Author(s): Zolotukhin, Mikhail; Hämäläinen, Timo; Kotilainen, Pyry

Title: Intelligent Solutions for Attack Mitigation in Zero-Trust Environments

Year: 2022

Version: Accepted version (Final draft)

Noname manuscript No.
(will be inserted by the editor)

Intelligent Solutions for Attack Mitigation in Zero-Trust Environments

Mikhail Zolotukhin · Timo Hämäläinen · Pyry Kotilainen

Received: date / Accepted: date

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. Once a device has been compromised, it can become the starting point for accessing other elements of the network at the next stage of the attack, since traditional IT security castle-and-moat concept implies that nodes inside the private network trust each other. For these reasons, IoT will benefit from adapting a zero-trust networking model which requires strict identity verification for every person and device trying to access resources on a private network, regardless of whether they are located within or outside of the network perimeter. Implementing such model can however become challenging, as the access policies have to be updated dynamically in the context of constantly changing network environment. Thus, there is a need for an intelligent enhancement of the zero-trust network that would not only detect an intrusion on time, but also would make the most optimal real-time crisis-action decision on how the security policy should be modified in order to minimize the attack surface and the risk of subsequent attacks in the future. In this research project,

M. Zolotukhin
Faculty of Information Technology
University of Jyväskylä,
E-mail: mizolotu@jyu.fi

T. Hämäläinen
Faculty of Information Technology
University of Jyväskylä,
E-mail: timoh@jyu.fi

P. Kotilainen
Faculty of Information Technology
University of Jyväskylä,
E-mail: pyjopeko@jyu.fi

we are aiming to implement a prototype of such defense framework relying on advanced technologies that have recently emerged in the area of software-defined networking and network function virtualization. The intelligent core of the system proposed is planned to employ several reinforcement machine learning agents which process current network state and mitigate both external attacker intrusions and stealthy advanced persistent threats acting from inside of the network environment.

Keywords Network security · Deep learning · Reinforcement learning · Software-defined networking

1 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 in 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. A potential solution to these and other emerging challenges in IoT is employing zero-trust networking model, that implies that all data traffic generated must be untrusted, no matter if it has been generated from the internal or external network [2].

In this research, we aim to design and implement an intelligent zero-trust networking solution capable of detecting attacks initiated by both external attackers and smart devices from the inside, adapt detection models under constantly changing network context caused by adding new applications and services or discovering new vulnerabilities and attack vectors, make an optimal set of real-time crisis-action decisions on how the network security policy should be modified in order to reduce the ongoing attack surface and minimize the risk of subsequent attacks in the future. These decisions that may include permitting, denying, logging, redirecting, or instantiating certain traffic between end-points under consideration, are based on behavioral patterns observed in the network and log data obtained from multiple intrusion and anomaly detectors and deployed on the fly with the help of cutting-edge cloud computing technologies such as software-defined networking and network function virtualization. Our implementation of the decision making mechanism in the system proposed is planned to rely on recent advances in reinforcement learning (RL), 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. RL 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.

Recent advent of cutting-edge technologies such as cloud computing, mobile edge computing, network virtualization, software-defined networking (SDN) and network function virtualization (NFV) have changed the way in which network functions and devices are implemented, and also changed the way in which the network architectures are constructed. More specifically, the network equipment or device is now changing from closed, vendor specific to open and generic with SDN technology, which enables the separation of control and data planes, and allows networks to be programmed by using open interfaces. With NFV, network functions previously placed in costly hardware platforms are now implemented as software appliances located on low-cost commodity hardware or running in the cloud computing environment. In this context, the network security service provision has shifted toward replacing traditional proprietary middle-boxes by virtualized and cloud-based network functions in order to enable automatic security service provision.

Software-defined perimeter (SDP) is an architecture for zero trust that borrows concepts from SDN and NFV. An SDP controller functions as a broker of trust between a client and a gateway, which can flexibly establish a transport layer security tunnel terminating on the gateway inside the network perimeter, allowing access to applications. Each device establishes a unique VPN tunnel with the service that is requested, and the origin is cloaked from public view. Each device establishes a unique VPN tunnel with the service that is requested, while the origin is cloaked from public view. SDP relies on the concepts of network access control in an attempt to minimize the impact of existing and emerging network threats by adding authentication of the hosts. Similar to micro-segmentation, SDP enforces the principle of only providing access to the services that are required. Besides this authentication function the SDP controller can enforce authorization policies that may include host type, malware checks, time of day access, and other parameters. The data plane will typically rely on an overlay network to connect hosts via VPN tunnels. However, SDP approach has several drawbacks. For example, an SDP implementation usually requires usage of specific hardware and software gateways and controller appliances. Gateways may be needed at each site where applications are located making the deployment, management, and maintenance of this infrastructure challenging, especially in large globally distributed, high availability environments. In addition, security appliances are supposed to be configured to accept connections and allow traffic from the SDP gateways. Intrusion detection system and firewall rules introduce complexity, holes in the perimeter, and added IT maintenance. In this research project, we focus on solving these drawbacks with the help of state-of-art machine learning techniques.

The purpose of this study is to highlight the implementation process of the defense framework proposed. The rest of the document is organized as follows. In Section 2, we evaluate various deep learning algorithms needed for implementation. Reinforcement learning algorithms are discussed in Section 3. Traffic generation problem is addressed in Section 4. Section 5 outlines implementation of SDN flows and security VNFs. The resulting system prototype is discussed in Section 6. Section 7 concludes the report and outlines future work.

2 Intrusion detection with deep learning

Artificial intelligence and deep learning are revolutionizing almost every industry with a seemingly endless list of applications ranging from object recognition for systems in autonomous vehicles to helping doctors detect and diagnose cancer. This list includes multiple branches of the field of cyber-security that include intrusion detection, malware classification, network traffic analysis and many others. A deep neural network consists of multiple layers of nonlinear processing units. The main idea behind deep learning is using the first layers to find compact low-dimensional representations of high-dimensional data whereas later layers are responsible for achievement of the task given, e.g. regression or categorical classification. All the neurons of the layers are activated through weighted connections. In order the network being capable to approximate a nonlinear transformation, a non-linear activation function is applied to the neuron output. The learning is conducted by calculating error in the output layer and backpropagating gradients towards the input layer. In regular deep neural network layer, each neuron in a hidden or output layer is fully connected to all neurons of the previous layer with the output being calculated by applying the activation function to the weighted sum of the previous layer outputs. Such layers have few trainable parameters and therefore learn fast compared to more complicated architectures.

To evaluate deep learning model capabilities to detect intrusions we use network packet captures from CICIDS2018 [3] dataset. It contains 560 Gb of traffic generated during 10 days by 470 machines. The dataset in addition to benign samples includes following attacks: infiltration of the network from inside, HTTP denial of service, web, SSH and FTP brute force attacks, attacks based on known vulnerabilities. We concentrate on the intrusion detection based on the analysis of network traffic flows. A flow is a group of IP packets with some common properties passing a monitoring point in a specified time interval: IP address and port of the source and IP address and port of the destination. Resulting flow measurements provide us an aggregated view of traffic information and drastically reduce the amount of data to be analyzed. After that, two flows such as the source socket of one of these flows is equal to the destination socket of another flow and vice versa are found and combined together. This combination is considered as one conversation between a client and the server. A conversation can be characterized by following four parameters: source IP address, source port, destination IP address and destination port.

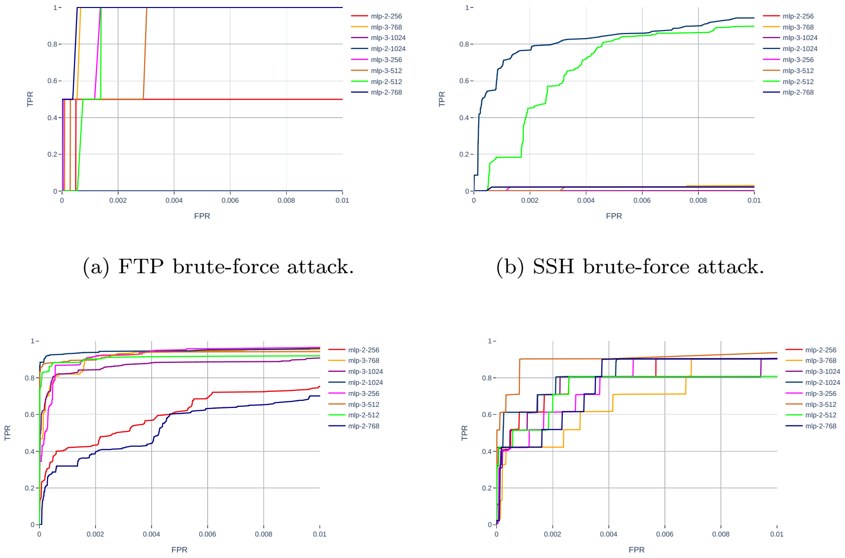
For each such conversation, at each time window, or when a new packet arrives, we extract the most essential features including flow duration, total number of packets in forward and backward direction, total size of the packets in forward direction, minimum, mean, maximum, and standard deviation of packet size in forward and backward direction and overall in the flow, number of packets and bytes per second, minimum, mean, maximum and standard deviation of packet inter-arrival time in forward and backward direction and overall in the flow, total number of bytes in packet headers in forward and backward direction, number of packets per second in forward and backward direction, number of packets with different TCP flags, backward-to-forward number of bytes ratio, average number of packets and bytes transferred in bulk in the forward and backward direction, the average number of packets in a sub flow in the forward and backward direction, number of bytes sent in initial window in the forward and backward direction, minimum, mean, maximum and standard deviation of time the flow is active, minimum, mean, maximum and standard deviation of time the flow is idle [3]. All the features can have different scale and therefore they are supposed to be standardized.

In our numerical experiments, we process raw packet capture files. First, we extract necessary packet features, then combine separate packets into conversations and, after that, we extract conversation features. It is worth noticing, that every time a new packet is transferred during the conversation or a certain time period (one second in our case) passes, we recalculate the conversation features and add a new data sample for the updated conversation. The idea behind that is that we attempt to evaluate how well the deep learning methods can detect intrusions in real time not when the conversation is over. Some results are presented on Figure 1.

As one can notice from the figures, basic neural networks allow us to detect malicious connections without many false alarms. Results for the classification models slightly vary in terms of true and false positive rates depending on the architecture. It is also worth noticing that increasing the number of trainable parameters does not improve accuracy of the models significantly. In the sense of efficiency, simple MLPs look the most promising solution. It is worth noticing that we also experimented with more complicated neural network layers, e.g. residual [4] and self-attention [5], but for our classification task those do not provide any increase in the detection accuracy. We also tested unsupervised models such as autoencoders, however we did not manage to obtain good results using those.

3 Deep reinforcement learning

Reinforcement learning is a machine learning paradigm in which software agents and machines automatically determine the ideal behavior within a specific context by continually making value judgments to select good actions over



(a) FTP brute-force attack.

(b) SSH brute-force attack.

(c) Web brute-force attacks.

(d) Botnet attack.

Fig. 1: Dependence of TPR on FPR for different intrusion detection with deep learning models applied to flow features.

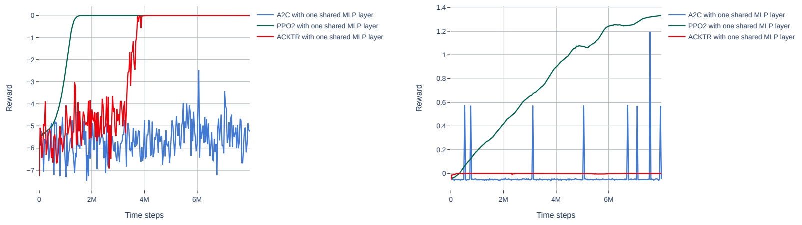
bad. A reinforcement learning problem can be modeled as Markov Decision Process (MDP) that includes three following components: a set of agent states and a set of its actions, a transition probability function which evaluates the probability of making a transition from an initial state to the next state taking a certain action, and an immediate reward function which represents the reward obtained by the agent for a particular state transition. If the transition probability function is known, the agent can compute the solution before executing any action in the environment. However, in real-world environment, the agent often knows neither how the environment will change in response to its action nor what immediate reward it will receive for executing the action. It is not enough to only account the immediate reward of the current state, the far-reaching rewards should also be taken into consideration. Most of the time RL algorithms focus on the optimization of infinite-horizon discounted model, implying that the rewards that come sooner are more probable to happen, since they are more predictable than the long term future reward.

There are three main approaches for the reinforcement learning: value-based, policy-based and model-based. In value-based RL, the goal is to maximize the value function which is essentially a function that evaluates the total amount of the reward an agent can expect to accumulate over the future, starting at a particular state. The agent then uses this function by picking an action at each step that is believed to maximize the value function. On the other hand, policy-based RL agent attempts to optimize the policy function directly without using the value function. The policy function in this case is the function that defines the action the agent selects at the given state. Finally, model-based approach focuses on sampling and learning the probabilistic model of the environment which is then used to determine the best actions the agent can take. Assuming the model of the environment has been properly learned, model-based algorithms are much more efficient than model-free ones, however, since the agent only learns the specific environment model, it becomes useless in a new environment and requires time to learn another model.

As a rule, a neural network is used to estimate an RL agent's policy with loss function being estimated based on the probability of the action taken multiplied by the cumulative reward obtained from the environment. Updating the policy network parameters by taking random samples may introduce high variability in probabilities and cumulative reward values, because trajectories during training can deviate from each other at great degrees. This results in unstable learning and the policy distribution skewing to a non-optimal direction. One way to reduce variance and increase stability is subtracting the value function from the cumulative reward. This allows one to estimate how much better the action taken is compared to the return of an average action. The value function can be estimated by constructing the second neural network, which estimates the environment's state value in the manner similar to DQN. The resulting architecture is called advantageous actor-critic (A2C), where the critic estimates the value function, while the actor updates the policy distribution in the direction suggested by the critic [8].

To improve stability of the learning even further, trust region policy optimization (TRPO) relies on minimizing a certain surrogate objective function that guarantees policy improvement with non-trivial step sizes [9]. TRPO uses average KL divergence between the old policy and updated policy as a measurement for a region around the current policy parameters within which they trust the model to be an adequate representation of the objective function, and then chooses the step to be the approximate minimizer of the model in this region. Although TRPO has achieved great and consistent high performance, the computation and implementation of it is extremely complicated. The current state-of-art algorithm policy optimization (PPO) attempts to reduce the complexity of TRPO implementation and computation by tracing the impact of the actions with a ratio between the probability of action under current policy divided by the probability of the action under previous policy and artificially clipping this value in order to avoid having too large policy update [11]. Another option to reduce the complexity closer to a first-order optimization is proposed in [10]. Actor-critic using Kronecker-Factored trust region (ACKTR) speeds up the optimization by reducing the complexity using the Kronecker-factored approximation.

To evaluate performance of different RL algorithms, we use OpenAI gym that has emerged recently as a standardization effort [6]. We run multiple copies of the environment in parallel. The training process is divided into episodes. Each episode lasts a certain fixed amount of time steps, during which one of the tasks is performed by an agent implemented using OpenAI baselines [7]. The tasks include swinging an inverted pendulum up from a random position and moving a two-dimensional car. We test three state-of-art RL algorithms A2C, ACKTR and PPO in those environments. We concentrate on these algorithms as they can be applied for both discrete and continuous environments. In our experiments, PPO consistently provides good results in terms of both average reward and convergence speed (see Figure 2). We run several experiments with different network architectures. The results on Figure 3 show that the network with one shared layer followed by two separate streams for policy and value function looks the most promising architecture variant. We also experimented with using shared LSTM layer for both policy and value function, but the results showed that much more steps is required for the algorithm convergence in this case, which can be critical in case of more complicated environment that requires more time per iteration.



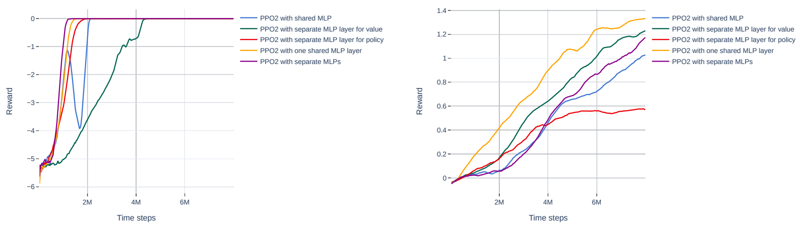
(a) Pendulum.

(b) Continuous mountain car.

Fig. 2: Performance of three different RL algorithms in two basic OpenAI gym environments.

4 Traffic generation

In order to train the reinforcement learning agents a simulation environment is supposed to be constructed since we cannot deploy, train and test those agents in a real production network. Traffic in such environment can be attempted to be generated with the help of conditional generative adversarial networks (GANs) [12]. In GANs, the discriminator generates an estimate of the probability that a given sample is real or generated. The discriminator is supplied with a set of samples which include both real and generated ones and it would generate an estimate for each of these inputs. The error between the discriminator output and the actual labels would then be measured by cross-entropy loss. GAN can be extended to a conditional model if both the



(a) Pendulum.

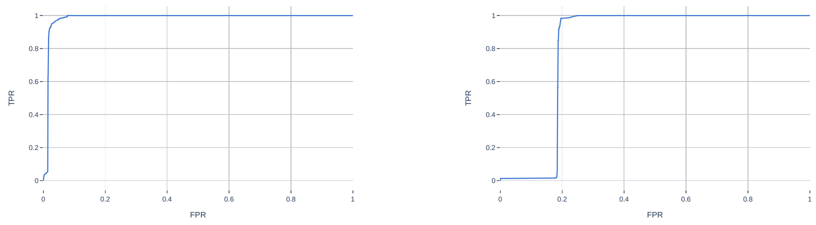
(b) Continuous mountain car.

Fig. 3: Performance of PPO2 with different policy and value network architectures in several OpenAI gym environments. Architectures: shared MLP (blue), one separate layer for value (green), one separate MLP layer for policy (red), one shared MLP layer (yellow), separate MLPs (purple)

generator and discriminator are conditioned on some extra information [13]. We can perform the conditioning by feeding this information into the both the discriminator and generator as additional input layer. In our case, this extra information includes several packets sent in a network traffic flow, and the GAN is trained to generate features of the next packet, i.e. its payload size, TCP window size, TCP flags, inter-arrival time, etc. However, cross-entropy loss fails in some cases and not point in the right direction in other cases. This may lead to mode collapse when the generator only learns a small subset of the possible realistic samples which the discriminator cannot recognize. One potential solution for this problem is using Wasserstein distance metric [14]. The Wasserstein metric looks at the distribution of each variable in the real and generated samples, and determines how far apart the distributions are for real and generated data. The Wasserstein metric looks at how much effort, in terms of mass times distance, it would take to push the generated distribution into the shape of the real distribution.

We use conditional Wasserstein GANs to generate inter-arrival time between two consecutive packets, payload size and TCP window size, the second generates n-gram distribution for the payload of the packet. Features for the condition include direction (request or reply) and TCP flags. In the generator network, a random noise vector is concatenated with the condition and the result is fed to an MLP, output of which is a feature vector for the next packet. The discriminator network also takes features extracted from the previous packets of the flow as an input. The second input is the feature vector generated by the generator. The generator produces packets that are closer to the real ones extracted from the datasets while the discriminator network tries to determine the differences between real and fake packets. The goal is to have a generative network which can produce traffic flows whose features resemble the ones extracted from the real flows.

We trained such GANs separately for different attacks presented in the dataset and the normal HTTP traffic. Figure 4 show the results of applying the classifiers trained in the previous stage to the traffic generated with GANs. As one can see, the results for models which are trained with flow features are more or less inline with the ones obtained using the real data.



(a) Web brute-force attacks.

(b) Botnet attack.

Fig. 4: Dependence of TPR on FPR for different intrusion detection with deep learning models applied to flow features extracted from fake traffic generated with Wasserstein GANs.

We implement an application for traffic generation in form of a Docker container. Docker allows users to package an application with all of its dependencies into a standardized unit for software development. Unlike virtual machines, containers do not have high overhead and hence enable more efficient usage of the underlying system and resources. The container includes client and server application implemented with Scapy module which is able to forge or decode packets of a wide number of protocols. We set the first ECN bit of each packet generated with the generator trained using malicious traffic to one in order to be able to provide ground truth labels for the AI in order to calculate the reward. Generator models of the trained GANs are first converted into a compressed .tflite format and added to the application. This has been done in order to deploy the trained model without installing the entire Tensorflow library.

5 Software-defined networking and network function virtualization

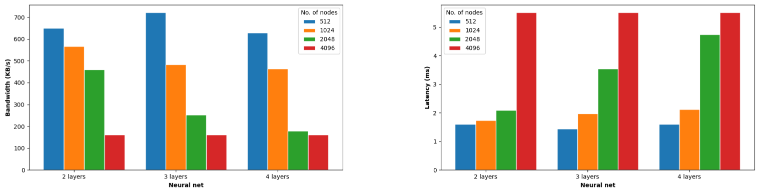
The main purpose of the SDN controller in our defense framework, which is to transform the security intent of the AI core to SDN flows and push them to the switches, can be implemented as an internal module of an existing SDN controller or an external application that uses RESTful APIs exposed by one or more plugins existing in the controller framework. There are many open-source controller options currently available, that can be modified in order to be used to redirect traffic between devices under protection and virtual security appliances. According to several SDN controller surveys, OpenDayLight [15] is one of the most featured controllers that are able to run on different platforms. Being under the partnership of well-known network providers and research communities, they have a clear development plan and good documentation. Even though Java-based OpenDayLight is inferior in performance compared to the controllers implemented in C in terms of throughput, they perform on similar level in terms of latency [16], which is alongside with high modularity and proper documentation makes it the most optimal choice to serve as an SDN controller in the defense system proposed.

Once all the necessary features have been installed on the Opendaylight controller, we implement a simple application for receiving information from operational data store of the controller and manipulating its data stores, which include pushing a flow into a switch table, finding existing tables on a switch, finding existing flows on a switch, deleting flow from a switch table and deleting entire table from a switch. These functions allow us to setup basic network configurations by resubmitting a packet with certain Ethernet protocol to another table, replying to an ARP request with a MAC address, redirecting an ARP packet with certain target protocol address to a port, redirecting an IP packet with certain destination to a port, outputting an IP packet with certain source to a port and resubmit to another table, modifying ECN of an IP packet with certain destination to a port and resubmit to another table. The purpose of the last action is to change the second ECN bit of a packet to one when it arrives at the first switch and change it back to zero when it sent from the last switch. The idea is to account for packets which are dropped in the environment in order to calculate the impact of the defense 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 attack detection and mitigation. We implement our own security middle box in order to use deep learning models trained. For that purpose, we first install OpenVSwitch on an Ubuntu virtual machine and connect it to our Opendaylight controller. It can then be connected to other network switches via VXLAN tunnels. For intercepting and analyzing network traffic we use Libnetfilter_queue [17] and Iptables firewall rules to gain access to network packets and the ability to reject or accept these packets for forwarding. Python library Netfilterqueue is used to interface with Libnetfilter_queue [18] from a python program. The interceptor program receives a network packet and extracts relevant features from it and uses pretrained classifier to determine whether it is malicious or not. The malicious flows are flagged by setting a desired bit in the TCP-protocol DSCP field (upper 6 bits of the TOS field) facilitating detection and further actions by downstream devices. All packets are then forwarded regardless of the analysis result.

Similarly to the traffic generation containers we use Tensorflow Lite interpreter in order to avoid installing the entire Tensorflow library. We select the best classifier in terms of the metric selected (e.g. accuracy or AUC) for each attack class tested and copy those models to the VNF. We finally implement a simple API using Flask which allows to manipulate two parameters: classifier model to use for the analysis and the threshold according to which we differentiate normal traffic from malicious one. This is basically a very straight forward implementation of transfer learning approach, i.e. a model developed for a task is reused as the starting point for a model on a second task. We train a model using traffic contained in the dataset and then use all its layers except the last one as a foundation for the classifier inside our VNF. The last layer is essentially one number since we only classify traffic as either normal or malicious. Thus, an intelligent agent can manipulate VNfs implemented by selecting the most optimal combination of the classifier and the value of the threshold.

The effect on network performance can be measured by setting up a three machine virtual network: two virtual machines in separate subnets and a virtual machine running the interceptor program acting as a router between the subnets. The software tool Qperf [19] is used to analyze network performance. A Qperf server is started on one of the test machines and a client running a test against the server was started on the other. All traffic passed through the interceptor machine and through the analysis. The recorded metrics are bandwidth and latency of TCP traffic. The test is repeated several times with the interceptor analyzing packets with several different classifiers. The tested configurations have 2,3 or 4 layers of 512, 1024, 2048 or 4096 nodes. The results are shown in figure 5. These results accompanied with the ROC curves obtained previously allow us to conclude that it is reasonable to use MLP classifiers with less trainable parameters, since increasing the number of trainable parameters does not improve accuracy of the models significantly, however it negatively affects the network performance.



(a) Bandwidth through the interceptor. (b) Latency through the interceptor.

Fig. 5: Network performance of the interceptor with varying number of layers and nodes in each layer.

6 Prototype environment

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. In order to train an intelligent agent in a reasonable amount of time, the training process can be carried out in several environments in parallel. For this reason, we build our training environment as a network of several virtual machines using Vagrant. Vagrant [20] is an open-source program which allows for automatic building and managing virtual machines. Vagrant uses existing hypervisors, in our case Qemu/KVM through Libvirt, to deploy and run the machines. Vagrant manages these machines through SSH connections and can provide access to files for the virtual machines through NFS shares. We use Vagrant to configure the VMs required for the system implementation which include SDN controller, several VMs with Docker containers, several VMs with Tensorflow network traffic flow classifiers, and one VM for traffic monitoring. All VMs except for the controller have Openflow-enabled OpenVSwitch preinstalled. Switches are connected between each other with VXLAN tunnels. It is worth noticing that switch of the traffic monitor is not controlled by Opendaylight, it simply acts as a "sink" for the network traffic in order to provide information on the network state for an RL agent. We use OpenAI gym [6] to implement the frontend for the virtualized environment. The RL agent is implemented using OpenAI baselines [7]. The resulting environment is shown in Figure 6.

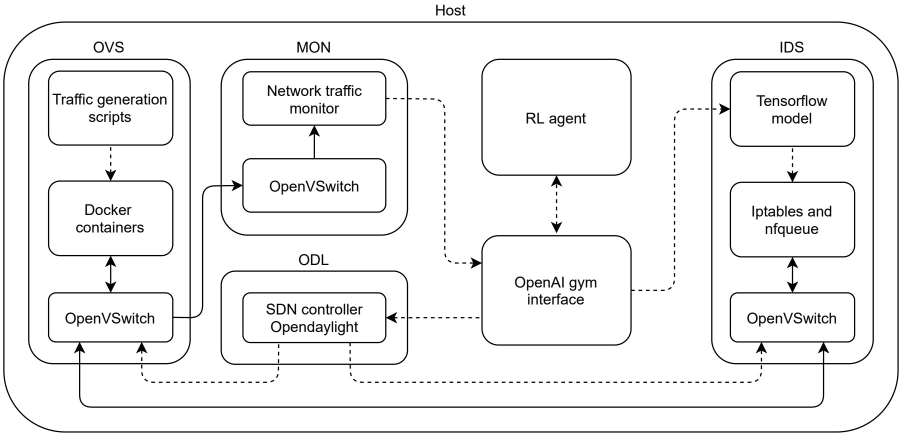
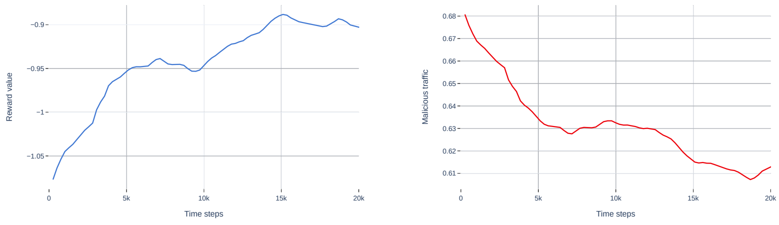


Fig. 6: The environment for training RL agent implemented using Vagrant. Network traffic flows and commands are shown by solid and dashed lines respectively.

The RL agent observes packet and byte counts sent from one host to another for each pair of hosts in the environment, one-hot encoded indexes of the classifiers deployed in the security boxes, threshold values used in the classifiers. Action set of the RL agent consists of changing the classifier model index for a certain VNF, changing the threshold for a certain VNF, redirecting traffic between a certain pair of subnets having certain DSCP label to a certain middle box, and blocking traffic between a certain pair of subnets having certain DSCP label. For the reward function calculation, we utilize our flow monitor VM. Since all malicious traffic packets generated have the first ECN bit equal to 1 and all packet which are received on the first switch have the second ECN bit equal to one, we can calculate percentages of the normal and malicious traffic which are blocked by the environment. The reward function is then calculated as a sum of these two components. We initialize flow tables of each SDN switch with basic flows to forward each packet to its destination. SDN flows to drop packets or redirect them to a particular security appliance are pushed to the dedicated flow tables with priority higher than default forwarding rules.

In order to evaluate the framework proposed, we consider the following attack scenario. Eight devices are connected to the internal network. These devices can be accessed via SSH and HTTP by both internal and external hosts. To generate malicious traffic, we generate three types of the attacks: SSH password brute-force, web application password brute-force and communication between C&C and one of the devices which is considered infected. The training process is divided into episodes. Each episode lasts for one minute, during which both benign and malicious traffic flows are generated. The RL agent is implemented using OpenAI baselines. The agent selects one of the actions for one of the flows that are sent to the environment back-end where they are transformed to SDN rules. We train the RL agent using PPO algorithm with multi-layer perceptron (MLP) as both policy and value function to detect and mitigate the attacks mentioned. Figure 7 shows the evolution of the reward function throughout few training episodes in the attack scenario mentioned. As one can notice, the agent starts to identify and block malicious connections reducing the number of malicious flows and subsequently increasing the reward value.



(a) Reward.

(b) Malicious traffic.

Fig. 7: Reward function during the training and percentage of the malicious traffic received by the attack targets.

7 Conclusion and future work

The main contribution of this research is developing a proof-of-concept of an intelligent network defense system which relies on SDN and NFV technologies and allows customers to detect and mitigate attacks performed against their smart devices by letting an artificial intelligent agent control network security policy. On infrastructure level, the defense framework proposed includes cloud compute servers in order to emulate elements of real infrastructure as well as launch security appliances. In order to forward traffic from the network under protection to these appliances as well as connect the appliances to each other, the system relies on SDN capabilities that include global visibility of the network state and run-time manipulation of traffic forwarding rules. The key component of the defense system proposed is a reinforcement learning agent that resides on top of the SDN and NFV controllers and is responsible for manipulating security policies depending on the current network state. In particular, the agents processes traffic flowing through edge switches as well as log reports from security appliances deployed and manipulates the network traffic by instructing the SDN controller to pass, forward or block certain connections. We used the resulting prototype to evaluate two state-of-art reinforcement learning algorithms for mitigating three basic network attacks against a small virtual network environment.

There are however still numerous issues which have to be addressed. Those are mostly related to the traffic generation procedure. We managed to implement simple traffic generation application, it however does not really represent the realistic traffic and therefore its usage is limited in a real world scenario. A potential solution would be to use real devices and applications and generate traffic using those. In the future, we are planning to continue this research by conducting experiments in the environment prototype, as well as testing various reinforcement learning algorithms for different attack scenarios. We are also aiming to improve the scalability of the framework proposed and evaluate the system performance for bigger network environments. We are also going to implement adversarial module for the traffic generators which would allow for spoofing a neural-network-based intrusion detection system by manipulating flow parameters. Finally, we are going to test the working prototype of the network defense system developed during the project in a non-SDN enterprise network environment.

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. J. Kindervag. No More Chewy Centers : Introducing The Zero Trust Model Of Information Security. pp. 1–15, 2010.
3. I. Sharafaldin, A. Lashkari, and A. Ghorbani. Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization. Proc of the 4-th International Conference on Information Systems Security and Privacy (ICISSP), pp. 108–116, 2018.

4. K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. Proc. of the IEEE conference on computer vision and pattern recognition, pp. 770–778, 2016.
5. A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. Proc. of the 31st International Conference on Neural Information Processing Systems (NIPS'17). pp. 6000–6010, 2017.
6. G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang and Wojciech Zaremba. OpenAI Gym.arXiv:1606.01540, 2016.
7. 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.
8. V. Mnih, A. Badia, M. Mirza, A. Graves, T.Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous Methods for Deep Reinforcement Learning. arXiv preprint arXiv:1602.01783, 2016.
9. J. Schulman, S. Levine, P. Moritz, M. Jordan, and P. Abbeel. Trust Region Policy Optimization. arXiv preprint arXiv:1502.05477, 2015.
10. Y. Wu, E. Mansimov, S. Liao, R. Grosse and J. Ba. Scalable trust-region method for deep reinforcement learning using Kronecker-factored approximation. arXiv preprint arXiv:1708.05144, 2017.
11. J. Schulman, F. Wolski, P. Dhariwal, A. Radford, O. Klimov. Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347, 2017.
12. I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. Advances in neural information processing systems, pp. 2672–2680, 2014.
13. M. Mirza and S. Osindero. Conditional Generative Adversarial Nets. arXiv preprint arXiv:1411.1784, 2014.
14. M. Arjovsky, S. Chintala and L. Bottou. Proc. of the 34th International Conference on Machine Learning, Vol. 70, pp 214–223, 2017.
15. J. Medved, R. Varga, A. Tkacik and K. Gray. OpenDaylight: Towards a Model-Driven SDN Controller architecture. Proc. of IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks, pp. 1–6, 2014.
16. O. Salman, I. H. Elhajj, A. I. Kayssi, and A. Chehab. SDN controllers: A comparative study. Proc. of the 18th Mediterranean Electrotechnical Conference (MELECON), pp. 1–6, 2016.
17. The netfilter.org "libnetfilter_queue" project. Retrieved December 1, 2020, from https://www.netfilter.org/projects/libnetfilter_queue/index.html.
18. NetfilterQueue - PyPi. Retrieved December 1, 2020, from https://pypi.org/project/NetfilterQueue/.
19. Github - linux-rdma/qperf. Retrieved December 1, 2020, from https://github.com/linux-rdma/qperf.
20. Vagrant by HashiCorp. Retrieved November 19, 2020, from https://www.vagrantup.com.