Author(s): Zolotukhin, Mikhail; Kumar, Sanjay; Hämäläinen, Timo

Title: Reinforcement Learning for Attack Mitigation in SDN-enabled Networks

Year: 2020

Version: Accepted version (Final draft)

Please cite the original version:

Zolotukhin, M., Kumar, S., & Hämäläinen, T. (2020). Reinforcement Learning for Attack Mitigation in SDN-enabled Networks. In F. De Turck, P. Chemouil, T. Wauters, M. Faten Zhani, W. Cerroni, R. Pasquini, & Z. Zhu (Eds.), NetSoft 2020 : Proceedings of the 2020 IEEE Conference on Network Softwarization. Bridging the Gap Between AI and Network Softwarization (pp. 282-286). IEEE. https://doi.org/10.1109/NetSoft48620.2020.9165383

Reinforcement Learning for Attack Mitigation in SDN-enabled Networks

Mikhail Zolotukhin
Faculty of Information Technology
University of Jyväskylä
Jyväskylä, Finland
mizolotu@jyu.fi

Sanjay Kumar
Faculty of Information Technology
University of Jyväskylä
Jyväskylä, Finland
sanjay.k.kumar@jyu.fi

Timo Hämäläinen
Faculty of Information Technology
University of Jyväskylä
Jyväskylä, Finland
timoh@jyu.fi

***Abstract*—With the recent progress in the development of low-budget sensors and machine-to-machine communication, the Internet-of-Things has attracted considerable attention. Unfortunately, many of today's smart devices are rushed to market with little consideration for basic security and privacy protection making them easy targets for various attacks. Unfortunately, organizations and network providers use mostly manual workflows to address malware-related incidents and therefore they are able to prevent neither attack damage nor potential attacks in the future. Thus, there is a need for a defense system that would not only detect an intrusion on time, but also would make the most optimal real-time crisis-action decision on how the network security policy should be modified in order to mitigate the threat. In this study, we are aiming to reach this goal relying on advanced technologies that have recently emerged in the area of cloud computing and network virtualization. We are proposing an intelligent defense system implemented as a reinforcement machine learning agent that processes current network state and takes a set of necessary actions in form of software-defined networking flows to redirect certain network traffic to virtual appliances. We also implement a proof-of-concept of the system and evaluate a couple of state-of-art reinforcement learning algorithms for mitigating three basic network attacks against a small realistic network environment.**

***Index Terms*—network security, machine learning, reinforcement learning, software-defined networking, network function virtualization**

I. Introduction

Increasing computing and connectivity capabilities of smart devices in conjunction with users and organizations prioritizing access convenience over security makes such devices valuable asset for cyber criminals. The intrusion detection on IoT is limited due to lack of efficient malware signatures caused by diversity of processor architectures employed by different vendors [1]. In addition to that, owners use mostly manual workflows to address malware-related incidents and therefore they are able to prevent neither attack damage nor potential attacks in the future. Furthermore, since not all devices support over-the-air security updates, or updates without downtime, they might need to be physically accessed or temporarily pulled from production. Thus, many connected smart devices may remain vulnerable and potentially infected for long time resulting in a material loss of revenue and significant costs incurred by not only device owners, but also users and organizations targeted by the attackers as well as network operators and service providers.

In this study, we investigate a possibility to overcome the aforementioned challenges by employing recent advances in reinforcement machine learning to evaluate risks related to an intrusion and make the most optimal real-time crisis-action decision on the network security policy. The actions are then implemented in the form of software-defined network (SDN) flows which are pushed to the corresponding forwarding elements of the network under protection. Reinforcement learning (RL) is a machine learning paradigm in which software agents automatically determine the ideal behavior within a specific context by continually making value judgments to select good actions over bad. Reinforcement learning algorithms can be used to solve very complex problems that cannot be solved by conventional techniques as they aim to achieve long-term results correcting the errors occurred during the training process.

However, deep RL has thus far not seen wide adoption in network security frameworks due to several practical obstacles. Probably, the biggest obstacle of applying reinforcement learning algorithms in real-life environments is its low sample efficiency. Even though, recent progress in developing better RL algorithms has led to significantly better sample efficiency, even in dynamically complicated tasks [2], [3], it still remains a challenge. Another major, often underappreciated, obstacle is reward function specification. Most of the time it is carefully shaped [4] which can be a significant challenge, as one must additionally build a perception system that allows computing dense rewards on state representations. Shaping a reward function so that an agent can learn from it is also a manual process that requires considerable manual effort. An ideal RL system would learn from rewards that are natural and easy to specify. These two obstacles can be partially eliminated by designing a realistic simulation software to emulate the real world environments such that the RL agents can be trained in simulations. Even if such software is real-time, we can solve the sample efficiency challenge by running several simulations in parallel. Most of the state-of-art reinforcement learning algorithms are capable to learn from many environments at once by design. The reward shaping challenge can also be solved in the simulated environment as we can construct any

978-1-7281-5684-2/20/$31.00 ©2020 IEEE

sort of dense or sparse reward function using data obtained from the real world model used in the simulator.

To the best of our knowledge, there are not many studies that focus on the application of RL in network security. Speaking of software-defined networks, RL algorithms have been mostly employed for intelligent route selection [5]. Other security applications of RL include malware samples detection [6], DoS attacks prevention [7], evaluation of anti-malware engines [8] and intelligent honeypot construction [9]. There are also studies focusing on attack detection by routing certain network traffic through a particular set of security middle boxes. For example, ASD and NAD functions proposed in [11] for 5G networks employ deep learning approach for both anomaly symptom detection (ASD) and network anomaly detection (NAD). We aim to improve this approach by enabling the AI agent to learn an optimal ensemble of security service functions for each particular state of the network environment to allow for a more accurate and resource-efficient anomaly detection. PSI, Precise Security Instrumentation, described in [10] translates high-level security posture into per-device intents, which are further enforced by launching virtual security appliances and tunneling each devices traffic through a particular set of such middle-boxes. PSI requires an administrator to manually enter a security policy according to which the system configures security appliances and forwards traffic between them. Furthermore, PSI does not enable more advanced attack detection rather focusing on isolation and flexibility. The framework described in our proposal addresses both of these issues.

The main purpose of this study is to investigate how deep reinforcement learning can be applied for real-time attack mitigation in SDN-enabled networks and evaluate several of its state-of-art algorithms by mitigating realistic attacks in a small network environment. The rest of the paper is organized as follows. The system implementation is outlined in Section II. Section III presents performance results of several RL algorithms for mitigating several basic attacks against a small virtual environment. Section IV concludes the paper and outlines future work.

II. System Implementation

As briefly mentioned in the introduction, the key idea of the defense framework proposed in this study is relies on an RL agent that pushes flows to the SDN controller in order to affect certain connections depending on the state of the network environment. The agent's actions may result in redirecting traffic to a set of security middle boxes, dropping suspicious connections or allowing the traffic which was blocked previously to pass. Security appliances reside on cloud compute servers and they may include intrusion detection systems, honeypots, firewalls, and proxies. Building security middle boxes that share commodity hardware allows us to reduce the costs significantly as one commodity server can support up to hundreds of such appliances. SDN capabilities that include global visibility of the network state and run-time manipulation of traffic forwarding rules allow one to forward traffic from the network under protection to these appliances as well as connect the appliances to each other.

In order to train the RL agent, virtual copy of the network under consideration is supposed to be created using the framework's cloud computing resources. These copies are expected to run the same applications as IoT devices in the target network (Figure 1). The biggest drawback of the reinforcement learning approach is its hunger for data: RL methods require to interact with the environment at each new training iteration. Furthermore, the stability of the RL methods can be increased by increasing the number of environments in which the agent is trained in parallel. Therefore, in order to train the RL-based intelligent agent that is able to control SDN flows in reasonable amount of time, multiple copies of the network environment under consideration, at least its key components, have to be created in such virtual gym. During the training phase, the agent then observes states of each such virtualized environment, takes necessary actions, and learns impact of each action taken in the state given simultaneously.

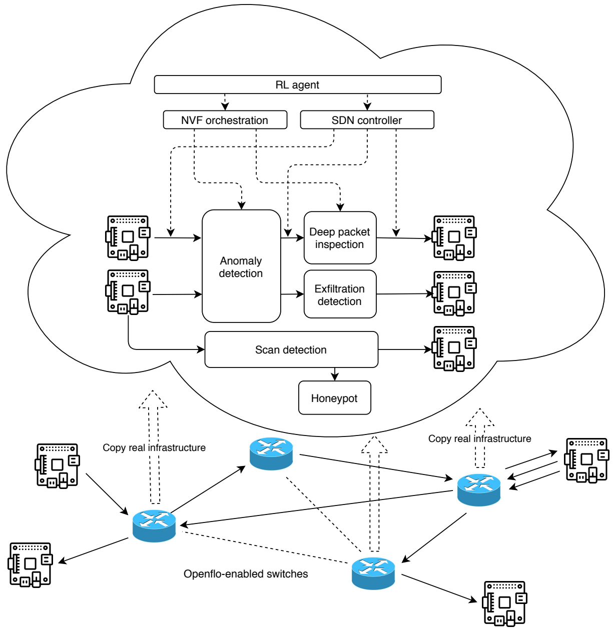


Fig. 1. The defense framework deployment.

The main purpose of the SDN controller in our defense framework is to transform the security intent of the AI core to SDN rules (flows) and push them to the switches. Such functionality can be implemented either as a separate module for the controller or an external application that uses RESTful APIs exposed by one or more plugins existing in the controller framework. Concerning the virtual security functions, there are many open-source intrusion detection and packet inspection software available that can be implemented as security middle boxes for timely signature-based attack detection and mitigation. Anomaly detection based functions can be implemented on top of existing packet sniffers or flow collectors. Such implementations require implementing modules to extract the most relevant features from the traffic under processing, build a model of normal user or application behavior during the training stage and classify new packets or flows as either normal or an outlier during the inference stage.

There are two possible options how the security functions can communicate with the rest of the defense framework: reactive and proactive. In the reactive case, the security middle-boxes require an application interface in order to send a statistics report to the AI core and process commands received from it. Once the RL agent has received such report, it makes a decision on what actions should be applied to this flow and sends required reconfiguration commands to the corresponding security functions for optimal threat mitigation. With proactive approach, the security functions require to label the flows under interest using for example flow tagging [12]. Proactive implementation requires the flow rules pushed by the controller to account for these flow tags in order to forward the packets to the destination or the next-hop virtual network function (VNF). In this case, the AI core is responsible for predicting the next possible states of the network environment enabling the SDN controller to modify flow tables in order to account for all possible tags assigned by the running VNFs. Even though the proactive approach may needlessly increase computing and storage resource consumption, it would be able to decrease the latency caused by pushing additional SDN flows to the switches and security downtime during the reconfiguration of security appliances.

The RL agent requires to select a set of countermeasures to mitigate the attack detected automatically. Most of the research studies aiming at dynamic attack countermeasure selection formulate selections of countermeasures into a multi-objective optimization problem that are not designed to adapt to constantly changing state of the network struggling to perform online resource scheduling and decision making [13], [14]. In theory, reinforcement learning approach should be capable to solve the problem of dynamic security appliance chaining as the RL agent takes into account state of each node in the network. The RL agent can also be trained to react to the ongoing attack by launching additional security appliances, however, in this study, we assume that all security appliances implemented are constantly active and there is no need to launch additional ones. Thus, the computing and network resources are used as a constraint the solution must satisfy when maximizing the reward function. Speaking of the reward, there are three key metrics that can be used to implement this function, namely attack damage cost, countermeasure positive effect and countermeasure negative impact [14]. The resulting RL agent is supposed to be trained to maximize the security performance and minimize the security impact on service quality. In our attack model, we assume that an attacker can be located either outside or inside of the network under protection. The external attacker's primary goal is to exploit and compromise vulnerable devices, whereas the internal attacker focuses on using infected devices as a tool to attack other services. Thus, the security performance is supposed to account for both the number of compromised devices and servers as well as volumes of malicious traffic sent towards external services. The security impact on service quality can be estimated by counting dropped connections and measuring jitter and latency in the network environment.

III. Experimental Results

First, we implemented the attack mitigation system proposed in this study in open-source software platform for cloud computing Openstack integrated with SDN controller Opendaylight, as shown in Figure 2. Both virtualized devices and security VNFs are created as appliances in the resulting cloud and connected to each other via Openflow-enabled OpenVSwitches. Later, however, we switched to Docker containers since all appliances used in the experiments had the same x86_64 architecture. This allows us to run more virtual copies of the network environment using the same amount of computing and storage resources. As a result, each copy of the environment is deployed in a dedicated virtual machine with applications belonging to different devices running in different containers.

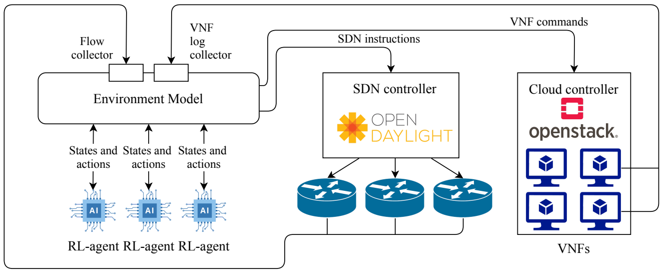


Fig. 2. The RL agent training ground.

Concerning security appliances, we launch several types of appliances: traditional signature-based IDS Snort, custom machine learning based anomaly detection IDS and honeypots that are essentially virtual machines with open SSH and Telnet ports and several user accounts with very easy-to-guess passwords. For each VNF, we implemented simple web interface that allows one to request their logs over TCP protocol.

Signature-based IDS. First, we launch Snort appliance that uses the latest sets of community rules. However, it turns out, that those sets do not include rules for some basic intrusions, e.g. SSH password brute force attack and DNS tunneling. For this reason, we implemented a simple set of custom rules by analyzing legitimate network traffic generated in the network environment under consideration. We find the maximum amount of packets of a particular type sent to a particular host per second and generate a list of rules that throw an alert if one of such thresholds is exceeded. This turns Snort into a basic anomaly-based detection system, since its rules are configured based on the normal behavior patterns. We implement two types of Snort middle boxes: one uses community rules and another relies on our custom rules.

Anomaly-based IDS. In order to detect anomalous payloads, we rely on centroid-based clustering. A small set of the normal traffic collected in advance before the RL agent training starts is used to divide 2-grams of the packet payloads into several clusters. During the inference phase, we evaluate resulting clusters using a mixed set of normal and malicious traffic. If the distance between a new sample and its nearest cluster centroid exceeds the expected value, the sample is marked as anomalous. The clustering algorithms we have tested include k-means, partitioning around medoids, fuzzy c-means, growing neural gas, and self-organizing maps (SOM). In our case, SOM have outperformed its analogues in detection of both anomalous DNS queries and HTTP requests. For this reason, we implement two types of anomaly detection middle boxes for DNS and HTTP traffic separately with the clusters being constructed with SOM algorithm [15].

Honeypot. We launch a standard Linux box, install SSH and Telnet server on it and add several users with default name-password combinations used by Mirai botnet [16]. Other services that connect to the network are disabled. Thus, if the honeypot appliance attempts to establish a connection with an external host, we mark this host as suspicious.

Firewall. We use SDN flows to block suspicious connections by installing rules to drop packets from a particular source host to a particular destination socket that use a particular IP protocol. We push these SDN flows to the switches with two different timeout values: zero, i.e. no timeout constraint, and one, i.e. the rule lasts only one second to block the connection in the current time window. We also implement pass rule that removes all SDN flows that affect certain network traffic from the controller storage.

We initialize flow tables of each SDN switch with one single flow that forwards packets to the next table, until the last table is reached. SDN flows to drop packets or redirect them to a particular security appliance are pushed to the dedicated flow table with priority higher than default forwarding rule. The last table outputs packet to the physical port of its destination as well as mirrors packet to a special patch port, on which we run a flow collector. For each network activity from a particular source host to a particular destination socket we extract a set of features that include destination port, packet size, TCP flag counts, and several others. Furthermore, security alerts are requested from the corresponding appliances and added to the feature vectors. The full list of features is shown in Table I.

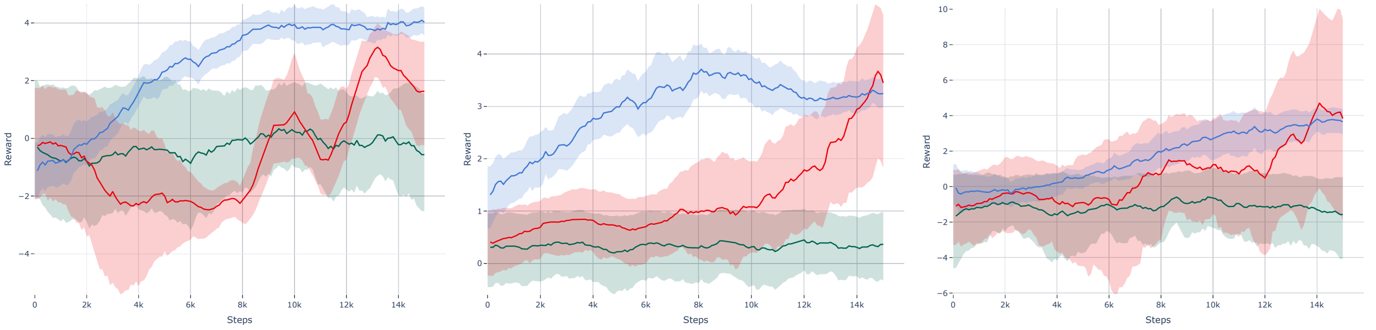
TABLE I
ENVIRONMENT STATE FEATURES.

Traffic data	Features
Port	Number of unique source ports, destination port
Packet counts	Number of requests, number of replies
Packet size	Minimal, maximal and average size
TCP flags	Number of FIN, SYN, RST, PSH and ACK flags
Protocol	IP protocol number
Security alerts	Number of alerts generated by different appliances in the recent time window
Security logs	Number of alerts generated by different appliances during the entire length of the experiment
SDN flows	Number of rules pushed to the SDN controller that affect the traffic flow

In order to evaluate the framework proposed, we designed a simple Python application that runs on each IoT device and sends a random sequence of alphanumeric symbols to one of the external data servers. In addition, every arbitrary amount of time each device connects to a randomly selected update server and requests several files from it. Furthermore, some devices are accessed via SSH by external entities imitating the administration process. DNS queries are resolved with the help of the internal DNS server. To generate malicious traffic, we implemented a simple Mirai-like malware with three attack capabilities. First, the malware scans its local network looking for open SSH server ports and, in case such server is found, it attempts to login to the server using a predefined list of user-password combinations. If the correct password has been found, the malware initiates download of its copy to the compromised device from an external server. When the download is complete, the malware initiates a HTTP connection to its C&C server to inform that the attack has been successful. The second attack type performed by the malware uses DNS tunneling to exfiltrate a randomly selected file found on the device to the attacker server using scheme similar to the C&C channel. The DNS server is configured in advance to forward such queries to the domain that belongs to the attacker. Finally, once multiple devices are infected with the malware, the attacker performs an application-based slow DDoS attack Slowloris against one of the data servers used by the legitimate application by sending never ending HTTP requests.

To train RL agent, we use OpenAI gym [17] to implement the front-end for the virtualized environment. We run 4 copies of the environment in parallel. The training process is divided into 1000 episodes. Each episode lasts 15 seconds, during which one of the attacks mentioned above is performed. The RL agent is implemented using OpenAI baselines [18]. The state returned to the agent is essentially the list of active and blocked flows with feature vectors described in Table I. The agent selects one of the actions for each flow that are sent to the environment back-end where they are transformed to SDN rules. The reward for the action is proportional to the number of packets transferred during the most recent time window and it is calculated for each flow separately. The proportion coefficients are positive for legitimate traffic and negative for the malicious one. The exact values of the coefficients are estimated by running the attack without the agent and counting the average number of packets that are sent by the application and the attacker. The coefficients are then calculated as such a way, that the total reward without the agent's intervention is equal to zero.

We test two state-of-art RL algorithms: deep Q-network (DQN) [19] and proximal policy optimization (PPO) [20] using multi-layer perceptron (MLP) as both policy and value function. The MLP consists of two layers each of which includes 512 neurons. In case of DQN, first 80 % of the episodes are used for ϵ-greedy exploration with ϵ value decreasing from one to zero. For PPO, we collect data from entire episode to calculate cumulative rewards and advantages for each unique host-to-socket tuple, before dividing the resulting dataset to mini-batches which are used to train both the critic and the



(a) SSH password brute-force

(b) DNS tunneling

(c) Slowloris DDoS

Fig. 3. DQN (red) and PPO (blue) performance for three different network attacks. Green line corresponds to ”do nothing” policy.

actor network. With DQN, for each such tuple, we store the state vector, the action taken by the agent, the reward and the feature vector of the same tuple in the next time window, only if there is still active connection. Otherwise, the state is marked as the terminate for this particular tuple. Figure 3 shows the evolution of the reward function throughout the training episodes for both DQN and PPO in case of three attacks mentioned. As one can notice, both algorithms are able to identify and block malicious connections reducing the number of malicious flows to minimum and subsequently increasing the reward value.

IV. Conclusion

The main contribution of this research is developing a proof-of-concept of an intelligent network defense system which allows customers to detect and mitigate attacks performed against their smart devices by letting an artificial intelligent agent control network security policy. We evaluated two state-of-art reinforcement learning algorithms for mitigating three basic network attacks against a small virtual network environment. The results show that such approach can be employed to decrease impact of the attacks in a small private network, however further investigation is required in order to estimate viability of the method in big production environments. In the future, we are planning to enhance our solution by developing more efficient and flexible reinforcement learning algorithms. We are also aiming to improve the scalability of the framework proposed and evaluate the system performance for bigger network environments. Finally, we are going to experiment with applications and real malware samples in order to evaluate the defense system capabilities in realistic attack scenarios.

References

[1] M. Alhanahnah, Q. Lin, Q. Yan, N. Zhang, and Z. Chen. Efficient signature generation for classifying cross-architecture IoT malware. Proc. of IEEE Conf. on Communications and Network Security, pp. 1–9, 2018.

[2] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor, Proc. of ICML, 2018.

[3] M. Hessel, J. Modayil, H. Van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Azar, and D. Silver. Rainbow: Combining improvements in deep reinforcement learning. Proc. of AAAI, 2018.

[4] I. Popov, N.M. Heess, T.P. Lillicrap, R. Hafner, G. Barth-Maron, M. Vecerík, T. Lampe, Y. Tassa, T. Erez, and M.A. Riedmiller. Data-efficient Deep Reinforcement Learning for Dexterous Manipulation. ArXiv, abs/1704.03073, 2018.

[5] S.-C. Lin, I. F. Akyildiz, P. Wang, M. Luo, Qos-aware adaptive routing in multi-layer hierarchical software defined networks: a reinforcement learning approach. Proc. of IEEE Int. Conf. on Services Computing, pp. 25–33, 2016.

[6] X. Xu, Adaptive intrusion detection based on machine learning: feature extraction, classifier construction and sequential pattern prediction, International Journal of Web Services Practices 2 (1–2), pp. 49–58, 2006.

[7] Servin, D. Kudenko, Multi-agent reinforcement learning for intrusion detection, in: Adaptive Agents and Multi-Agent Systems III. Adaptation and Multi-Agent Learning, Springer, 2008, pp. 211–223.

[8] Z. Fang, J. Wang, B. Li, S. Wu, Y. Zhou and H. Huang. Evading Anti-malware Engines with Deep Reinforcement Learning. IEEE Access, 2019.

[9] T. Luo, Z. Xu, X. Jin, Y. Jia, and X. Ouyang. Iot candy jar : Towards an intelligent-interaction honeypot for iot devices. Proc. of Black Hat, 2017.

[10] T. Yu, S. Fayaz, M. Collins, V. Sekar, and S. Seshan. Psi: Precise security instrumentation for enterprise networks. Proc. of the 24th Network and Distributed System Security Symposium, 2017.

[11] L. Fernandez Maimo, L. Perales Gomez, F. J. Garcia Clemente, et al. A Self-Adaptive Deep Learning-Based System for Anomaly Detection in 5G Networks. IEEE Access, vol. 6, pp. 7700–7712, 2018.

[12] S. K. Fayazbakhsh, L. Chiang, V. Sekar, M. Yu, and J. C. Mogul. Enforcing network-wide policies in the presence of dynamic middlebox actions using FlowTags. Proc. of NSDI, pp. 533–546, 2014.

[13] C. Chung, P. Khatkar, T. Xing, J. Lee, and D. Huang. Nice: Network intrusion detection and countermeasure selec-tion in virtual network systems. IEEE transactions on dependable and secure computing, vol. 10, no. 4, pp. 198–211,2013.

[14] A. S. Sendi, H. Louafi, W. He, and M. Cheriet. Dynamic optimal countermeasure selection for intrusion response system. IEEE Transactions on Dependable and Secure Computing, vol. 15, no. 5, pp. 755–770, 2016.

[15] M. Zolotukhin, T. Hämäläinen, A. Juvonen. Growing Hierarchical Self-organising Maps for Online Anomaly Detection by using Network Logs. Proc. of WEBIST, pp. 633–642, 2012.

[16] M. Antonakakis, T. April, M. Bailey, et al. Understanding the mirai botnet. Proc. of the 26th USENIX Conference on Security Symposium (SEC). pp. 1093–1110, 2017.

[17] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang and Wojciech Zaremba. OpenAI Gym.arXiv:1606.01540, 2016.

[18] P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, Y. Wu, and P. Zhokhov. OpenAI Baselines. GitHub, https://github.com/openai/baselines, 2017.

[19] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra and Martin Riedmiller. Playing Atari with Deep Reinforcement Learning. arXiv:1312.5602, 2013.

[20] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, O. Klimov. Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347, 2017.