ATMoS: Autonomous Threat Mitigation in SDN using Reinforcement Learning

Iman Akbari, Ezzeldin Tahoun, Mohammad A. Salahuddin, Noura Limam, and Raouf Boutaba
University of Waterloo, Waterloo, Ontario, Canada
{iakbaria, etahoun, mohammad.salahuddin, n2limam, rboutaba}@uwaterloo.ca

***Abstract*—Machine Learning has revolutionized many fields of computer science. Reinforcement Learning (RL), in particular, stands out as a solution to sequential decision making problems. With the growing complexity of computer networks in the face of new emerging technologies, such as the Internet of Things and the growing complexity of threat vectors, there is a dire need for autonomous network systems. RL is a viable solution for achieving this autonomy. Software-defined Networking (SDN) provides a global network view and programmability of network behaviour, which can be employed for security management. Previous works in RL-based threat mitigation have mostly focused on very specific problems, mostly non-sequential, with ad-hoc solutions. In this paper, we propose ATMoS, a general framework designed to facilitate the rapid design of RL applications for network security management using SDN. We evaluate our framework for implementing RL applications for threat mitigation, by showcasing the use of ATMoS with a Neural Fitted Q-learning agent to mitigate an Advanced Persistent Threat. We present the RL model's convergence results showing the feasibility of our solution for active threat mitigation.**

I. Introduction

Threat actors are constantly evolving their arsenals, weaponizing new technologies to design more sophisticated attacks for exploiting their targets. Mitigating these attacks has become extremely challenging in today's increasingly complex networks with many vulnerable layers, ever-expanding attack surfaces and advanced threat vectors. Software-defined Networking (SDN) is a new paradigm to networking, rapidly adopted in recent years by large enterprises, which decouples the control and data planes in a network. Its logically centralized control benefits from the network global view, and opens new opportunities for enhanced defences against network intrusions.

The autonomy of network security management is motivated by the high cost and inaccuracies of manual human inspection of network data. The detection of attacks is of paramount importance, but some attack vectors, such as Advanced Persistent Threats (APTs), are built for stealthiness. APTs change their behaviour depending on the environment to deceive human analysts, as they take small steps over extended periods of time, and act benign under scrutiny. This grants the threat actors a significant amount of time to go through the attack cycle, propagate and achieve their objectives on the victim's network.

Machine Learning (ML) has recently had a large impact on many areas of computer science, including automation of network management and cyber security [1], [2]. On the other hand, Reinforcement Learning (RL), as a well studied area of ML, has grown in importance in the past few years, after showing promising results in achieving human-level control in video games [3]. Therefore, it is natural to investigate the feasibility of RL in autonomous defence of SDN networks. RL deals with problems that require discrete-time sequential decision-making based on the notion of *learning* a good behaviour. As illustrated in Fig. 1, at each iteration, an RL agent follows a trial-and-error strategy by interacting with its environment, and modifying its behaviour based on the *reward* it receives from the environment.

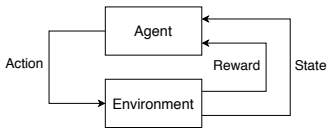

Fig. 1: A standard reinforcement learning model

In a standard RL model, an agent observes the environment via perception. It also takes actions inside the environment, through an actuator, which may change the *state* of the environment. At each time step, the agent receives an observational input reflecting the state s of the environment. In turn, it can take an action a that transfers the state of the network into a new state s'. This is communicated to the agent, along with a scalar *reinforcement signal* r, indicating the value and desirability of the taken action [4].

Recent efforts in applying RL to network threat mitigation [5]–[8] have been limited in their scope and applicability. In most cases, the defined states and action-sets are constrained to a certain kind of attack or a particular setup in a network, making it difficult to generalize their approaches to arbitrary attacks, mitigation tasks, and network topologies. The results of these ad-hoc solutions can not be compared easily. Furthermore, they are mostly evaluated on simple threat vectors that do not take full advantage of RL's *sequential* decision-making. Thus, the use of RL, as opposed to typical supervised learning, is not really justified.

In this paper, we present our framework, ATMoS, which provides a unified general scheme for designing complex RL agents for the network threat mitigation task. We provide concrete definitions of how threat mitigation should be formulated as an RL problem, which is one of the primary challenges in RL for network security. We showcase the use of our framework in a setup where ATMoS is successfully used to mitigate an APT attack. We envision that this will facilitate future efforts

978-1-7281-4973-8/20/$31.00

in the application of RL to threat mitigation, in a way that the results of the consequent solutions can be easily compared.

ATMoS is a step towards autonomous networks [1] that have proven to be necessary due to the enormous scale and stealthiness of threat vectors, and degrees of uncertainty in future networks. Our main contributions are:

- The ATMoS framework that enables the rapid design of RL applications for network security, along with a concrete formulation of threat mitigation as an RL problem
- An open source implementation[1] of ATMoS on Mininet using state-of-the-art industry tools, such as OpenDaylight, Open vSwitch, Docker, Snort, *etc*.
- A set of experiments with ATMoS using Neural Fitted Q-learning (NFQ) to mitigate an APT attack against a number of benign hosts
- An insightful discussion of the related literature, highlighting their shortcoming in the design of RL problems with respect to reward functions and action-sets

The rest of the paper is structured as follows. We provide a review of the related literature in Section II. The architecture and design of ATMoS are delineated in Section III. In Section IV, we present our implementation and experimental results. Finally, we instigate opportunities for future research in Section V and conclude with a brief summary of our work in Section VI.

II. Related Work

A large body of work in the literature has been dedicated to ML-assisted intrusion detection [2]. However, acting upon the detected potential threats with the help of ML remains quite obscure. Mitigating network attacks, in general, can be a very tricky task, especially when the attacker deliberately tries to keep a low profile and evade detection (*e.g.*, APT). Intrusion detection systems and traffic classifiers are highly prone to false positives. Thus, simply dropping the traffic that is suspected to be malicious can have severe consequences for the availability and functionality of the network.

In the past, there has been limited work in leveraging RL techniques for threat mitigation. Vishnupriya *et al.* [7] have theorized using SDN and RL's Direct Learning strategy to set a dynamic threshold for packets per port based on the network state to mitigate Denial of Service (DoS) attacks. However, in their evaluations the authors implemented a constant threshold and did not report the effectiveness or the feasibility of the proposed method.

Liu *et al.* [6] propose the use of Deep Deterministic Policy Gradient (DDPG) [9] to mitigate DoS attacks in SDN. The authors define the state as a vector of different statistical features of the traffic. The action-set is defined as the maximum bandwidth allowed for each host on the network, and the reward as a function of three variables: victim server CPU usage, benign traffic throughput and malicious traffic throughput. In real world scenarios, CPU usage will not be a reliable metric as it is subject to drastic changes due to a wide range of factors (*e.g.*, seasonality of workload bursts). This undermines the feasibility of the proposed method in many environments. Furthermore, this approach can hardly be generalized to attack types other than DoS, since the damage of most attacks is not proportionate to the volume of malicious traffic reaching the victim host.

It is important to highlight that evaluating such strategies against a simple attack where the correct policy can be inferred from the current state alone and sequential decision-making is not required, is fundamentally flawed. From an ML perspective, in such scenarios a supervised learning problem is being incorrectly rephrased as an RL problem. Most attacks assumed in the literature we surveyed are problems that do not require sequential logic to mitigate. Another shortcoming in the related literature is that reward functions are usually either over-simplified or based on highly unstable features. In Section III we define a more complex attack that needs a sequential approach to be mitigated and for our RL reward we select reliable indicators of compromise.

Han *et al.* [5] use the RL algorithms Double Deep Q-network (DDQN) [10] and Asynchronous Advantage Actor Critic (A3C) [11] to migrate critical resources and rewire the network to mitigate a DoS attack against a local server. They define a state to hold two insights about each node: whether the node is hacked and whether it is powered on. The implementation of the data retrieval is not mentioned and might not be trivial in practical deployments. The actions are defined to be isolating a node, patching a node, reconnecting a node to its links, or migrating a critical resource to a destination on the network, or no action. Reward is defined based on whether the critical resources are compromised, number of nodes accessible from critical resources, number of compromised nodes, whether the action taken is valid in current environment, and the migration cost. The authors focus on experimenting with attacking the RL using adversarial tactics, but fail to showcase their experiments with RL successfully protecting the servers and converging.

Malialis and Kudenko [12] ran multiple RL agents on a number of routers, and the agents were trained to rate-limit the traffic sent to a node that is under attack in the simulated environment. It is important to outline that each agent can not see the whole space, and finding an optimal solution is much harder in such a setup. The proposed model makes the continuous action-space discrete and might lead to combinatorial explosion. In contrast, we leverage the global view that SDN provides and hence, facilitate reaching an optimal policy compared to a distributed multi-agent model.

DoS mitigation has been thoroughly explored in the past and the area has been systematically surveyed over the years [13]. Researchers have used the powerful monitoring that SDN brings to deliver insightful ideas in DoS mitigation [14]–[18]. Many researchers experimented with various ML techniques combined with SDN to mitigate attacks and build a NIDS. Shin *et al.* [19] propose FRESCO, a scripting language that enables flow constraints and implements actuators to work with legacy security systems. Their framework is used to build an application to entrap malicious bot scanners as well as

[1]https://github.com/ATMoS-Waterloo

other applications. Few authors have used SDN and Deep Learning for anomaly detection. Mehdi *et al.* [20] used rate limiting, TRW-CB, NETAD, and maximum entropy detectors to predict anomalies in a Small Office/Home Office network at the network edge, while authors in [21] and [22] used Self Organizing Maps to detect Distributed DoS (DDoS) and U2R with DPI, respectively. More deep learning techniques later emerged in the literature but were mostly unrealistic in terms of scalability or had bottleneck problems [23]–[25].

III. System Design

In this section, we begin by explaining the high-level functional architecture of ATMoS. We then elaborate the design with a concrete example of a system that can be implemented based on this architecture. We also explore the formulation of the corresponding problem as a classic RL control problem and address numerous obstacles in this regard.

A. Problem

We hypothesize a scenario where a network is infiltrated by one or more malicious hosts (MH). The network also has legitimate benign hosts (BH) that need to run undisturbed. Furthermore, it is important to note that a MH is not running an overly-simplistic model of a DoS flooder, but more realistically, it is running an APT vector. APT is a stealthy attack that uses multiple attack vectors, tools, and tactics to avoid detection, with the primary goal of retaining access to the unauthorized network and system for an extended period of time [26]. In our scenario, we expect the APT to evade detection, and deliver attacks only when a vulnerable attack surface is in range.

We need to protect the convenience and functionality of the BHs, while identifying suspicious activity by the MH based on subtle observations taken over multiple steps. We want our action-set to be proportional to host maliciousness, in order to apply preventive actions in a strategic and systematic manner, while preserving the convenience to benign users. We start by gradually adding security constraints and tools against malicious actions, until our suspicions are confirmed and the threat actor is quarantined. This problem is sequential as the threat actor changes its behaviour based on the network state and our policies. RL is suitable to solve this problem, as it interacts with the environment in real-time and measures online metrics.

B. ATMoS Overview

Although ML-assisted network threat and anomaly detection is a well-explored area of research, the effort in ML-assisted threat mitigation remains limited (*cf.*, Section II). False positives are very common in intrusion detection systems and an alert by itself does not justify immediate ramifications for the host triggering it. Rather, active interaction with the host, testing against different environments and using different levels of analytic network functions are required to confirm its maliciousness.

As depicted in Fig. 2, our proposed system model is comprised of three components: SDN Infrastructure, Host Behaviour Profiling, and Autonomous Management where the RL agent resides. This design directly reflects the typical RL model depicted in Fig. 1. The RL agent, observes the network state provided by a module called *Network Observer*, which is installed inside the SDN, and aggregates information and insights about network state.

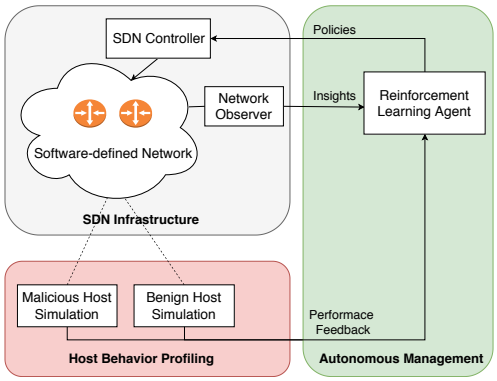

Fig. 2: The high-level system model — The RL agent in the Autonomous Management component operates by observing (i) a feed from the Network Observer installed in the SDN infrastructure, and (ii) the performance feedback from the simulated hosts inside the Host behaviour Profiling component

For the training, a set of simulations are deployed to the network. The malicious simulations mimic the behaviour of attackers, posing the same kind of threats we want to mitigate. The benign simulations imitate the low-profile user behaviour allowing the model to establish a semblance of what normal network usage should look like. These simulations report quantitative metrics about their performance to inform the RL agent that the current network policies affect their quality of experience. The main idea is that the mitigation task can be boiled down to maximizing the quality of experience for the benign hosts, while minimizing the success rate of the attacks by malicious hosts.

More specifically, the three components are as follows:

(I) **SDN Infrastructure**—For an RL agent to deploy policies in real-time, we assume the existence of an *SDN controller* that takes commands from the agent through a northbound API. The nature of these commands (*i.e.*, policies) are further discussed in Sections III-E1 and III-G. SDN acts as an *enabler* to our proposed approach for threat mitigation. A key element in our SDN infrastructure is the *Network Observer* that monitors the network traffic and provides insight into the network status. This can include the output of a traditional IDS, flow-meters, or any other network function that can provide useful real-time information relevant to network security. In our proof-of-concept, we leverage IDS/IPS, as they provide an *overview* of the network status and the potential threats.

(II) **Host behaviour Profiling**—Two sets of hosts programmed to behave as benign hosts (BHs) and malicious hosts (MHs) are deployed at training time, to provide a feedback loop for the RL agent. The MHs conduct the

same kinds of attacks that we hope to prevent in our network and BHs help the system to characterize the sorts of network traffic it should allow and not interfere with. The idea is that by looking at how network configurations at each point in time affect the performance of MHs and BHs, we can construct a *reward signal* for the RL model. This is the equivalent of establishing the ground truth. For instance, a BH can simply be an agent browsing the web mimicking human behaviour and an MH can be a host conducting known attacks against internal or external targets using the Metasploit framework [27]. The profiling can be updated frequently or even automated to mirror new benign behaviour, attack signatures, zero-day tactics, tools, and know-hows based on a threat database. The BHs constantly report a quantitative metric to reflect their quality of experience and the MHs report on how successful their attacks have been. Thus, the reward function can be a normalized mean of these values.

(III) **Autonomous Management**—The agent *observes* the network through the network observer feeds (*e.g.*, IDS alerts, traffic stats, *etc.*), and enforces policies on the network via the SDN controller. The deployed configuration will affect the performance of MHs and BHs, which would in turn be reflected in their reported performance metrics. The agent then receives this feedback from the simulations and is able to modify its strategy in a way that would maximize its expected cumulative reward through time. Thus, the agent can correct network policies through trial-and-error during training time.

C. Establishing Ground Truth (Reward) via Simulations

The design of the reward function is one of the biggest challenges in RL-based threat mitigation, one that has been rather over-simplified or avoided in most previous works in this area (*cf.*, Section II). For example, Sampaio *et al.* [8] use RL for DDoS mitigation with the following reward function: $R = 1$, if there is a congested network link, and $R = 0$, otherwise. With such an extreme simplification, the reward function will not only be unable to capture the dimensions and insights of the network condition, it can hardly generalize to other use-cases and different kind of attacks.

In our proposal, the reward function is based on the feedback from a set of *simulations*, which are designed to maintain a resemblance in behaviour of normal users and attackers inside the network. The principal idea is that an optimal mitigation strategy would ideally hinder the activity of MHs, without affecting the experience of BHs. Hence, the RL agent can assess the fitness of its configurations at any point, by looking at the performance feedback from these simulations. For instance, a combination of QoS metrics from BHs and attack success rate from MHs can be used to constitute the reward function.

Using simulations, we have a general way to train models that are able to perform mitigation against arbitrary attacks. Intuitively, as long as the simulations are chosen to fit the desired mitigation task and the RL model has enough capacity, the agent will be able to handle a wide range of attacks that are *similar* to the ones performed by MHs. The notion of similarity here is with respect to the attack class and not the exact fingerprint. Therefore, it is possible that even zero-day attacks that belong to a known attack class would be mitigated, as they will be classified in the same category by the RL model. Furthermore, since the agent benefits from BHs to characterize a *normal behaviour*, it is theoretically possible for it to tag and eliminate new unseen threats based on their deviation from the norm.

D. Network Observer

An integral part of our high-level model is the network observer, which provides the *state observation* to the RL agent. In essence, the network observer should provide a digest of the network state at different points in time. This contains useful raw information that can be processed to perform threat mitigation. While our model does not place any restrictions on the nature of the network observer, it is expected to be a system whose output can distinguish attackers from benign users, if analyzed properly. For instance, IDSs and flow-meters are good candidates, as they are highly likely to provide information that can help the agent single out the attackers.

Even with proper rules, IDSs are highly prone to false positives. This makes it hard for a traditional mitigation system to impose restrictions on a host that is deemed suspicious, solely based on the IDS alerts. Using the information from the IDS, ATMoS can carefully setup, as a counter measure, mitigation rules. For example, migrating hosts across Virtual Networks that are created specifically with different security architectures and policies, while monitoring the feedback from the simulations to ensure that these rules only affect MHs and not the BHs.

E. RL in Networking Problem Formulation Challenges

When applying RL to any real-world problem, the main objective is to define the states, actions, and reward to effectively solve the problem. We start with corresponding challenges and lead the discussion into our solution.

1) Action-set and State-set Design Challenges: It is difficult to imagine a set of countermeasures against all network attacks, let alone parameterize it as a finite discrete set of actions to be used as the set of possible actions for the agent in RL algorithms. This is one of the primary barriers of designing an RL agent for threat mitigation. Traditionally, attack countermeasures are defined specifically for each category of threat. For example, countermeasures against DDoS attacks include egress filtering, load balancing, deployment of honeypots, and traffic throttling [28].

One approach is to design a formal framework for these threat-specific defence methods, as done by Yau *et al.* [29] where the authors define the action-set as deploying traffic throttlers on routers to handle DDoS attacks. At each step, the agent might increase or decrease the throttling rate by 5%. However, there are a few major drawbacks to this approach:

- The action-set should be designed separately for each different type of attack. This would require an enormous

amount of effort from domain experts to come up with an all-encompassing action-set for the general use-case.
- The experience and learned know-hows from an RL solution can hardly be transferred to another, since the RL formulation of the threat mitigation problem can be so vastly different for various solutions.
- The size of the action-set can explode really fast based on the complexity of the defence method. Since RL is based on trial-and-error, a large action-set will cause the algorithm to take an extremely long time to converge, making it impractical for any real-world deployment.

Another intuitive approach, is to define the elements of the action-set as deploying a certain OpenFlow [30] rule. Although this might be thought of as the ultimate action-set, as it encompasses almost everything that the SDN controller can do, the size of the action-set will be immense. Finding well-generalized solutions in such a gigantic action-space would require a very complex model, along with very expensive and time-consuming training. This issue is further discussed in III-E2.

2) State-set & Action-set Size Challenges: In RL, the high dimensionality of states and the large number of actions can both be problematic. RL algorithms based on neural networks, such as DQN [3], allow the model to ingest a high-dimensional state while maintaining reasonable complexity. For example, to train models for playing video games, the state is often defined as the color value of all the screen pixels from every K^{th} frame of the game, but the value function is estimated using a neural network that takes this large state as an input and returns the estimated cumulative reward of taking each action from the given input as its output [31]. The use of Convolutional Neural Networks is also prevalent due to their effectiveness in finding patterns in visual input.

On the other hand, large action-sets are more complicated to manage. Current techniques in ML literature for managing large discrete action-sets, such as [32], rely on the action-set being in some sense *continuous i.e.*, the actions that are close to each other in the action-space domain would perform a *similar* thing in reality. This way, *function approximation* can be used to pick a close-to-perfect action. Unfortunately, the *continuity* property does not apply to standard representations of OpenFlow rules or anything involving network addresses. For example, a rule for dropping traffic coming from IP address 10.0.0.1, is likely to be close to the rule for dropping those of 10.0.0.2 in the action-space. Nevertheless, these actions are vastly different in the effect they have in reality. Finding a better representation of IP addresses and OpenFlow rules, is one possible approach to overcome this. For instance, Li *et al.* [33] have used the number of network hops between nodes to learn a more meaningful representation of IP addresses. However, coming up with one such method for the RL threat mitigation task is very challenging as it should reflect similar behaviour in network hosts.

F. State-space in ATMoS

The RL module in ATMoS makes use of the NFQ algorithm [34], which uses neural networks (NN) for estimating the expected cumulative reward from taking each action at each state, following Q-learning as a base algorithm. The regular Q-learning relies on a value-function Q that signifies the estimated expected reward of taking an action in a given state. The Q-learning update rule is:

$$Q(s', a) \leftarrow (1-\alpha) \cdot Q(s, a) + \alpha \cdot (r + \gamma \cdot \max_{a'} Q(s', a')) \quad (1)$$

Where α is the learning rate, γ is the *discount factor* indicating the trade-off between immediate and long-term reward. r, s, s' and a represent the reward, the previous state, the new state, and the action taken, respectively. NFQ extends the same concept to neural networks with the following update rule:

$$\begin{aligned} Y_k^Q &= r + \gamma \max_{a' \in A} Q(s', a' : \theta_k) \\ \theta_{k+1} &= \theta_k + \alpha \left(Y_k^Q - Q(s, a; \theta_k) \right) \nabla_{\theta_k} Q(s, a; \theta_k) \end{aligned} \quad (2)$$

The idea is that the Q-function can be implemented using a neural network. θ_k marks the weights of the network at step k. Y_k^Q indicates the target value of the Q-function, which is also used for calculating the MSE loss [35]. As mentioned in Section III-E2, using a neural network can help handle the large size of the state-space, allowing highly detailed information to be consumed by the RL agent for choosing actions. Hence, in ATMoS we are free to use a very large state-space.

To constitute the state, observations from the network observer are fetched periodically and aggregated. Later, they are ingested by the RL model as an $N_1 \times N_2 \times N_3$ 3-D tensor, as depicted in Fig. 3. N_1 is the number of tracked hosts, N_2 is an adjustable hyper-parameter that signifies the number of samples taken from the network observer at each training step, and N_3 is the dimensionality of the vector to encode each observation of a host. For example, in our proof-of-concept where the network observer is a traditional IDS, N_3 is the size of the vector used to represent each IDS alert that is simply a one-hot encoding of the alert type, while N_2 is the maximum number of alerts under consideration from each host at each step. Notice that due to the use of neural nets, N_1, N_2 and N_3 should be fixed throughout the training.

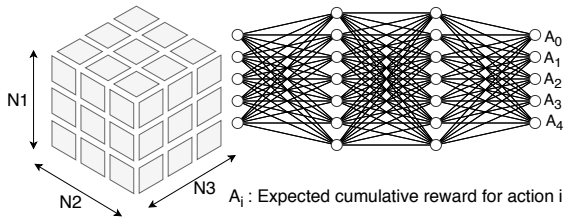

Fig. 3: Tensor representation of network observations at each step is fed into the neural network

This formulation, keeps a uniform style of representing network state, while remaining generalizable to different approaches in what this information should be. This state is fed as the input to the RL model's NN.

G. Action-space in ATMoS

In ATMoS, *Virtual Networks (VNs)* are leveraged to build a framework for mitigation, which can be controlled through a minimal action-set. As shown in Fig. 4, multiple VNs are deployed on top of the underlying SDN, which is convenient and has a low overhead using modern controllers, such as OpenDaylight. Each VN, embodies a different *security level*. This can be implemented using different network policies and functions (*e.g.*, different IDS/IPS systems, deep packet inspection engines, honeypots, traffic throttlers, *etc*.) in a way such that VNs with higher security levels would have more rigorous security measures.

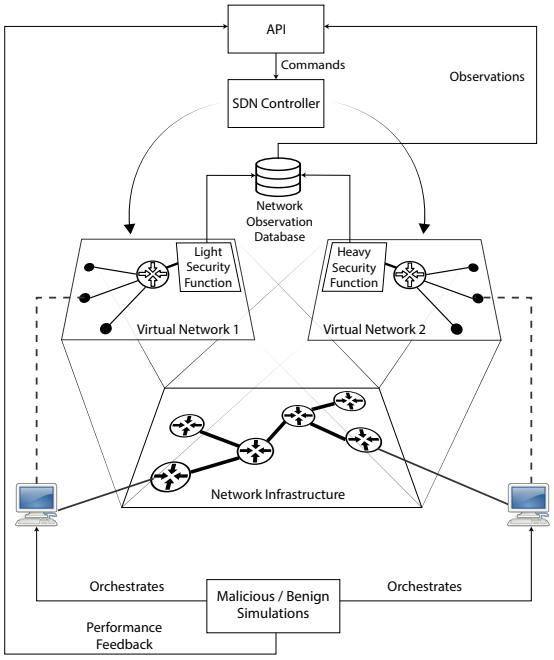

Fig. 4: The sample implementation of ATMoS architecture. Hosts are moved between two VNs while being oblivious to these changes. The API provides a unified gateway for interacting with the architecture.

The hosts are initially placed in the lowest security VN. The agent constantly monitors the network observations on all the hosts, and based on its trained model decides to move the hosts to a higher security level VN, if necessary. This migration is transparent to the host itself, as the VNs are only a logical view on top of the underlying network and the migration happens almost instantly.[2] Hence, the action-set can be defined as *migrating each host to a VN with higher or lower security level*. We also consider a *no-op i.e.*, a no operation action, for the case when nothing shall be done. Thus, for a network of K hosts, the size of the action-set is $2K + 1$. The simulations are deployed to the network as regular users, with the difference that they are capable of reporting their performance metrics. The network observer is orthogonal to the VNs *i.e.*, it monitors all hosts regardless of their VN.

[2]Our VN framework is very close to what can be achieved through traditional VLANs. However, VTNs enabled by SDN controllers allow for much higher flexibility than simple VLANs *i.e.*, virtual topologies

Fetching the observations, receiving the performance feedback from simulations to calculate the reward, and invoking the actions are all done through API calls, ensuring that the management module and the infrastructure are fully decoupled. The RL agent uses this API as its single point of contact with the rest of the network. Based on this architecture, we are able to provide a unified framework for developing a wide range of RL applications. We transfer the complexity and domain-specific nuances of designing the mitigation actions, to the network functions installed in the VNs and simplify the RL action-space. On the other hand, the network observer is a general scheme for incorporating arbitrary information and insights as the input of the RL model. Hence, our solution addresses the complexities of formulating threat mitigation as an RL problem (*cf.*, Sections III-E1 and III-E2), while not limiting its scope to a specific kind of threat. The results and experiences from one RL agent can thus be extrapolated to other cases, addressing the obstacles discussed in Section II.

IV. Evaluation

In this section, we demonstrate an implemented sample proof-of-concept, explain the technologies used, and present the evaluation results.

A. Technology Stack

We ran our experiments on Google Compute Engine on a Linux machine with 16 Intel Xeon 2.30 GHz cores and 60GB RAM. The SDN infrastructure was implemented using Containernet [36], a fork of Mininet [37], which allows using Docker containers as hosts on the network. For the network controller, OpenDaylight (ODL) was used since it is one of the most popular choices in the industry, and also its Virtual Tenant Network (VTN) plugin is a feature-rich addition that grants us the implementation of our virtual networks, described in Section III-G. Open vSwitch was used as the underlying software switch inside Mininet. We found that the operations using this stack are quite fast. The start-up of ODL with all the necessary plugins takes about 20.45 seconds on our hardware, with an extra 3.14 seconds for the VTN daemon. The latency for reassignment of a host's VN is in the tenth of a second time-scale, varying between 0.11s and 0.46s on our Python Flask API. This latency can be easily reduced in the future by optimizing the API code, if necessary.

MHs were implemented using a Python script, which in turn triggers *hping3*, a packet assembling tool that can also be used as a flooder. BHs were implemented using Google's Puppeteer library in NodeJS, which operates a headless Google Chrome instance for emulating a human user's behaviour. These simulations are explained in more detail in Section IV-B. The network observer, explained in Section III-D, in our experiment is a standard Snort IDS/IPS [38], which is piped to a MySQL database. Finally, a RESTful API is implemented in Python to act as the single gateway for the RL agent to interact with the SDN infrastructure. On the ML side, for convenience, an OpenAI Gym [39] environment is implemented, which wraps the calls to the REST API mentioned above, hence making the

RL code fully decoupled from the engineering aspects of the project and closer to standard RL problems. The agent itself uses an instance of the NFQ algorithm implemented using Tensorflow [40] and Keras.

B. Proof-of-Concept Setup

In our sample implementation of ATMoS there are only two VNs for two different security levels, as depicted in Fig. 4. In the low-security VN, Snort is running in passive mode acting as IDS, while in the high-security VN, Snort is deployed in-line and acts as an IPS that intercepts all the VN's traffic. Snort was chosen over Suricata and Bro as it is more fit for small to medium sized networks. To get the Snort alerts into our MySQL database efficiently, a spooler called Barnyard2, was used. Using an in-line IPS can be infeasible and unscalable for many networks as it can reduce the bandwidth and increase the latency. However, the RL agent learns to only place the most suspicious hosts in the high-security VN, to avoid degrading the performance of BHs, which is reflected in the reward function. Hence, the benign user's traffic will not go through the IPS and their QoS will not be affected.

In our experiments, we deployed n_b BHs referred to as *BH* k, for k in range 1 to n_b, in the rest of the paper. These hosts were built to mimic human benign web surfing, running a headless Chrome Browser that queries the Google Search Engine with random sentences generated from a dictionary. We also designed two kinds of MHs, one of which constantly performs a simple DoS attack, while the other one uses APT techniques. Although the APT MH runs the benign behaviour by default in the background, it is capable of adapting to the environment changes, sensing reachable local targets, and launching crafted attacks against vulnerable targets when in range. If no such target exists, the MH resorts to its normal benign behaviour.

Our proof-of-concept APT imitates human benign behaviour while periodically scanning the network and looking for vulnerable hosts. Only when certain target hosts are found, the APT launches tailored attacks against them, such as ICMP and SYN flooding attacks which were chosen for the sake of simplicity. In the presence of certain vulnerable hosts (*i.e.*, those who answer to ICMP or TCP pings), the MH starts attacking them with SYN or ICMP floods. All hosts constantly register their performance metrics (*e.g.*, attack success rate, page load time, *etc.*) by hosting a tiny RESTful API.

The NN used in our experiment receives the 3-D state tensor (network observation), in Section III-G, as the input and returns the expected cumulative reward of taking each action as its output. For 2 and 3 host experiments, the neural net has only one dense hidden layer of 4 neurons. For the other experiments, it has two such layers of 8 neurons. Learning rate is set to 0.01 at the beginning and decimated every thousand steps. The reward function in our experiments is a factor of the number of hosts placed in the correct VN, which allows for faster convergence. We also make use of the ϵ-greedy policy, which performs random actions with the probability ϵ at each step, setting the trade-off between exploration and exploitation. ϵ is set to ten percent at the beginning and lowered by an epsilon decay factor of 0.9 at the end of each episode. MSE and ReLU have been used as cost and activation functions, respectively. The hyper-parameter γ that sets the trade-off between valuing immediate or future rewards is set to 0.8.

C. Results

In our first experiment, we deployed our network with 1 BH and 1 MH. We compare two different scenarios when the malicious host runs an APT attack versus the case where a simple TCP SYN-flood attack is performed. We argue that the latter does not need sequential decision-making and hence it is an inherently simpler problem. As depicted in Fig. 5, we observe that both models converge although the model in SYN-flood experiment converges *much* faster, in only 150 iterations as opposed to 1400 iterations for APT. The vertical bars in all the figures signify epochs when random actions are taken for exploration purposes. In epochs where the vertical bar is not present, the RL acts on what it already learned, in other words exploiting its previous knowledge acquired during the experiment so far. All convergence diagrams show reward on y-axis against time and iterations on x-axis.

We run our second experiment with 2 BHs and 1 MH, acting like an APT attacker. As shown in Fig. 6, the convergence is seen around 1800 epochs and demonstrate to keep its knowledge afterwards. The y-axis ranges from -3 to 0 showing better rewards when closer to 0, and worse rewards, signifying failed mitigation of attack, on the more negative values. By design the epsilon decay allows us to exploit more and explore less, as we run more epochs with no change.

We investigate optimizing our convergence by teaching the model that sometimes it looses the round, reaching terminal state. By reaching terminal state a highly negative reward is given to the agent, stressing it to learn that its decisions took a really wrong turn. We observe that this gives us slightly better experimental results, as it converges in 1000 steps, with loosing state, rather than 1400, with non-loosing state, in the 1 MH (APT) and 1 BH experiment, as depicted in Fig. 7. Table I shows a summary of our experiments.

No. Hosts	Attack	Loss State	Iters. to Convergence
2	SYN	No	150
2	APT	No	1400
2	APT	Yes	1000
3	APT	No	1800

TABLE I: Summary of results (iterations to convergence) in our experiments

V. Future Work

Our work lays the foundation for many new opportunities, which could be used to adapt ATMoS to handle more complicated attacks. In this section, we list a few research directions that are worth pursuing.

- **Refined Network Observation**—In our evaluations, we used an IDS/IPS for the network observer as a proof-of-concept. We urge the use of a more sophisticated network

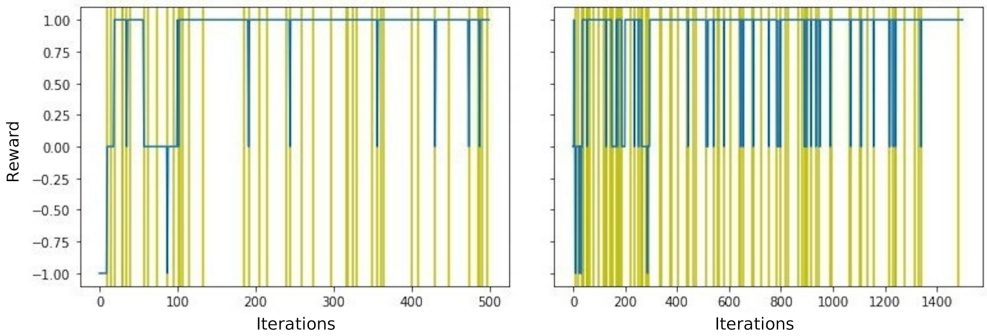

Fig. 5: Convergence diagram for the experiment with 1 BH and 1 MH running TCP SYN Flooder (on left) and running APT (on right)

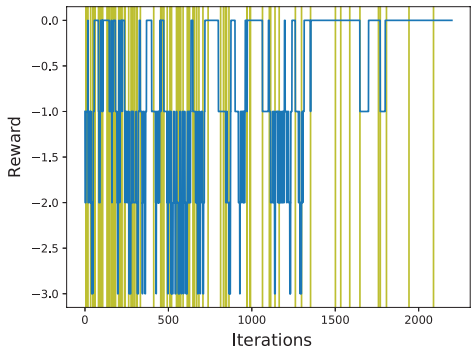

Fig. 6: Convergence diagram for the experiment with 1 MH running APT and 2 BHs

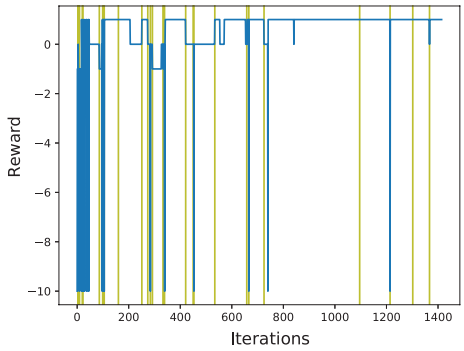

Fig. 7: Convergence diagram for the experiment with 1 MH running APT and 1 BH, where actions with no consequence cause a losing terminal state

observer, especially one that would make use of graph-based features, such as those proposed in [41]. Such data sources can provide better insights to the RL agent.

- **Applicability to Unseen Attacks**—Another direction worth exploring, is the potential of an RL system to mitigate zero-day attacks. For instance, having the model trained for defence against a set of attacks and evaluating its performance against one that has not been present in training. Theoretically, the model is capable to grasp the patterns, which constitute "normal" and "malicious" behaviour, and extrapolate them to unseen attacks, given the network observer is thoroughly designed and the model has enough capacity.
- **VN Design Based on Domain-specific Knowledge**—There is room for expanding the design of VNs. Multiple VNs can be used with different VNFs placed inside them *e.g.*, Sniffers, Honeypots, SSL DPI engines. Another possible approach is to have VNs with different bandwidth throttling rates so that moving a host to a certain VN would mean limiting its bandwidth. This encompass approaches already seen in the surveyed literature [6], [8].
- **Behaviour Profiling Enhancement**—The feedback from the simulated hosts could be normalized to reduce the high variance of the reward signal. It is also worthwhile to make the MHs run a more complicated variant of APT, with higher logical complexity. For instance, the MH could detect security mechanisms in place, and lie dormant when they are activated, even if a vulnerable host is seen in range making it harder to detect.
- **RL Algorithms**—We believe that experimenting with different RL algorithms (*e.g.*, NFQ, DDQN, A3C, *etc.*), tweaking the hyper parameters in NFQ (*e.g.*, neural network architecture) and comparing the results is worth exploring. Training the RL agent on ATMoS in an adversarial setting is also of paramount importance for the robustness of the model in real networks.

VI. Conclusion

In this paper, we defined a framework to implement RL-based solutions for threat mitigation in SDN. We also publicly released an open-source implementation of the framework, to promote reproducible research. To evaluate the applicability of our techniques, we put ATMoS against an APT attack and analyze how it detects and mitigates the actors. Our results show promising potential using this architecture and framework to mitigate APT attacks. Cyber-attacks have advanced in the recent years, to the extent where traditional signature and heuristic based detectors paired with manual human administered mitigation are not catching up. We hope research efforts in automating threat mitigation would be facilitated and accelerated with the help of ATMoS.

Acknowledgements

This work is supported in part by the Royal Bank of Canada, NSERC CRD Grant No. 530335, ezSec Inc. and Google LLC. Authors Iman Akbari and Ezzeldin Tahoun have contributed equally to this paper.

REFERENCES

[1] S. Ayoubi *et al.*, "Machine learning for cognitive network management," *IEEE Communications Magazine*, vol. 56, no. 1, pp. 158–165, 2018.

[2] R. Boutaba *et al.*, "A comprehensive survey on machine learning for networking: evolution, applications and research opportunities," *Journal of Internet Services and Applications*, vol. 9, no. 1, p. 16, Jun 2018.

[3] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, p. 529, 2015.

[4] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *Journal of artificial intelligence research*, vol. 4, pp. 237–285, 1996.

[5] Y. Han *et al.*, "Reinforcement learning for autonomous defence in software-defined networking," in *International Conference on Decision and Game Theory for Security (GameSec)*. Springer, 2018, pp. 145–165.

[6] Y. Liu *et al.*, "Deep reinforcement learning based smart mitigation of DDoS flooding in software-defined networks," in *23rd International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD)*. IEEE, 2018, pp. 1–6.

[7] A. VishnuPriya, "Reinforcement learning-based DoS mitigation in software defined networks," in *International Conference on Communications and Cyber Physical Engineering (ICCCE)*. Springer, 2018, pp. 393–401.

[8] L. Sampaio *et al.*, "Using NFV and reinforcement learning for anomalies detection and mitigation in SDN," in *IEEE Symposium on Computers and Communications (ISCC)*, 2018, pp. 432–437.

[9] T. Lillicrap *et al.*, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[10] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," in *Thirtieth AAAI conference on artificial intelligence (AAAI-16)*, 2016.

[11] V. Mnih *et al.*, "Asynchronous methods for deep reinforcement learning," in *International conference on machine learning*, 2016, pp. 1928–1937.

[12] K. Malialis and D. Kudenko, "Distributed response to network intrusions using multiagent reinforcement learning," *Engineering Applications of Artificial Intelligence*, vol. 41, pp. 270–284, 2015.

[13] J. Mirkovic and P. Reiher, "A taxonomy of ddos attack and ddos defense mechanisms," *ACM SIGCOMM Computer Communication Review*, vol. 34, no. 2, pp. 39–53, 2004.

[14] A. Zaalouk *et al.*, "Orchsec: An orchestrator-based architecture for enhancing network-security using network monitoring and sdn control functions." in *IEEE/IFIP Network Operations and Management Symposium (NOMS)*, 2014, pp. 1–9.

[15] J. Wang *et al.*, "Detecting and mitigating target link-flooding attacks using sdn," *IEEE Transactions on Dependable and Secure Computing (TDSC)*, 2018.

[16] J. Wu *et al.*, "Big data analysis-based secure cluster management for optimized control plane in software-defined networks," *IEEE Transactions on Network and Service Management*, vol. 15, no. 1, pp. 27–38, 2018.

[17] S. Lim *et al.*, "A SDN-oriented DDoS blocking scheme for botnet-based attacks," in *IEEE Sixth International Conference on Ubiquitous and Future Networks (ICUFN)*, 2014, pp. 63–68.

[18] D. Hu, P. Hong, and Y. Chen, "FADM: DDoS flooding attack detection and mitigation system in software-defined networking," in *IEEE Global Communications Conference (GLOBECOM)*, 2017, pp. 1–7.

[19] S. Shin *et al.*, "Fresco: Modular composable security services for software-defined networks," in *20th Annual Network & Distributed System Security Symposium*. NDSS, 2013.

[20] S. A. Mehdi, J. Khalid, and S. A. Khayam, "Revisiting traffic anomaly detection using software defined networking," in *International workshop on recent advances in intrusion detection*. Springer, 2011, pp. 161–180.

[21] R. Braga, E. Mota, and A. Passito, "Lightweight DDoS flooding attack detection using NOX/OpenFlow." in *35th Annual IEEE Conference on Local Computer Networks (LCN)*, vol. 10, 2010, pp. 408–415.

[22] D. Jankowski and M. Amanowicz, "On efficiency of selected machine learning algorithms for intrusion detection in software defined networks," *International Journal of Electronics and Telecommunications*, vol. 62, no. 3, pp. 247–252, 2016.

[23] T. Tang *et al.*, "Deep learning approach for network intrusion detection in software defined networking," in *IEEE International Conference on Wireless Networks and Mobile Communications (WINCOM)*, 2016, pp. 258–263.

[24] Q. Niyaz, W. Sun, and A. Y. Javaid, "A deep learning based DDoS detection system in software-defined networking (SDN)."

[25] N. Sultana *et al.*, "Survey on SDN based network intrusion detection system using machine learning approaches," *Peer-to-Peer Networking and Applications*, vol. 12, no. 2, pp. 493–501, 2019.

[26] C. Tankard, "Advanced persistent threats and how to monitor and deter them," *Network security*, vol. 2011, no. 8, pp. 16–19, 2011.

[27] D. Kennedy *et al.*, *Metasploit: the penetration tester's guide*. No Starch Press, 2011.

[28] S. Specht and R. Lee, "Distributed denial of service: Taxonomies of attacks, tools, and countermeasures." in *ISCA International Conference on Parallel and Distributed Computing (and Communications) Systems (ISCA PDS)*, 2004, pp. 543–550.

[29] D. Yau *et al.*, "Defending against distributed denial-of-service attacks with max-min fair server-centric router throttles," *IEEE/ACM Transactions on Networking*, vol. 13, no. 1, pp. 29–42, 2005.

[30] N. McKeown *et al.*, "Openflow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[31] V. Mnih *et al.*, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[32] G. Dulac-Arnold *et al.*, "Deep reinforcement learning in large discrete action spaces," *arXiv preprint arXiv:1512.07679*, 2015.

[33] M. Li *et al.*, "Deep learning ip network representations," in *ACM SIGCOMM Workshop on Big Data Analytics and Machine Learning for Data Communication Networks (Big-DAMA)*, 2018, pp. 33–39.

[34] M. Riedmiller, "Neural fitted Q iteration–first experiences with a data efficient neural reinforcement learning method," in *European Conference on Machine Learning (ECML)*. Springer, 2005, pp. 317–328.

[35] P. François-Lavet *et al.*, "An introduction to deep reinforcement learning," *Foundations and Trends in Machine Learning*, vol. 11, no. 3-4, pp. 219–354, 2018.

[36] M. Peuster, H. Karl, and S. Van Rossem, "MeDICINE: Rapid prototyping of production-ready network services in multi-pop environments," in *IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, Nov 2016, pp. 148–153.

[37] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: rapid prototyping for software-defined networks," in *the 9th ACM SIGCOMM Workshop on Hot Topics in Networks (HotNets-IX)*, 2010, p. 19.

[38] M. Roesch, "Snort: Lightweight intrusion detection for networks." in *Lisa*, vol. 99, no. 1, 1999, pp. 229–238.

[39] G. Brockman *et al.*, "OpenAI gym," *arXiv preprint arXiv:1606.01540*, 2016.

[40] M. Abadi *et al.*, "Tensorflow: A system for large-scale machine learning," in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pp. 265–283.

[41] A. Abou Daya *et al.*, "A graph-based machine learning approach for bot detection," in *IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, 2019, pp. 144–152.