Meta-ATMoS+: A Meta-Reinforcement Learning Framework for Threat Mitigation in Software-Defined Networks

Hauton Tsang, Mohammad A. Salahuddin, Noura Limam and Raouf Boutaba
David R. Cheriton School of Computer Science, University of Waterloo, Waterloo, Canada
{hauton.tsang, mohammad.salahuddin, n2limam, rboutaba}@uwaterloo.ca

2023 IEEE 48th Conference on Local Computer Networks (LCN) | 979-8-3503-0073-4/23/$31.00 ©2023 IEEE | DOI: 10.1109/LCN58197.2023.10223403

***Abstract*—As cyber threats become increasingly common, automated threat mitigation solutions are more necessary than ever. Conventional threat mitigation frameworks are difficult to tune for different network environments, but frameworks utilizing deep reinforcement learning (RL) have been proven to be an effective approach that can adapt to different networks automatically. Existing RL-based frameworks have shown to be generalizable to different network sizes and threats, and robust to false positives. However, training RL agents for these frameworks can be challenging in a production environment as the training process is time-consuming and disruptive to the production network. Hence, a staging environment is required to effectively train them. In this paper, we propose Meta-ATMoS+, a meta-RL framework for threat mitigation in software-defined networks. We leverage Model-Agnostic Meta-Learning (MAML) to find an initialization for the RL agent that generalizes to a variety of different network configurations. We show that the RL agent with MAML-learned initialization can accomplish few-shot learning on a target network with comparable performance to training on a staging environment. Few-shot learning not only allows the model to be trainable directly in the production environment but also enables human-in-the-loop RL for the mitigation of threats that do not have an easily-definable reward function.**

***Index Terms*—Meta-Reinforcement learning, human-in-the-loop, threat mitigation, software-defined networks**

I. Introduction

Computer networks are getting more complex than ever, and cyber threats against organizations are ever-evolving. A malicious email attachment could easily compromise an employee's machine. A legacy system inadvertently made accessible to the internet may easily be compromised. A disgruntled employee may release malware in a network purposefully. Malicious actors can even surreptitiously install malware on a wide scale simply by compromising the update servers of popular software. The most concerning of these types of attacks are Advanced Persistent Threats (APTs), which are very difficult to detect until they are activated [1]. These attacks aim to establish a foothold within the network, avoid detection through lateral movement, and exfiltrate data undetected.

To guard against these threats, organizations typically adopt commercial or open-source security solutions to monitor them and respond manually. However, there have been multiple studies that have shown that these security solutions can generate a large number of false positives (*i.e.*, false alarms). FireEye conducted a survey that showed 37% of respondents detecting more than 10,000 alerts, of which more than 50% were false positives [2]. Too many false alerts can cause alert fatigue and could allow genuine attacks to remain undetected in the mix of false alerts. A survey conducted by the International Data Corporation (IDC) found that on average security analysts ignore 23% of alerts that are received in large companies, while smaller companies ignore an even higher percentage [3]. IDC also found that each alert can take around half an hour to investigate, significantly decreasing productivity.

More recently, organizations have adopted systems that streamline the investigation process and automate responses to incidents using Security Orchestration, Automation, and Response (SOAR) solutions [4]. In a traditional network, SOAR needs to be configured to interface with multiple backend services since the network is distributed. A software-defined network (SDN) can allow security responses to be much simpler, as a SOAR system only needs to communicate with the SDN controller via Northbound interfaces to control the entire network. By utilizing a Virtual Tenant Network plugin, multiple virtual networks (VNs) can be defined so that hosts on the network can be placed in them so that hosts can be isolated based on customizable network policies that can prevent the malware from spreading uninhibited.

However, automated solutions are only viable if false positive rates are low. Automated solutions that cannot learn to adapt to false positives in a network may cause additional problems if the responses they execute impact critical network components. Threat mitigation frameworks have been proposed that use reinforcement learning (RL) to learn how to ignore false positives when responding to alerts (*e.g.*, [5], [6]). However, although these frameworks are scalable to arbitrary network sizes, they still require training to be done on a separate staging network similar to the network it will be deployed in, as the training process requires disrupting the network to operate. Furthermore, the training process of these frameworks can take a significant amount of time to converge.

We propose Meta-ATMoS+, a meta-learning framework that runs on an SDN and automates the mitigation of threats by automatically placing hosts in the correct VNs. Meta-ATMoS+ can adapt to different networks using few-shot learning. Rather than training exclusively in the target network, the RL agent in the framework is trained on a set of networks beforehand

using Model-Agnostic Meta-Learning (MAML) to learn a good initialization [7]. We show that the Meta-ATMoS+ RL agent obtained from applying MAML provides some major advantages: (i) the RL agent can perform decently even without training specifically on the target network, (ii) the RL agent can adapt to a target network with a small number of iterations, and (iii) the RL agent is scalable to larger network sizes and more types of cyber attacks.

Unlike other threat mitigation algorithms that target particular threats, such as Denial of Service (DoS), Meta-ATMoS+ can be applied to generic threats as its reward function is not constrained to any particular threat. Also, due to only needing a small number of iterations to adapt, the Meta-ATMoS+ RL agent does not cause significant disruption. It can be trained without a staging network and can be integrated with a human-in-the-loop framework to provide feedback in a production environment.

The rest of the paper is structured as follows. Section II evaluates other works related to threat mitigation. Section III presents the structure of the Meta-ATMoS+ framework, while Section IV provides implementation details of the framework and performance results. Section V concludes the paper by summarizing results and providing future directions.

II. Related Works

A. Non-RL-based Threat Mitigation on DoS attacks

Threat mitigation focuses on responding to a threat once it has been discovered. Most literature on threat mitigation relates to the mitigation of DoS attacks against SDNs. Our paper focuses on RL-based approaches to threat mitigation, however, there have been works that use non-RL methods.

SDN-Guard [8], a non-RL approach, aims to mitigate DoS-type attacks that have been detected by an intrusion detection system (IDS). SDN-Guard samples network traffic from switches, mirrors them to an IDS, and collects aggregated flow statistics on detected malicious traffic and combines them with other network statistics. The network statistics and IDS alerts are fed into a flow management module, which applies flow rules to redirect malicious flows to less-saturated links. However, SDN-Guard does not provide feedback to the model on whether the alerts detected by the IDS were legitimate. Consequently, SDN-Guard cannot outright block malicious traffic because of potential false positives.

SoftThings [9] leverages anomaly detection and classifiers in order to mitigate DoS attacks. Flow and packet-level features from network observers are used as input to the anomaly detection model. The output of the anomaly detector is then fed into a Support Vector Machine classifier that determines if traffic is malicious or not. If the classifier predicts the traffic to be malicious, then flow rules are installed to immediately block the traffic and blacklist the originating IP address. However, SoftThings has the drawback of potentially blocking a lot of legitimate traffic as well, since the classifier's output could contain false positives.

B. RL-based Threat Mitigation

RL-based approaches allow the use of reward functions in order to incorporate feedback for the threat mitigation framework. Reward functions could be used to autonomously fine-tune a model and improve its performance. One type of network attack that has an easily-definable reward function is DoS. For example, DeepAir is a framework that uses a reward function that penalizes based on how much legitimate traffic was impacted by an attack [10]. The reward also increases based on the amount of malicious packets dropped, and decreases based on the number of flow rules needed to perform the mitigation. The RL agent is implemented as a double Deep Q-Network (DQN), and is able to block the IP address, rate limit the traffic, redirect the attack, reroute the traffic, or do nothing. However, it is not straightforward to extend a DoS-specific approach such as DeepAir for additional types of threats.

Another example of a DoS-specific approach applies RL to mitigate slow DoS attacks [11]. DoS attacks are detected using a neural network-based IDS system. The attacks are then mitigated using an intrusion prevention system (IPS) powered by a deep RL agent. The deep RL agent receives a reward based on the number of malicious flows between hosts detected by the IDS that were not blocked by the IPS. In contrast to the Meta-ATMoS+ framework, their work assumes that the IDS has minimal false positives as it has been trained on a dataset collected on the same network it will be running from. Collecting this dataset in a realistic environment may be difficult and it is unclear if the IDS could remain reliable even if deployed in different network architectures.

C. Threat Mitigation on Generic Threats

The major challenge of extending frameworks beyond mitigating DoS attacks is that there is no straightforward reward function for automatically learning to mitigate generic threats. One method of generalizing to more general network attacks is to define a reward function that rewards the blocking of malicious network packets, while punishing the blocking of legitimate packets, such as in [12]. This work proposes RL agents that use a DQN and Proximal Policy Optimization (PPO) architecture. To train their RL agents, they set up an SDN using an OpenDaylight (ODL) controller, Docker containers, and Open vSwitch switches. The reward for the RL agent is the number of legitimate network packets subtracted by the number of malicious packets transferred through the network. However, for this kind of reward function to be evaluated, the malicious packets must be reliably identified by an IDS, otherwise, the RL agent may not be trained properly. The authors also fail to evaluate the scalability of their agent.

A different approach to address generic threat mitigation is to model the problem as a two-player game with an attacker and defender [13]. The defender attempts to defend against an attacker hacking various hosts. One host is assigned critical services, and the game is over if the host with critical services is hacked. The defender attempts to either evade the attacker by moving critical services or block a potential attacker it

has identified. With this setup, the authors are able to apply DeepMind's MuZero RL agent [14] to mitigate threats. The reward function is based on the total impact caused by the attacker, the cost of the countermeasures used by the defender, and the number of moves before the end of the game. The major limitation of this approach is scalability. MuZero itself is a complex RL agent and requires a lot of resources to fine-tune. Given the significant amount of resources needed to run a 12-host network, this RL agent may not be feasible to run in most production environments.

Finally, ATMoS [5] and its extended variant ATMoS+ [6] attempt to solve the problem of mitigating generic threats. ATMoS runs on an SDN with VNs that can be applied to particular hosts in a network. ATMoS uses a DQN-based RL agent, and its input consists of alerts from an IDS to decide if a host needs to be moved to a different VN. ATMoS trains the RL agent using network simulations, and its reward function is based on whether the RL agent correctly mitigated a host using prior knowledge of which hosts were malicious.

ATMoS+ extends the ATMoS framework using set functions in its neural network model, allowing its inputs to be invariant to host re-ordering, and allowing it to scale to accommodate different network sizes. Some limitations of ATMoS and ATMoS+ are that the training of the agents can take a considerably long time, and both need to be deployed in a staging network hosting simulations with labeled malicious hosts for initial training. Furthermore, as these RL agents rely on prior knowledge of malicious hosts to calculate their reward, they cannot be directly updated in a production environment to accommodate for changes in the network over time.

III. META-ATMOS+

Meta-ATMoS+ is an extension of ATMoS+ that aims to address several limitations in existing works, including: (i) extensive training time that makes it infeasible to directly train in production, (ii) lack of generalizability in threat mitigation without relying on an IDS or prior knowledge, and (iii) not being able to scale to larger networks.

Meta-learning is able to reduce the number of training iterations (*i.e.*, total number of steps) an RL agent needs to solve a particular problem by learning on a set of similar problems. This addresses two of the three limitations that have been outlined above. For the first limitation, a reduction in the number of training iterations means that disruption due to training the meta-RL agent is minimized. Furthermore, in practice, an RL agent that has been optimized with meta-learning performs reasonably well in a production environment without any training at all.

To address the second limitation, a reduction in the amount of training needed to adapt the Meta-ATMoS+ RL agent can allow a human to feasibly check the decisions the RL agent makes and provide feedback prior to execution. This feedback could then serve as a reward to the RL agent in lieu of a reward function that uses the output of an IDS or prior knowledge to determine which packets or hosts are malicious. This generalizes the RL agent to mitigate any cyber attack that can feasibly be identified by a human analyst.

To address the final limitation, the Meta-ATMoS+ RL agent utilizes permutation equivariant and permutation invariant neural networks to decrease the number of parameters in the RL agent, as well as accommodate network size changes. The neural networks used in the Meta-ATMoS+ RL agent's model are small in size so adaptation could be performed more quickly.

A. Problem Formulation

In order to apply RL to the problem of threat mitigation, we first model the problem of threat mitigation as a Markov decision process (MDP). Assuming an environment consisting of an SDN, a network observer, and predefined VNs, we can define the set of states of the MDP as the current VN, alert information from the IDS, and the history of VN locations for all hosts. The actions of the MDP are whether to change the VN of any host or to do nothing. During the evaluation, each host has a corresponding VN assigned to it (*i.e.*, malicious hosts are assigned to a high-security VN, and benign hosts are assigned to a low-security VN). The reward function r is defined based on whether the model correctly placed the host being acted upon in the current step.

The reward function is asymmetric as it was experimentally found that punishing the RL agent more when taking an incorrect action decreases training time compared to a symmetric reward function where correct and incorrect actions were weighted equally. This reward function serves as a proxy for human analysts who would be responsible for investigating if a host is malicious in a real-world deployment of the framework.

B. Meta-ATMoS+ Architecture

Fig. 1 shows the architecture of the Meta-ATMoS+ framework. Hosts within the SDN will be located in VNs connected to an Openflow-enabled switch. The switch will forward network traffic to the network observer, which is running an IDS and a network statistics collector. The network observer then sends the results of the IDS and flow statistics to the RL agent, which extracts features from the results and feeds it to the RL agent model, and selects an action to take (*i.e.*, change the VN of a host, or do nothing). The RL agent can be in either automated or manual mode. Once the action is chosen, if the RL agent is in automated mode, the action is immediately forwarded to the SDN controller, where the controller then performs the action. If the RL agent is in manual mode, the action is sent to the security analyst, which will investigate whether the action is correct. If it is, the action is approved, a positive reward will be sent to the model, and the action will be forwarded to the SDN controller which executes the action. If the action is not approved, a negative reward will be sent to the model. The RL agent can be switched between manual and automated modes by the user.

C. Meta-ATMoS+ RL Agent Architecture

The Meta-ATMoS+ RL agent is an off-policy agent with a neural network model m. The model architecture is based on

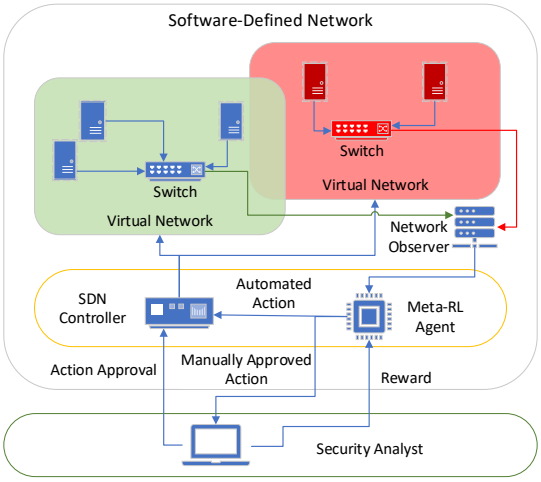

Fig. 1: Architecture of Meta-ATMoS+

ATMoS+ [6], and uses layers based on permutation invariant functions and permutation equivariant functions. The weights on the permutation invariant and equivariant layers are shared between all hosts, so the number of trainable weights in the model is invariant under changes in network size. Thus, not only is the model robust to host re-ordering, but the model can also accommodate changes in the size of the network.

A permutation invariant function is a function $f : \mathbb{R}^n \rightarrow \mathbb{R}^p$, such that any input can be swapped with any other input, and the output will remain constant. A permutation invariant function is applied to determine whether any actions need to be taken on the hosts. The permutation invariant function implemented in the Meta-ATMoS+ RL agent's model is based on permutation invariant Deep sets [15]:

$$f(\boldsymbol{x}) = \rho(\max(\phi(x_1), \phi(x_2), ...\phi(x_n))),$$

where ρ and ϕ are approximated using dense neural networks. The function is permutation invariant as the aggregation function $max(x_1, x_2, ...)$ is permutation invariant, therefore, it erases information on the ordering of the input once it is applied. The max function was chosen as its range is constant even when the number of hosts is increased.

A permutation equivariant function is a function $g : \mathbb{R}^n \rightarrow \mathbb{R}^n$ such that if the i-th input of g may be exchanged with the j-th input, the corresponding output at i and at j will also be swapped. In other words, a permutation applied on the inputs of g will result in an identical permutation applied to the outputs of g. Therefore, any inputs of g that have been swapped will cause the corresponding outputs of g to also be swapped. The permutation equivariant function that is used in the RL agent is based on a formulation by Sannai *et al.* [16]. Each input of the function x_i is applied to the following function:

$$g_i(x_1, ..., x_n) = \rho'(\max(\{\phi'(x_j)\}_{j \neq i}), x_i),$$

where $\rho' : \mathbb{R}^p \rightarrow \mathbb{R}$ and $\phi' : \mathbb{R}^n \rightarrow \mathbb{R}^p$ are arbitrary functions. The permutation equivariant function $g(\boldsymbol{x})$ is defined as the concatenation of all outputs of $g_i(x_i)$:

$$g(\boldsymbol{x}) = (g_1(\boldsymbol{x}), g_2(\boldsymbol{x}), ...g_n(\boldsymbol{x})).$$

The RL model of Meta-ATMoS+ $m : \mathbb{R}^{n \times k} \rightarrow \mathbb{R}^{n+1 \times k}$ can be defined in terms of f and g_i as follows:

$$m(\boldsymbol{x}) = (f(\boldsymbol{x}), g_1(\boldsymbol{x}), g_2(\boldsymbol{x}), ...g_n(\boldsymbol{x})).$$

The architecture of the Meta-ATMoS+ RL agent model is shown in Fig. 2.

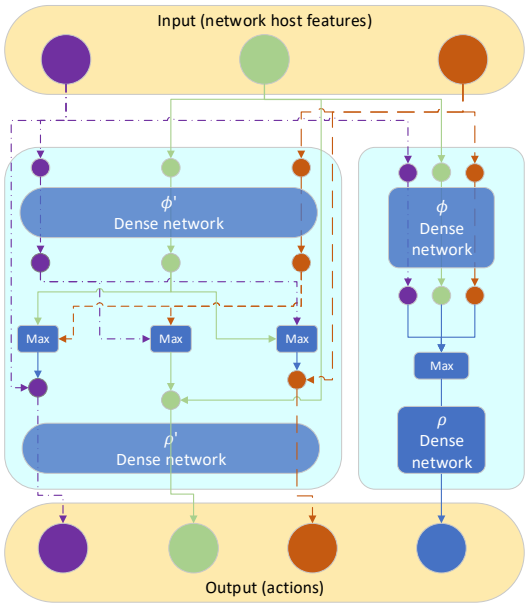

Fig. 2: Model architecture of the Meta-ATMoS+ RL agent

The $x_i \in \mathbb{R}^k$ in the model input $\boldsymbol{x} = \{x_1, ..., x_n\}$ represent a vector of k features corresponding to the i-th host in the network. Each x_i is a vector with the following features: (i) a one-hot encoded list of alerts for the host obtained from an IDS, (ii) the current VN the host is located in, (iii) the VNs the host was located in the past, and (iv) summary flow statistics (bytes transferred, bytes received, packets transferred, packets received, and connection duration averaged over 5 minutes). The functions ϕ and ϕ' are approximated using two dense neural network layers, the first layer has 36 neurons, and the second layer has 8 neurons. Both layers use ReLU activation functions. The functions ρ and ρ' are approximated using a single dense neural network layer with 8 neurons and ReLU activation functions.

The RL agent also incorporates ϵ-greedy exploration in meta-learning. During exploration, the X-means algorithm is used to cluster the input from hosts, and a random cluster is chosen before selecting a host to act upon. The X-means algorithm is a clustering algorithm that can also infer the number of clusters based on a set of initial points. This reduces

the probability of the RL agent exploring the same types of hosts consecutively during exploration, reducing training time. For example, in a network with a large number of hosts belonging to users and a small number of hosts hosting infrastructure such as web servers, the X-means algorithm groups the users into a cluster and the web servers into a different cluster. Therefore, during exploration, both types of hosts would be explored more equally than if the host was selected at random. For Meta-ATMoS, the X-means initial points were selected using two centers selected using K-means++. Both the X-means and K-means++ algorithms were implemented using the PyClustering library [17].

D. Meta-ATMoS+ RL Agent Training

For meta-learning, the Meta-ATMoS+ RL agent uses the MAML algorithm. By defining a distribution of networks $\mathcal{T}$, MAML can be used to find a good set of initial parameters (*i.e.*, initialization) for the model to solve any task in $\mathcal{T}$ by optimizing the agent on a sample of networks $t \subset \mathcal{T}$. The initialization obtained from optimizing the RL agent on t would then be applicable to other tasks in $\mathcal{T}$. To apply the MAML algorithm in our case, we define $\mathcal{T}$ to be a distribution of networks, and t is a set of training networks that are part of this distribution.

The MAML algorithm for the Meta-ATMoS+ RL agent is outlined in Algorithm 1. The algorithm updates m using policy gradient-based methods. This procedure improves the model parameters of m for all networks in $\mathcal{T}$, as it performs gradient descent on the average of the gradients for all networks in t, which serves as an estimate of the gradient for $\mathcal{T}$.

Algorithm 1 MAML training for Meta-ATMoS+

Require: : Set of training networks $t \subset \mathcal{T}$
Require: : Model m
1: Randomly initialize parameters for m
2: **for all** training networks t **do**
3: Sample a trajectory $\mathcal{D}$ using parameters of m
4: Evaluate the gradient of the loss function calculated from trajectory $\mathcal{D}$
5: Make a copy of m denoted as m'
6: Update m' with new parameters using gradient descent
7: Sample new trajectories $\mathcal{D}'$ using m'
8: **end for**
9: Evaluate the gradient of the loss function calculated using parameters for m and trajectories $\mathcal{D}'$ for all t
10: Update m with new parameters using gradient descent

This process is depicted visually in Fig. 3. The model initially starts at a random initialization. Then, by incorporating gradients obtained from a set of sample networks (in this case, $t_1, ..., t_4$), a meta-gradient can be obtained that allows the initialization to move closer to an optimum initialization for all the sample networks. The optimum initialization is an initialization that provides the shortest adaptation process needed to obtain the optimum parameters for any t_i. The optimum initialization could then be used as the starting point for adapting a target network. As long as the optimum parameters of the target network are close to the optimum initialization and optimums of the sample networks, the adaptation process would be much shorter than starting from random initialization. One way of ensuring that the optimum initialization is close to the target network optimum is to define a distribution of networks that includes the target network, then randomly select a representative sample of networks from that distribution.

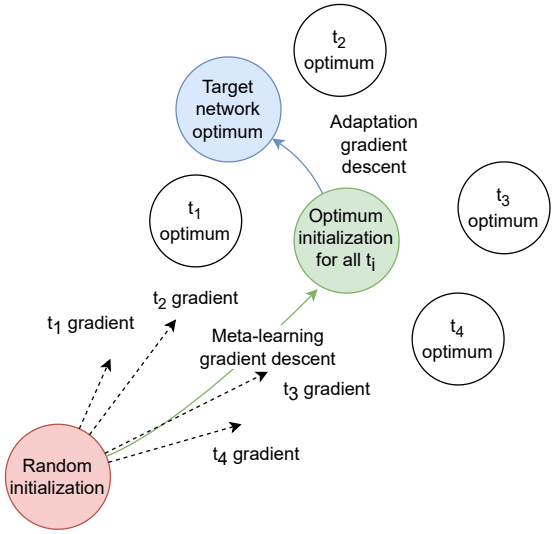

Fig. 3: MAML meta-learning process

In practice, defining a distribution of networks is unnecessary as long as the training networks have similar traffic patterns to the target network. The most reliable method of measuring this similarity is to evaluate the performance of a meta-learned model using the training networks on the target network. Alternatively, heuristics could be used to approximate similarity prior to evaluation. For example, if the selected training networks and target network were all from educational institutions, then they should have similar traffic patterns, and meta-learning performance might be higher. However, if the training networks were educational institution networks, and the target network is a financial institution's network, meta-learning performance will likely be lower, as the traffic patterns would likely not be similar.

E. Meta-ATMoS+ RL Agent Adaptation

Once the model has been trained, the model can then be adapted by sampling new trajectories directly in the target network. Each step in the trajectory will then be sent to the security analyst for evaluation, and a reward for each step will be assigned. The resulting RL agent can then be adapted to any network similar to networks in $\mathcal{T}$ with few-shot learning.

IV. Evaluation

The goal of the evaluation section is to answer the following questions: (i) How quickly can the Meta-ATMoS+ RL agent adapt to a network compared to adapting an already-trained

RL agent? (ii) Can the meta-learning framework scale to large networks? (iii) Does meta-learning improve adaptation performance over attempting to adapt an already-trained model? (iv) Can the meta-learning framework scale and accommodate more types of attacks?

A. Environment Setup

The Meta-ATMoS+ RL agent was tested on an SDN deployed using Containernet, an SDN network emulator that deploys hosts using Docker images. ODL was used as the SDN controller for the network, and the Virtual Network Tenant plugin was installed on ODL to provide VN functionality. The switches used in the SDN are handled by Open vSwitch. The RL agent is implemented in Python with Keras and Tensorflow. The model parameters of RL agent are in Table I. The size of the list of alerts and list of previous VN locations were chosen as larger sizes did not meaningfully alter the performance of the model, but significantly increased resource usage and model prediction latency. Both meta-learning and adaptation training use the Adam optimizer. The model hyper-parameter values such as exploration probability (ϵ), discount factor (γ), and learning rate (α), were chosen using grid search. The grid search aimed to find parameters that resulted in fewer iterations for adaptation. The search ranges were based on the hyper-parameters from ATMoS+.

The Docker images used for the experiment consist of labeled benign and malicious images that were implemented to simulate a range of different users, services, and attacks that can be found in real-world network environments. For simplicity, each host is assumed to either be a self-contained service or a specific type of user. Docker images containing a service will start the service using a set of initialization shell scripts when the Docker image is run. Docker images representing a user contain multiple actions that simulate how a user may behave in a realistic scenario. When the Docker container is run, the container will randomly select and execute one of the actions. Once the action is complete, a different action will be executed. This repeats until the container is stopped. The following are the different types of benign Docker images used to evaluate our framework:

- **Web Server:** Hosts an insecure PHP web server running on Apache with a MySQL database.
- **System Administrator:** Simulates a user that uses Secure Shell (SSH) to access both web servers and file servers to perform maintenance using a shell script to run various SSH commands.
- **Researcher:** Simulates a person searching for information on Google and occasionally connecting to the web servers in the network to upload documents.
- **File Server:** Hosts a vulnerable Samba server with SMBv1 enabled.

The following are the attack types of malicious Docker images used to evaluate our framework:

- **SQL Injection (SQLi):** Injecting SQL on form fields in the login page and URL parameters of websites.
- **Directory Traversal:** Attempts to read sensitive system files, such as /etc/passwd, by injecting URL parameters in websites.
- **SMB Attack:** Attempts to exploit vulnerabilities in SMBv1.
- **Port Scan:** Attempts to scan common ports on all hosts in the network.
- **TCP SYN Flood:** Attempts to DoS the web server by flooding the HTTP port with TCP SYN packets.
- **UDP Flood:** Attempts to DoS the web server by flooding it with UDP traffic.
- **APT Data Exfiltration:** Attempts to log in to the web server and exfiltrate sensitive data after a delay.
- **APT DoS:** Attempts to perform DoS after a delay.

The traffic in the network is redirected to an IDS running Snort, which has the following categories of alerts enabled:

- **Port scanning:** Triggered by a large number of connections to different ports on the same host.
- **Reconnaissance:** Triggered by pings to a large number of hosts.
- **SMBv1 connection:** Detects if SMBv1 is being used.
- **DoS:** Triggers if an excessive number of TCP or UDP packets are being received.
- **Bandwidth:** Triggered by high sustained bandwidth usage.
- **Injection attempt:** Triggered by specific character sequences in the URL or POST request, such as *;–* or emph../../.

Some of these alerts (*i.e.*, Reconnaissance and Bandwidth) can be triggered by legitimate operations of the System Administrator or Researcher images to simulate an unreliable IDS.

For our experiment, we define the distribution of networks to be the set of networks whose hosts are a combination of any of the hosts in Table I. Therefore, the networks used for evaluation contain a random combination of malicious and benign hosts. To keep the networks realistic, the number of malicious hosts in each network is capped at 40% of the network size or 10 malicious hosts in total, whichever is smaller. All of the experiments in the evaluation were conducted on Amazon Web Services in m5.8xlarge instances, which consist of 32 vCPU cores and 128GB RAM.

To compare against state-of-the-art, we also performed experiments with ATMoS+ and DeepAir-based RL agents. These agents were adapted in order to run them in our network. ATMoS+ represents the current state-of-the-art generic threat mitigation framework, and the DeepAir-based RL agent adapts a state-of-the-art DoS mitigation framework to work with generic threats.

The ATMoS+ RL agent uses the same model architecture, but the reward function was modified to be the same as the Meta-ATMoS+ RL agent. This modification is necessary because the ATMoS+ reward function is evaluated by checking every host in the network for malicious behavior. This is infeasible for a human analyst to accomplish in the real world. The Meta-ATMoS+ reward function can be evaluated by only

checking one host, *i.e.*, the host that the RL agent is currently acting upon.

The DeepAir-based RL agent also uses the same model architecture, but the action space of the model was adjusted to indicate which VN the model wants to place a host at, and the reward function was also changed to use the Meta-ATMoS+ RL agent's reward function. Additionally, DeepAir's input features were initially intended to detect DoS attacks only, so they were modified to include features from Meta-ATMoS+ as well.

TABLE I: Meta-ATMoS+ RL Parameters

State	Flow statistics (bytes transferred, bytes received, packets transferred, packets received, connection duration) avg over 5 mins, list of last 10 alerts from IDS, list of last 5 VN locations, current VN location
Action	Change VN of a host, or do nothing
Reward	$r = \begin{cases} 1 & \text{if host placed in correct VN} \\ 0 & \text{if no action was taken} \\ -2 & \text{if host placed in incorrect VN} \end{cases}$
α (meta-learning)	0.005
α (adaptation)	0.2
γ	0.9
ϵ	0.1
# of sample networks	10
Training trajectories	5,000 per sample network, each 10 iterations long

B. Adaptation Performance versus State-of-the-art

To evaluate the adaptation performance of the Meta-ATMoS+ RL agent, we trained it initially using 10 different 10-host networks using MAML. To compare the adaptation performance of the Meta-ATMoS+ RL agent with ATMoS+ and DeepAir, both the ATMoS+ and DeepAir RL agents were trained using the same set of 10-host networks in sequence. For each network, they were trained for 5,000 trajectories with 10 iterations per trajectory.

To test adaptation performance, we generated a random 100-host target network. During adaptation, instead of 10 iterations per trajectory, each trajectory only contained a single iteration. At the end of every 10 iterations, a 10-iteration trajectory for the target network was generated from the model, and the reward for this trajectory was recorded. This was repeated for 30 different target networks. As the reward varies depending on the number of misplaced hosts in the network, the reward was scaled so that the maximum reward for each network was 10. The results are shown in Fig. 4, with the solid and dashed lines indicating the average scaled reward, and the shaded region showing a standard deviation from the mean.

The figure demonstrates that the Meta-ATMoS+ RL agent already performs respectably with an average reward of 7.7 even without adaptation, and most malicious hosts were successfully mitigated. As the number of iterations increases, the scaled reward steadily improves. At 390 iterations and above, the Meta-ATMoS+ RL agent was able to achieve the maximum reward on every target network. In contrast, the ATMoS+ and DeepAir RL agents have an average reward of 3.5 and -4.6 initially, which is significantly lower. Furthermore, the number of iterations to achieve an acceptable reward is also much higher, with ATMoS+ requiring around 1,000 iterations to achieve the maximum reward, and DeepAir requiring almost 1,600 iterations.

The decreased initial performance of ATMoS+ and DeepAir may be due to catastrophic forgetting, *i.e.*, general knowledge about previous networks may not be retained when training on a new one. This leads to an RL agent that is more specialized to the final training network than the Meta-ATMoS+ RL agent, resulting in lower performance when running on the target network. The larger number of iterations needed for DeepAir may be due to the fact that DeepAir's RL agent model has a much larger number of parameters than the Meta-ATMoS+ RL agent, necessitating additional training iterations to fully adapt it.

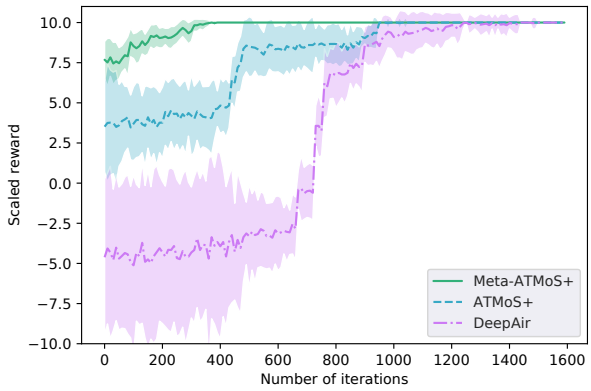

Fig. 4: Scaled reward versus number of training iterations for RL agents of different frameworks

C. Scalability to Network Size

In order to test the scalability of the Meta-ATMoS+ RL agent, we evaluated the number of iterations that would be needed to adapt a model to a new network for various network sizes. To do this, we utilized the same procedure to train the RL agent from Section IV-B but varied the training network size from 10 to 100. Then, we tested how long it would take to adapt the RL agent on target networks with sizes ranging from 10 to 100 hosts. For each network size, we recorded the number of iterations the RL agent needed to achieve the maximum reward. We ran this experiment 30 times with different target networks and averaged the number of iterations for each network size. Results are shown in Fig. 5. Each line denotes a model trained on different-sized training networks.

From the figure, it is clear that the Meta-ATMoS+ RL agent performs similarly regardless of training network size. It is likely that as long as the training networks have samples from all the different attack types, the initial size of the training network doesn't matter. The figure also shows that the Meta-ATMoS+ RL agent can optimize for smaller networks more

quickly than larger ones, and the number of iterations needed scales almost linearly with the number of hosts. This may be due to a larger network being more complicated, necessitating additional training iterations to optimize. However, the number of iterations is still low enough that a human can feasibly double-check all the decisions made by the Meta-ATMoS+ RL agent and provide feedback to it in a production environment. In comparison, DeepAir and ATMoS+ require 70,000 and 5,000 iterations, respectively, to maximize the reward obtained by their RL agents from a random initialization [10], [6].

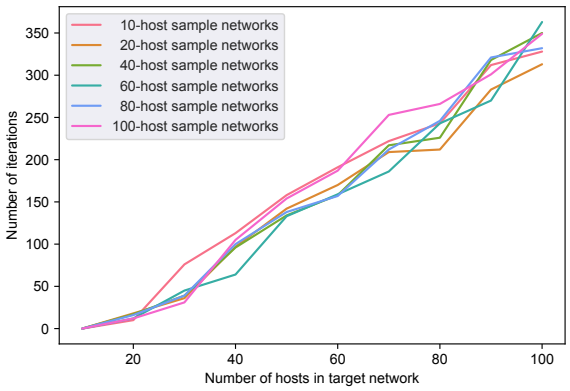

Fig. 5: Average number of iterations needed to train the Meta-ATMoS+ RL agent for various network sizes

D. Scalability versus State-of-the-Art

We also compared the Meta-ATMoS+ RL agent's scalability performance with the ATMoS+ and DeepAir RL agents. To do this, we optimized the Meta-ATMoS+ RL agent with ten different 10-host networks using MAML. For ATMoS+ and DeepAir, we trained the models using the same set of 10-host networks in sequence but using our reward function for 5,000 trajectories with 10 iterations per trajectory. Then for all three RL agents, we found the number of training iterations they took before being able to achieve the maximum reward in the target network. This was then repeated 50 times. The result is shown in Fig. 6. The solid and dashed lines denote the mean number of iterations, and the shaded region denotes the values within a standard deviation of the mean.

The figure shows that the Meta-ATMoS+ RL agent clearly outperforms the ATMoS+ and DeepAir RL agents across all network sizes from 10 to 100, though ATMoS+ and the Meta-ATMoS+ RL agent perform comparably in smaller networks. This may be due to the fact that the small networks are more similar to each other than larger networks, as fewer combinations of hosts can be chosen. Therefore, the optimal parameters for the models of the networks are closer together. As the trend seems to indicate that the number of iterations will scale linearly depending on the size of the network, we believe that the Meta-ATMoS+ RL agent will require significantly fewer iterations to adapt to a large network than either ATMoS+ or DeepAir RL agents.

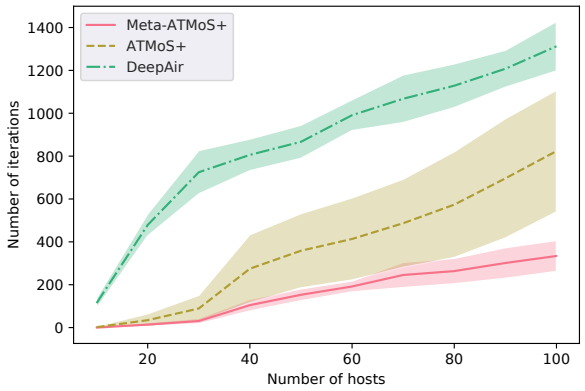

Fig. 6: Number of iterations needed to train RL agents of different frameworks

E. Scalability to Attack Types

We evaluated how the Meta-ATMoS+ RL agent scales with the number of attack types. This was done by restricting the attack types of malicious hosts during both the meta-learning and adaptation phases. First, a subset of attack types was selected based on the number of attack types that were being evaluated. We then optimized the Meta-ATMoS+ RL agent using ten 10-host networks where the malicious hosts were chosen solely from the selected attack types. Then, we adapted the model to a 100-host network where the malicious hosts were also chosen from the initially selected attack types. We recorded the number of iterations needed for the model to achieve the maximum reward in the 100-host network. This procedure was repeated for both the DeepAir and ATMoS+ models. The results are shown in Fig. 7.

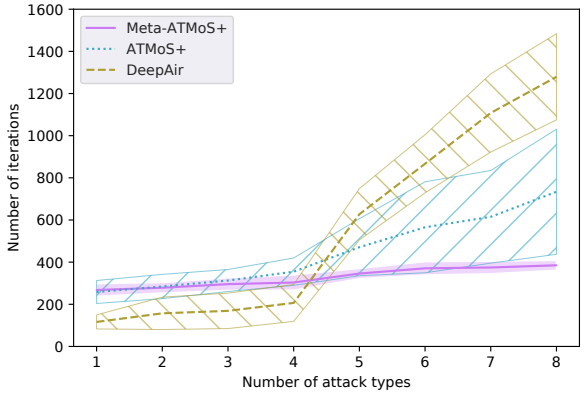

Fig. 7: Number of iterations needed to train RL agents as attack types increase

The figure shows a positive correlation between the number of attack types and the number of iterations needed to optimize all three RL agents, which is expected due to additional knowledge that they need to learn to mitigate additional types of malicious hosts.

For a few attack types, the DeepAir RL agent performs slightly better than ATMoS+ or Meta-ATMoS+. This is expected as the architecture of the DeepAir RL agent's model is a deep neural network consisting of dense layers, which can encode information more efficiently than the permutation equivariant and permutation invariant layers in the Meta-ATMoS+ RL agent. However, its performance drastically decreases after more than four attack types are introduced. This may be due to the number of malicious hosts in the training networks being capped at 40%, so in a 10-host network only four attack types can be present, so if there are five or more types, the training networks will not be able to contain all of them. The DeepAir RL agent does not seem to be able to efficiently capture attack-type information if the attacks are not all present together in the same network, limiting its scalability.

ATMoS+ also performs slightly worse than the Meta-ATMoS+ RL agent. The Meta-ATMoS+ RL agent can better retain information about previously seen attack types than ATMoS+ due to the meta-learning process. This allows it to efficiently optimize for previously-seen attacks.

V. Conclusion

In this paper, we proposed a meta-learning-based RL framework for automating threat mitigation in SDN. By applying MAML, Meta-ATMoS+ can adapt to new networks with few-shot learning. We show that the model performs better than state-of-the-art and can scale to accommodate larger networks. Therefore, Meta-ATMoS+ could readily be integrated into any organization's network, where the adaptation could be supervised by a human-in-the-loop.

In real-world scenarios, the human-in-the-loop process can leverage resources in a company's Security Operations Center (SOC), which typically has cybersecurity personnel investigating potential malicious activity in the company's network. The Meta-ATMoS+ RL agent's adaptation could easily be integrated into an existing SOC by creating security events that SOC personnel can investigate using their existing workflow, minimizing friction in the adoption of the Meta-ATMoS+ framework. Although smaller companies may have smaller cybersecurity teams, their network would likely be smaller, so the number of steps needed to adapt the Meta-ATMoS+ RL agent would also decrease.

An assumption in the evaluation of the model is that the human analyst in the loop is perfect. As the reward for the RL agent is based on feedback from human analysts, the quality of the feedback that the human analyst provides is very important. As the model is adapted using few-shot learning, even a single mistake could significantly affect the accuracy of the model, requiring additional training to correct.

A future direction that could be explored is expanding the RL agent to accommodate federated reinforcement learning (FRL) [18]. FRL can allow the RL agent to learn from many different types of enterprise networks directly, which can improve the generalizability of the RL agent. The RL agent currently relies on its initial set of networks to find a good initialization for similar networks, so the selection of these initial networks is important. However, if FRL can be integrated into the model, then the generalizability of the RL agent can be augmented further during the adaptation process, decreasing the model's reliance on the initial set of networks it was trained on.

References

[1] A. Alshamrani, S. Myneni, A. Chowdhary, and D. Huang, "A survey on advanced persistent threats: Techniques, solutions, challenges, and research opportunities," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 2, pp. 1851–1877, 2019.

[2] C.-P. S. V. Organization, "Automating threat detection desired for security analysts battling fear of missing incidents and security operations center inefficiency," https://cps-vo.org/node/74490, 2021, Accessed: 2023-03-30.

[3] C. Robinson, "In cybersecurity every alert matters," https://www.criticalstart.com/wp-content/uploads/2021/11/US48277521_TLWP.pdf, 2021, Accessed: 2023-04-14.

[4] P. Alto, "What is soar?" https://www.paloaltonetworks.com/cyberpedia/what-is-soar, Accessed: 2023-04-14.

[5] I. Akbari, E. Tahoun, M. A. Salahuddin, N. Limam, and R. Boutaba, "Atmos: Autonomous threat mitigation in sdn using reinforcement learning," in *IEEE/IFIP Network Operations and Management Symposium*, 2020, pp. 1–9.

[6] H. Tsang, I. Akbari, M. A. Salahuddin, N. Limam, and R. Boutaba, "Atmos+: Generalizable threat mitigation in sdn using permutation equivariant and invariant deep reinforcement learning," *IEEE Communications Magazine*, vol. 59, no. 12, pp. 105–111, 2021.

[7] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," in *International conference on machine learning*. PMLR, 2017, pp. 1126–1135.

[8] L. Dridi and M. F. Zhani, "A holistic approach to mitigating dos attacks in sdn networks," *International Journal of Network Management*, vol. 28, no. 1, p. e1996, 2018.

[9] S. S. Bhunia and M. Gurusamy, "Dynamic attack detection and mitigation in iot using sdn," in *International Telecommunication Networks and Applications Conference*, 2017, pp. 1–6.

[10] T. V. Phan and T. Bauschert, "Deepair: Deep reinforcement learning for adaptive intrusion response in software-defined networks," *IEEE Transactions on Network and Service Management*, vol. 19, no. 3, pp. 2207–2218, 2022.

[11] N. M. Yungaicela-Naula, C. Vargas-Rosales, J. A. Perez-Diaz, and D. F. Carrera, "A flexible sdn-based framework for slow-rate ddos attack mitigation by using deep reinforcement learning," *Journal of Network and Computer Applications*, p. 103444, 2022.

[12] M. Zolotukhin, S. Kumar, and T. Hämäläinen, "Reinforcement learning for attack mitigation in sdn-enabled networks," in *IEEE conference on network softwarization*, 2020, pp. 282–286.

[13] J. Gabirondo-López, J. Egana, J. Miguel-Alonso, and R. O. Urrutia, "Towards autonomous defense of sdn networks using muzero based intelligent agents," *IEEE Access*, vol. 9, pp. 107 184–107 199, 2021.

[14] J. Schrittwieser, I. Antonoglou, T. Hubert, K. Simonyan, L. Sifre, S. Schmitt, A. Guez, E. Lockhart, D. Hassabis, T. Graepel *et al.*, "Mastering atari, go, chess and shogi by planning with a learned model," *Nature*, vol. 588, no. 7839, pp. 604–609, 2020.

[15] M. Zaheer, S. Kottur, S. Ravanbakhsh, B. Poczos, R. R. Salakhutdinov, and A. J. Smola, "Deep sets," *Advances in neural information processing systems*, vol. 30, 2017.

[16] A. Sannai, Y. Takai, and M. Cordonnier, "Universal approximations of permutation invariant/equivariant functions by deep neural networks," *arXiv preprint arXiv:1903.01939*, 2019.

[17] A. Novikov, "Pyclustering: Data mining library," *Journal of Open Source Software*, vol. 4, no. 36, p. 1230, apr 2019. [Online]. Available: https://doi.org/10.21105/joss.01230

[18] J. Qi, Q. Zhou, L. Lei, and K. Zheng, "Federated reinforcement learning: Techniques, applications, and open challenges," *arXiv preprint arXiv:2108.11887*, 2021.