1 Presentation of the different algorithms

1.1 Algorithm 2: Adaptive Learning-based Routing with Deception in MEC

Algorithm 1 Adaptive Learning-based Routing with Deception in MEC

Initialization:
Initialize θ_{CF}, θ_{CBF} for CF and CBF.
Load or initialize Q-table for routing decisions.
Configure parameters: learning rate α, discount factor γ, exploration rate ε.
Configure MEC-specific metrics (latency, bandwidth, resource state) and deception parameters (number of honeypots, types of fake vulnerabilities).
while MEC network is active do **do**
    Collect network and device metrics:
        Device profiles p_i, vulnerabilities v_i, exploit history e_i.
        MEC-specific metrics: latency, resource usage, and link quality.
    Construct a feature vector x_i for each deviced_i, including:
        Network characteristics (traffic, latency, and link state).
        Historical attack data, real and fake vulnerabilities.
    Calculate CF and CBF scores for potential routing paths:

        $$\text{Score}(p_j, d_i) = \theta_{CF} \cdot \text{CF-Score}(p_j, d_i) + \theta_{CBF} \cdot \text{CBF-Score}(p_j, d_i).$$

    Deceptive Vulnerability Deployment:
        Identify real vulnerabilities and create fake vulnerabilities based on attacker behavior.
        Deploy honeypots simulating these vulnerabilities to capture attacker interactions.
    Select optimal routes using Q-learning (see Algorithm 2).
    Deploy recommended routes and monitor traffic behavior:
        Supervise traffic for anomaly detection and validate routing effectiveness.
    Collect feedback on routing efficiency (latency, energy consumption, attack resilience).
    Validate and refine recommendations through simulations.
end while

Description:

Algorithm 1, entitled "Adaptive Learning-based Routing with Deception in MEC," is designed to secure the ARPMEC routing protocol in IoT networks integrating Mobile Edge Computing by combining advanced techniques such as Collaborative Filtering (CF), Content-Based Filtering (CBF), Reinforcement

Learning (Q-learning), and deception strategies using honeypots. The process begins with an initialization phase where the weights θ_{CF} and θ_{CBF} are defined (for example, $\theta_{CF} = 0.6$ and $\theta_{CBF} = 0.4$) to adjust the respective importance of CF and CBF in computing the routing path score. A Q-table is either initialized or loaded to store the Q-values used in Q-learning, while its parameters are set: α (learning rate, e.g., 0.1, determining how quickly the model adapts to new data), γ (discount factor, e.g., 0.9, which values future rewards), and ε (exploration rate, e.g., 0.2, balancing the exploration of new options and exploitation of the best-known ones). MEC-specific metrics, such as latency, bandwidth, and resource status, along with deception parameters (number and types of honeypots), are also established to adapt to the network context.

The data collection phase is crucial and applies to both IoT devices d_i and routing paths P_j. For each device d_i, its profile p_i (type, firmware version), vulnerabilities v_i (known flaws), and exploit history e_i (number and nature of past attacks) are collected and compiled into a feature vector x_i (for example, $x_i = [1, 2.1, 1, 3, 10, 50]$, where 1 indicates a sensor, 2.1 is the firmware version, 1 indicates a CVE-2023 vulnerability, 3 is the number of attacks, 10 ms is the average latency, and 50 kbps is the bandwidth). This feature vector x_i is a multidimensional numerical representation that captures the properties and needs of the device, serving as the basis for compatibility and security assessments. Similarly, for each path P_j, data such as the average latency, available bandwidth, security level (based on past evaluations), and link status are collected, forming a feature vector specific to the path (for example, $x_{P_j} = [5, 100, 0.9]$, for a latency of 5 ms, 100 kbps of bandwidth, and a security level of 0.9). These data are essential for assessing the suitability of each path.

The evaluation of paths calculates an overall score for each pair (P_j, d_i) using the formula

$$\text{Score}(P_j, d_i) = \theta_{CF} \cdot \text{CF-Score}(P_j, d_i) + \theta_{CBF} \cdot \text{CBF-Score}(P_j, d_i). \quad (1)$$

Collaborative Filtering leverages the collective experiences of devices similar to d_i. Its principle is to identify devices with feature vectors x_k similar to x_i (using cosine similarity, defined as

$$\text{cosine similarity}(x_i, x_k) = \frac{x_i \cdot x_k}{\|x_i\| \, \|x_k\|}, \quad (2)$$

and to assess the past performance of the path P_j for these devices). The CF-Score is calculated as

$$\text{CF-Score}(P_j, d_i) = \frac{\sum_{d_k \in \text{sim}(d_i)} \text{performance}(P_j, d_k)}{\text{number of similar devices}}, \quad (3)$$

where $\text{performance}(P_j, d_k)$ is a binary measure (1 for a secure path, 0 for an attacked one) or a continuous measure (for example, 0.9 for low latency). For

instance, if d_i has three similar devices with performances $[1, 0, 1]$ on P_j, then CF-Score $= \frac{1+0+1}{3} = 0.67$, reflecting an average performance. The role of CF is to anticipate risks by relying on collective patterns, thereby enhancing resilience against attacks such as DoS.

In contrast, Content-Based Filtering evaluates the direct compatibility between the features of the routing path P_j (via x_{P_j}) and the needs of d_i (derived from x_i). Its principle is to maximize the alignment between the properties of the path (latency, bandwidth, security) and the device's requirements (maximum acceptable latency, minimum bandwidth, required security level). The CBF-Score is defined as

$$\text{CBF-Score}(P_j, d_i) = \text{similarity}(\text{features of } P_j, \text{needs of } d_i), \tag{4}$$

where the similarity is measured using cosine similarity

$$\text{cosine similarity}(x_{P_j}, \text{needs of } d_i) = \frac{x_{P_j} \cdot \text{needs of } d_i}{\|x_{P_j}\| \, \|\text{needs of } d_i\|}. \tag{5}$$

For example, if $x_{P_j} = [5, 100, 0.9]$ (indicating a latency of 5 ms, 100 kbps of bandwidth, and a security level of 0.9) and the needs of d_i are $[10, 50, 0.8]$ (maximum latency 10 ms, minimum bandwidth 50 kbps, and minimum security level 0.8), the resulting CBF-Score might be 0.85, indicating a good match. The role of CBF is to ensure that the selected path meets the specific constraints of d_i, thereby optimizing performance.

The deception phase identifies the actual vulnerabilities of d_i and deploys honeypots that simulate false vulnerabilities to lure attackers, thus alleviating pressure on real devices. Path selection employs Q-learning to choose the routing path P_j that maximizes the score, while continuous monitoring adjusts strategies by analyzing feedback (latency, energy consumption, resilience) through simulations. For a given device d_i, the score $\text{Score}(P_j, d_i)$ is calculated for all available paths $P_j, P_k, P_l, \ldots$, and the optimal path is selected as

$$P^* = \arg\max_{P_j} \text{Score}(P_j, d_i). \tag{6}$$

For instance, if $\text{Score}(P_1, d_1) = 0.75$, $\text{Score}(P_2, d_1) = 0.82$, and $\text{Score}(P_3, d_1) = 0.68$, then P_2 would be chosen. This algorithm provides robust and adaptive defense; however, its complexity necessitates optimizations for the limited resources of IoT devices.

1.2 Algorithm 2: QLR-MEC: Q-learning for Routing with Deception in MEC

Algorithm 2 QLR-MEC: Q-learning for Routing with Deception in MEC

while receiving feedback from MEC environment do **do**
  Observe the current states_t, including:
    Security context (attacker tactics, honeypot interactions).
    Current network parameters (traffic, latency, link state).
  Select action a_t (weights and routing paths) using ε-greedy policy:
  if random() $< \varepsilon$ then **then**
    Choose a random set of weights and paths.
  else
    Choose weights and paths that maximize current Q-values:
      $a_t = \arg\max_a Q(s_t, a)$.
  end if
  Apply weights θ_{CF} and θ_{CBF}, observe reward r_t and new state s_{t+1}.
  Update Q-values:
    $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_t + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)]$.
  Add extra reward for honeypot interactions revealing attack tactics.
  Apply penalty if a critical route is exposed to intrusion.
  Dynamically adjust ε to reduce exploration as the model learns.
end while

Description:

Algorithm 2, named "QLR-MEC: Q-learning for Routing with Deception in MEC," focuses on optimizing routing decisions in a Mobile Edge Computing (MEC) environment using a Q-learning based reinforcement learning approach, while integrating deception elements to enhance security. This process operates in a continuous loop as long as feedback is received from the MEC network. At each iteration, the algorithm observes the current state, s_t, which encapsulates information such as the security context (attacker tactics, interactions with honeypots) and network parameters (traffic, latency, link status), and is essential for guiding routing decisions. The selection of action a_t, which includes choosing the weights θ_{CF} and θ_{CBF} as well as the routing path, is based on an ε-greedy policy: with a probability ε (e.g., initially 0.2), a random action is chosen to explore new options, while with probability $1 - \varepsilon$ the action that maximizes the Q-value,

$$a_t = \arg\max_a Q(s_t, a), \tag{7}$$

is selected to exploit current knowledge.

Once action a_t is applied, the algorithm observes the reward r_t (e.g., +1 for a secure path, +2 if a honeypot attracts an attacker, -1 if an attack succeeds)

and the new state s_{t+1}. The core Q-learning update uses the following formula:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_t + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \right], \tag{8}$$

where α (learning rate, e.g., 0.1) adjusts the speed of learning, and γ (discount factor, e.g., 0.9) weights future rewards. This equation enhances the prediction of action values based on past experiences and future expectations. To encourage interactions with honeypots, additional rewards are provided, while a penalty (e.g., -2) is applied if a critical route is exposed to an intrusion. Finally, ε is dynamically adjusted (e.g., reduced by 0.01 per iteration) to decrease exploration as the model learns, thereby favoring the exploitation of the best strategies.This algorithm offers an adaptive approach to optimize routing paths while incorporating deception as a defense mechanism. Convergence to optimal decisions depends on the quality of rewards and the definition of states, which can be challenging in highly dynamic environments such as MEC. However, its integration with Algorithm 1 leverages the CF and CBF scores to enhance decision-making, creating a powerful synergy to protect ARPMEC against DoS attacks. For example, if a state s_t indicates an ongoing attack and a honeypot is activated, the reward $r_t = +2$ will reinforce the Q-value of that action, encouraging the repetition of this strategy in similar contexts.