 6. Computational Overhead.

V. Conclusion

In this paper, we propose an RL-based multi-agent real-time defense system called RESTRAIN for trigger-action IoT environments, which facilitates a defense agent to optimize its defense policy by modeling attack strategies and an attack agent to optimally inject fake event conditions to invoke invalid actions in target IoT devices. We design novel reward functions and train both agents using Deep Recurrent Q-Network (DRQN) by creating a custom IoT environment using the OpenAI Gym framework. We run extensive simulations to evaluate our proposed framework. Simulation results exhibit that the defense agent can effectively safeguard IoT environments from complex remote injection attacks with minimal computational overhead.

In the future, we aim to evaluate RESTRAIN efficacy in more complex IoT environments using real-world testbed scenarios to verify its scalability and increase the number of attack agents to enhance RESTRAIN robustness against diverse and aggressive attack behaviors.

Acknowledgment

This work is supported in part by DoD Center of Excellence in AI and Machine Learning (CoE-AIML) under Contract Number W911NF-20-2-0277 with the U.S. Army Research Laboratory, National Science Foundation under Grant No. 2219742 and Grant No. 2131001. It is also supported in part by the Coastal Virginia Center for Cyber Innovation (CoVA CCI).

References

[1] J. Fan, Y. He, B. Tang, Q. Li, and R. Sandhu, "Ruledger: Ensuring execution integrity in trigger-action iot platforms," in *IEEE INFOCOM 2021 - IEEE Conference on Computer Communications*, 2021, pp. 1–10.

[2] Z. B. Celik, E. Fernandes, E. Pauley, G. Tan, and P. McDaniel, "Program analysis of commodity iot applications for security and privacy: Challenges and opportunities," *ACM Comput. Surv.*, vol. 52, no. 4, Aug. 2019. [Online]. Available: https://doi.org/10.1145/3333501

[3] M. M. Alam, M. S. I. Sajid, W. Wang, and J. Wei, "Iotmonitor: A hidden markov model-based security system to identify crucial attack nodes in trigger-action iot platforms," in *2022 IEEE Wireless Communications and Networking Conference (WCNC)*, 2022, pp. 1695–1700.

[4] M. M. Alam, I. Jahan, and W. Wang, "Iotwarden: A deep reinforcement learning based real-time defense system to mitigate trigger-action iot attacks," in *2024 IEEE Wireless Communications and Networking Conference (WCNC)*, 2024, pp. 1–6.

[5] M. M. Alam and W. Wang, "A comprehensive survey on data provenance: State-of-the-art approaches and their deployments for iot security enforcement," *Journal of Computer Security*, vol. 29, pp. 423–446, 06 2021.

[6] M. O. Ozmen, R. Song, H. Farrukh, and Z. B. Celik, "Evasion attacks and defenses on smart home physical event verification," in *30th Annual Network and Distributed System Security Symposium, NDSS*, 02 2023.

[7] S. Birnbach, S. Eberz, and I. Martinovic, "Peeves: Physical event verification in smart homes," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2019.

[8] C. Fu, Q. Zeng, and X. Du, "HAWatcher: Semantics-Aware anomaly detection for appified smart homes," in *30th USENIX Security Symposium (USENIX Security 21)*, Aug. 2021, pp. 4223–4240.

[9] Z. B. Celik, G. Tan, and P. Mcdaniel, "Iotguard: Dynamic enforcement of security and safety policy in commodity iot," *Proceedings 2019 Network and Distributed System Security Symposium*, 2019. [Online]. Available: https://api.semanticscholar.org/CorpusID:141048877

[10] D. T. Nguyen, C. Song, Z. Qian, and S. V. Krishnamurthy, "IotSan: Fortifying the Safety of IoT Systems Dang," *Proceedings of the 14th International Conference on emerging Networking EXperiments and Technologies*, pp. 387–400, 2018.

[11] M. M. Alam, A. B. M. Mohaimenur Rahman, and W. Wang, "Iothaven: An online defense system to mitigate remote injection attacks in trigger-action iot platforms," in *2024 IEEE 30th International Symposium on Local and Metropolitan Area Networks (LANMAN)*, 2024, pp. 15–20.

[12] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin *et al.*, "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," *arXiv preprint arXiv:1603.04467*, 2016.

[13] M. Hausknecht and P. Stone, "Deep recurrent q-learning for partially observable mdps," in *2015 aaai fall symposium series*, 2015.

[14] P. Ammann, D. Wijesekera, and S. Kaushik, "Scalable, graph-based network vulnerability analysis," in *Proceedings of the 9th ACM Conference on Computer and Communications Security*, ser. CCS '02. New York, NY, USA: Association for Computing Machinery, 2002, pp. 217–224. [Online]. Available: https://doi.org/10.1145/586110.586140

[15] O. Sheyner, J. Haines, S. Jha, R. Lippmann, and J. Wing, "Automated generation and analysis of attack graphs," in *Proceedings 2002 IEEE Symposium on Security and Privacy*, 2002, pp. 273–284.

[16] S. Jajodia, "Topological analysis of network attack vulnerability," ser. PST '06. New York, NY, USA: Association for Computing Machinery, 2006. [Online]. Available: https://doi.org/10.1145/1501434.1501437

[17] R. Trimananda, J. Varmarken, A. Markopoulou, and B. Demsky, "Packet-level signatures for smart home devices," in *Network and Distributed System Security Symposium*, 2020. [Online]. Available: https://api.semanticscholar.org/CorpusID:211266570

[18] W. Zhang, Y. Meng, Y. Liu, X. Zhang, Y. Zhang, and H. Zhu, "Homonit: Monitoring smart home apps from encrypted traffic," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '18. New York, NY, USA: Association for Computing Machinery, 2018, pp. 1074–1088. [Online]. Available: https://doi.org/10.1145/3243734.3243820

[19] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, nov 1997. [Online]. Available: https://doi.org/10.1162/neco.1997.9.8.1735

[20] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," 2016. [Online]. Available: https://arxiv.org/abs/1606.01540

[21] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, "Algorithms for hyper-parameter optimization," *Advances in neural information processing systems*, vol. 24, 2011.

[22] S. R. Yadavalli, L. C. Das, and M. Won, "Rlpg: Reinforcement learning approach for dynamic intra-platoon gap adaptation for highway on-ramp merging," in *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2023, pp. 5514–5521.

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

[24] M. Gimelfarb, S. Sanner, and C.-G. Lee, "ϵ-bmc: A bayesian ensemble approach to epsilon-greedy exploration in model-free reinforcement learning," *arXiv preprint arXiv:2007.00869*, 2020.

[25] L. Das and M. Won, "D-acc: Dynamic adaptive cruise control for highways with ramps based on deep q-learning," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2021, pp. 1572–1578.