Defeat MITM Attack with Reinforcement Learning

Yang Hu yh2948
Xiaotian Hu xh2332

Abstract

Although 30 years have passed since man-in-the-middle attack first introduced, it remains a critical threat to modern network security. Instead of putting effort to optimize the protocol, we use reinforcement learning algorithm to bypass the potentially intruded nodes. Our model solves two major problems: one is the detection of intrusion could be misleading and the other is the agent doesn't know when and where the decker will intrude. To solve these problems, we first introduce an intrusion likelihood ratio to increase the detection accuracy. Then we build a model with neural Q-learning to minimize the threat of Decker by keeping track of the network's global information. As a result, the agent can change forwarding path immediately as long as one node on the forwarding path gets affected. In the experiment, we implemented the scheme in a 13 nodes network. The packets could be successfully delivered to the server without hijacking after 4 hours training.

1. Introduction

Man-in-the-middle (MITM) attack is a kind of attack where an adversarial computer between two computers pretending to one to be the other (Desmedt, 2011). There are many methods work against the MITM attack, among which Secure Socket Layer(SSL) (Heinrich, 2011) and Transport Layer Security(TLS) are the most effective ones. SSL/TLS is adopted as a predominant communication security protocol for the Internet and denoted as https together with http protocol. Generally, SSL/TLS is robust enough to deal with active eavesdropping. But MITM attack remains for two reasons. One is many websites fail to provide https service. According to Google's transparency report GoogleReport (Google) ("HTTPS encryption on the web"), 33 websites of top 100 non-Google sites (provide approximately 25% of all website traffic) have trouble providing an https website. And these websites include big names like Alibaba, IMDB, and Sina. The other is some news suggest certification authority(CA) may get corrupted and release fake certificates.

In this paper, we propose a reinforcement-learning based approach to detect and defeat man-in-the-middle attack. Instead of closing the loophole at client and server, we put efforts in detecting man-in-the-middle attack in a given network and try to bypass the nodes infected by the decker. So, our solution generally has two folds: first, use supervised learning to analyze the TLS traffic obtained from different resources, and try to identify whether the node has been hijacked by analyzing the characteristics of TLS traffic. Second, we use reinforcement learning to provide an optimal path for the routers alongside. Since the likelihood of a node has been hacked would change over time, the Neural Q-learning is a perfect fit to solve this kind of problem. After generating the routing policy, we can use Software Defined Network (SDN) to embed our routing protocol to the routers.

2. Related Work

2.1. MITM Attack Detection System

To successfully bypass the infected routers while forwarding the packet, a proper MITM attack detection system is crucial. Generally, the detection system has two prongs: probes and analysis engine.

- **Probe** As its name would suggest, a probe captures packets in the network and extract certain information like the packets' source and destination IP addresses. Normally, these probes are network cards in promiscuous mode scattered on the switches/routers(hosts), like NetFlow(Hofstede et al., 2014). However, the probs may have a severe impact on the performance of conventional host, so only a small number of hosts will install the probe. But the probes should also be sufficent in number and properly distributed to successfully monitor all nodes in the network.

- **Analysis Engine** In this context, analysis engine aims to discern intrusion nodes after aggregating data from all distributed probes. According to (Aziz & Hamilton, 2009), the flow duration will increase if a traffic encounters a MITM attack. And considering the decker has to connect to the LAN before the attack, some features, like protocol type, service, src bytes,

destination bytes, flag, etc, in intrusion traffic also suits MITM attack traffic. (Mukkamala et al., 2002) Basing on these features, we can manually label a training set with 1,0 to indicate the node has been intruded or not. After training a classification model with the labeled training set, the analysis engine will be able to discern each node's status by fitting the upcoming traffic into the classification model. However, the results of classification model could be misleading sometime, and we will use reinforcement learning method to deal with it.

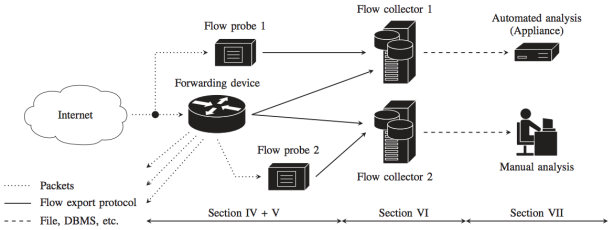

Figure 1. MITM Attack Detection System

2.2. Make Model Practical with SDN/OpenFlow

As mentioned above, we are using the reinforcement-learning method to give an optimal path which is able to bypass the infected node. Therefore, finding all the possible forwarding paths and switching between them whenever necessary is the premise of the method. Since MITM attack usually happens inside a LAN, the routing can narrow down to the intradomain routing protocols including Routing Information Protocol(RIP).

But in order to update the routing table as desired, it is necessary to use Software Defined Network(SDN), an architecture composes of controllers and three-layer switches who usually supports OpenFlow protocol. The controller will process the demands of user-defined applications into the corresponding behavior of OpenFlow switches.

Under the framework of reinforcement learning, the controller will work as an agent, collecting the data from probes located at OpenFlow switches, running a network flow classification process (Analysis Engine mentioned above) to choose the optimal path, which gives the highest performance of bypassing the decker's dynamic attack. The agent training time may scale with the increase of network scope. But additional training is no longer required after the first train as long as the number of paths between start and end hosts remains the same.

3. Approach

3.1. Settings

For a target client-server pair, the decker will randomly take over an arbitrary number of routers on the forwarding path. As mentioned in 2.1, the MITM attack detection system will label each node as intruded(1) or not(0) during the analyzation of flows' features. Normally, we expect the controller always switches to a safe forwarding path according to the output of MITM attack detection system. There are several challenges though.

The first challenge comes from the MITM attack detection system. Since the training set can be limited and bias, the result of detection system isn't perfect. There are two kinds of detection error: one is the false positive error which regards normal router as attacked, and the other is the false negative error which views the intruded one as normal.

The second challenge is how to notify the client. Since the client has no knowledge of network's global condition and can't be controlled by the controller, it will keep sending packets if there's no signal. This will lead to the leak of information.

The third challenge is exploration and exploitation. The MITM attack detection system can only return the nodes' status within the traffic path. However, the decker could dynamically change its victims, which requires the controller could switch to a safe path swiftly. But this immediate action should base on the information of nodes on other paths, which ask the controller do some exploration time to time.

For the first challenge, we introduce a parameter $lr(t)$ to evaluate a node's intrusion likelihood ratio basing on the accumulative detection results.

$$lr(t) = \prod_{i=t-t_0}^{t} l(i) \tag{1}$$

$$l(i) = \begin{cases} \frac{ACC}{FOR}, & d(i) = 1 \\ \frac{1-ACC}{1-FOR}, & d(i) = 0 \end{cases} \tag{2}$$

where ACC is the accuracy of the system, FOR is the false omission rate of the system, d(i) is the detection result at time i, and the t_0 is the time span used to evaluate the likelihood ratio. The likelihood ratio indicates that possibility that node is intruded. Detailed derivation can be found in the supplementary material.

For the second challenge, the controller could signal the client by dropping packets on switches. When the packet loss rate reaches a certain threshold, the client will automatically adopt congestion control algorithm, like Robust random early detection, to prevent leaking more information. We will give further discussion to the third challenge in the rest of the paper.

3.2. Neural Q-learning

To deal with the third challenge mentioned, we adopt Neural Q-learning, an algorithm in reinforcement learning. Reinforcement learning (Sutton & Barto, 1998) aims to guide the agent interacts with the unknown environment, which always formalized as a Markov Decision Process (MDP) and described by a 4-tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R})$.

In our scenario, for a network with N nodes and given client-server pairs with M optional paths, the state can be represented as

$$S_t = [lr_1, lr_2, lr_3, \ldots, lr_N, P_t] \tag{3}$$

lr_1 to lr_N is the intrusion likelihood for each node in the network. P_t is the index of current path.

For each state, the agent has $2M$ of actions $a \in \mathcal{A}$ to choose from. $\mathcal{A}$ includes dropping the package on path 1 to M, or forwarding the packet on path 1 to M.

For each state action pair, the agent will receive a reward $r_t \sim \mathcal{R}\{s_t, a_t\}$ base on the output of the MITM attack detection system. At time t, if the controller is forwarding the packets on a path whoes nodes' status are all 0, which means the path is safe, r_t will be 4. But if any of them is 1, then r_t is -5. If controller decide to drop packets, r_t will always be -1.

Q-learning helps evaluates the value of controller's from a given network state. And this value is known as state action value or Q-value. Q-value is updated iteratively by updating the guessed current Q-value towards the observed Q-value:

$$Q(s,a) := Q(s,a) + \alpha \left(r + \gamma \max_a Q(s', a') - Q(s,a) \right) \tag{4}$$

Since the state is continuous and infinite, we use Neural Q-learning(NQL)(Mnih et al., 2015) to approximate the Q-value. In NQL, the Q-value becomes a neural network parameterized by weights and biases collectively denoted as θ. Loss function is uesed to minimize the gap between the real Q-value and estimated Q-value.

$$\mathcal{L}(s, a|\theta_i) \approx \left(r + \max_a Q(s', a|\theta_i) - Q(s, a|\theta_i) \right)^2 \tag{5}$$

And θ gets updated for each step.

$$\theta_{i+1} = \theta_i + \alpha \nabla_\theta \mathcal{L}(\theta_i) \tag{6}$$

And in order to ensure the learning stability, we also adopt *experience replay*, which first store the experience $e_t = (s_t, a_t, r_t, r_t + 1)$ in replay memory $\mathcal{D}$ and sampled uniformly at training time. Following is the loss function we use for the algorithm:

$$\mathcal{L}_i(\theta_i) = E_{s,a,r,s' \sim D} \left[\left(y_i^{NQL} - Q(s, a; \theta_i) \right)^2 \right] \tag{7}$$

$$y_i^{NQL} = r + \gamma \max_a Q\left(s', a'; \theta_i^-\right) \tag{8}$$

Detailed description of learning process can be found in supplementary material.

4. Experiments and Results

4.1. Environment Setting

To test our algorithm, we create a network with 13 nodes. Its topology is illustrated in figure 2

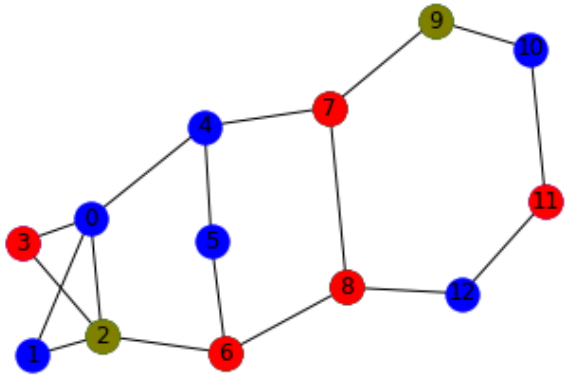

Figure 2. Network topology. Yellow nodes: client-server pair Red nodes: hacked router Green Nodes: regular switches

In the experiment, we realize a simple routing protocol to find all the possible paths for a given client-server pair. And the path will be removed if a loop could form with one additional edge added. Take figure 3 as an example, we discard this path because there will be a loop if we connect 7 and 9. Figure 4 is a path satisfy all the requirements.

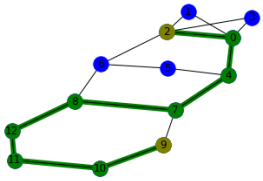

Figure 3. One path with loop

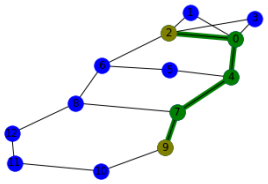

Figure 4. Standard path

By testing our routing protocol with arbitrary network topology and scale, we found the average possible paths between a client-server pair is 4. So we choose 4 as the number of routing paths to switch.

To get the optimal policy, the agent should experience all the attack pattern enough times in the training phase. In our experiment, we assume the number of intrusion nodes is

less than 5, which is reasonable. Because if 5 or more nodes are intruded in our network, the safe path hardly exists. And if all possible routes are cut down, maybe the safest policy is stopping communication. Besides, we maintain a list of intruded nodes which gets updated every 50 frames. As the features of the controller, ACC (detection accuracy) and FOR (false omission rate) are set to be 0.8 and 0.05 respectively.

4.2. Training

The model strictly follows the general NQL algorithm: for each training batch, a fixed number of experience frames are uniformly fetched and used to train the feedforward network. Those experiences are generated by exploration in ϵ-greedy policy.

In the experiment, we found that with a relatively large learning rate, the model is likely to converge into local optimal value, where the network will give constant output regardless of the value of the input.

To avoid converging into local optimal without hurting the training speed, we trained the model in a two-stage scheme. At first stage, we assign the model a large learning rate and check whether the output parameters could be constant or not. If they are, we go into second stage training with a smaller learning rate to refine the network. If the neural network unable to converge at first stage, go back to the first stage.

The learning rates for the first and second stage are 0.001 and 0.00001 respectively. In each stage, the agent runs 60000 steps. 5 and 6 illustrate the loss curve for stage one and two.

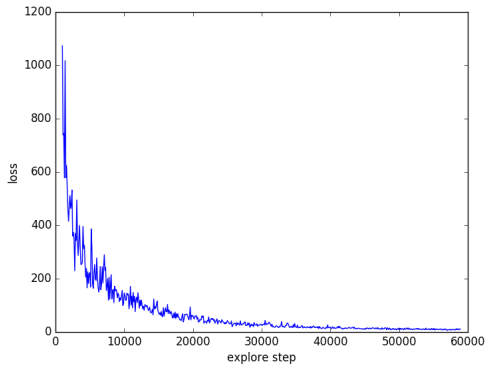

Figure 5. The loss curve of the first training phase. The loss fall drasticly within the first 5000 steps. To present the curve in detail, the loss data of the first 1000 steps is omitted

As these figures 5, 6 show, the loss falls rapidly at the beginning of training and then change slightly. The average reward of the exploration agent was shown in the figure 7 and 8

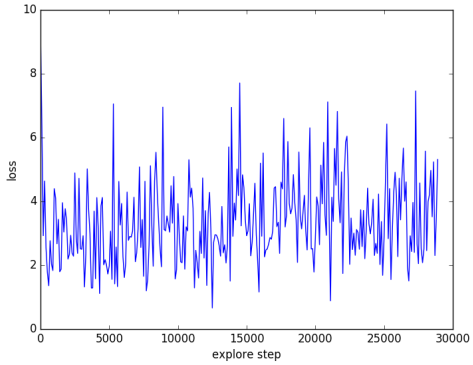

Figure 6. The loss curve of the first training phase. The loss oscillate within the range of length 4.

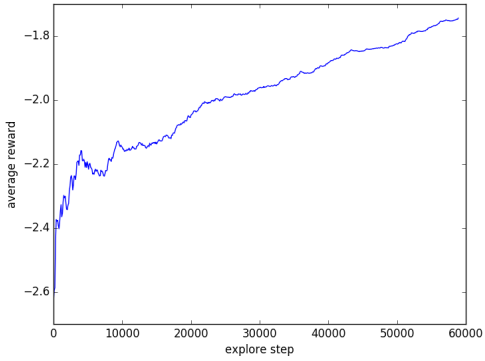

Figure 7. The average reward curve in the first training phase. The average reward increase significantly.

4.3. Experiment Results

To test the agent performance in different attack patterns, we set 4 attack configurations from combination of the number of attacked nodes: 3 or 5 and attack change frequency: 50 or 100. The agent ran 3000 steps with greedy policy for each configuration.

The agent presented the desired behaviors. It avoids attacked nodes and find the safe path, and keep dropping packages when all paths were under attack. These are depicted in figure 9, 10

Moreover, the agent displays the ability to ignore the falsely positive detection and falsely negative detection.

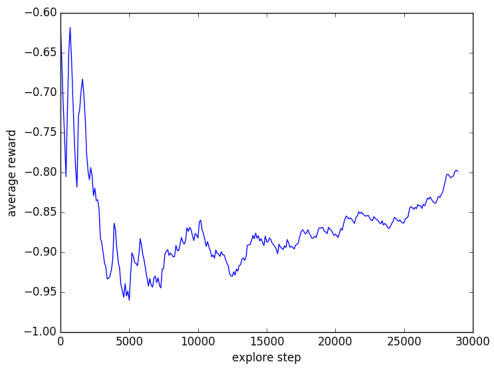

Figure 8. The average reward curve in the second training phase.

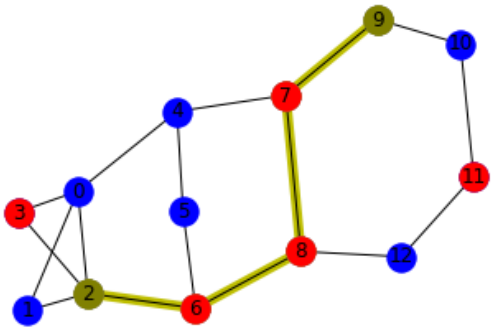

Figure 9. The agent successfully find the safe path and send the packages

The falsely negative detection that ignores the intrusion did not confuse the agent from releasing the risk of the path, and the falsely positive detection that mistakes the normal nodes as intruded nods did not scare the agent away from the safe path after the connection was established.

The average rewards of agent in different attack configuration are plotted in the figure 11, 12, 13, and 14

4.4. Analysis

Obviously, there are some difference in performance of different configuration, which matches the intuition.

On the one hand, the number of attack nodes negatively influences the performance. The more routers are intruded, the less possible the safe path still exist. When the attack pattern that blocks all the optional path appears, the agent will keep dropping packages with reward -1. Hence, agent achieved better average reward in the 3 attacked nodes environment than the 5 attacked nodes environment.

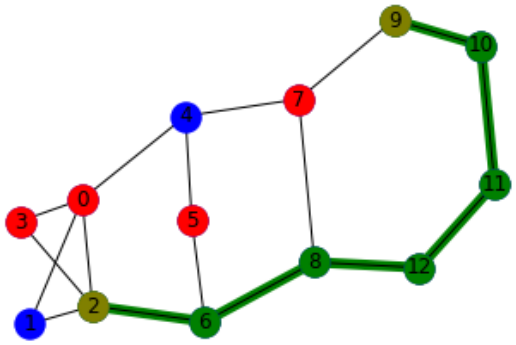

Figure 10. When all optional paths are blocked, the agent keeps testing optional paths and drop packages to signal the client

On the other hand, high frequency of attack pattern change impairs the performance. Because agent needs several steps to find the safe routes and ensure it is trustworthy by keep testing, and the high frequency of change may force the agent to abandon the path immediately after it found the path, which leads to a high package loss rate.

In addition to that, the error of the probe also influences the behavior of agent at the beginning of testing an optional path. Consecutive detection mistake might confuse the agent and make it send the packages in the intruded path or leave the safe path in a short time, which causes extra negative rewards.

5. Conclusion

In our project, we proposed a mechanism to defend man-in-middle-attack in the wire link. Our mechanism takes advantage of the flow probe and decides whether drop the sent packages or not based on the intrusion detection from the probe. To compensate the limitation of flow probe, the reinforcement learning is used to drive the policy on packages sending and paths selecting. Since the state space of our MDP model is continuous, we introduce the neural network to approximate the Q value for each state-action pair. We did series of experiments on a practical network. The result shows that our algorithm can avoid the intruded nodes in the network and choose the safe path to send packages. And our algorithm can also react to the change of the intruded nodes and, hence, is able to be applied to dynamic attacking scenarios.

Although out algorithm succeeded in avoiding intrusion in the most cases, it presents some delay when reacting the intrusion changes and low-efficiency in paths selection. This limitation can be improved in the future work. under the current model, the agent tests the optional path in an almost

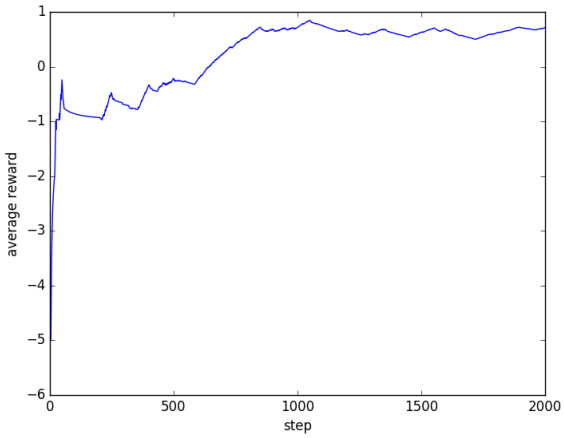

Figure 11. The average reward when 3 nodes are instruded each time and attack pattern change each 50 frames

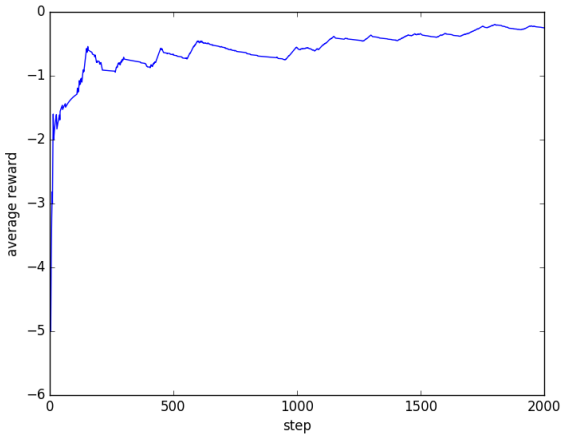

Figure 12. The average reward when 5 nodes are instruded each time and attack pattern change each 50 frames

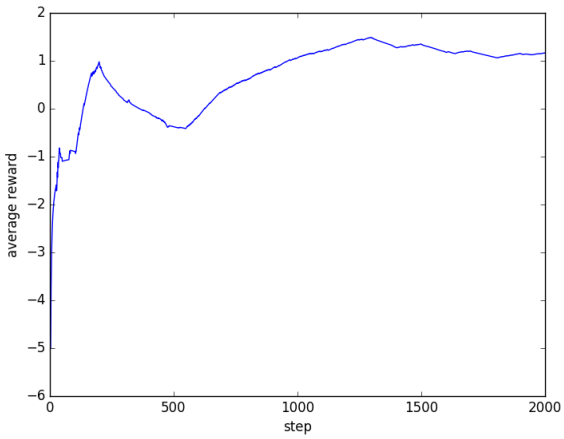

Figure 13. The average reward when 3 nodes are instruded each time and attack pattern change each 100 frames

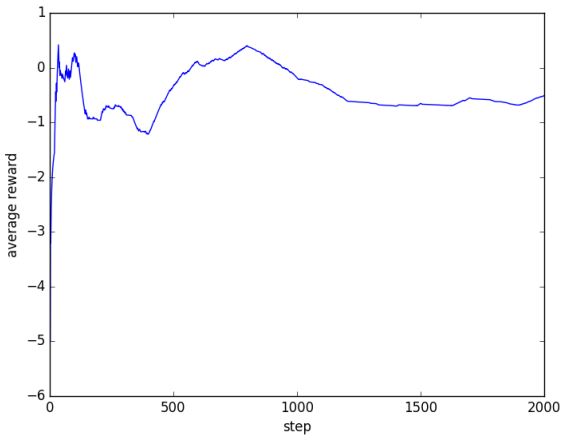

Figure 14. The average reward when 5 nodes are instruded each time and attack pattern change each 50 frames

random fashion, which brings unnecessary testing costs. It can be improved by logging the last renew time of likelihood ratio for each node and add these logs into the state vector. We tried this idea, but the network ended up with being trapped by constant output local optima. In the further work, we can tried different neural network topology and try different type of neural network to integrate the time log into the state vector.

6. contribution

Xiaotian Hu: realize the model, build model, write report
Yang Hu: provide idea, build model, write report
Github link: https://github.com/DoubleBiao/RL_MIMA

References

Aziz, Benjamin and Hamilton, Geoff. Detecting man-in-the-middle attacks by precise timing. In *Emerging Security Information, Systems and Technologies, 2009. SECURWARE'09. Third International Conference on*, pp. 81–86. Ieee, 2009.

Desmedt, Yvo. *Man-in-the-Middle Attack*, pp. 759–759. Springer US, Boston, MA, 2011. ISBN 978-1-4419-5906-5. doi: 10.1007/978-1-4419-5906-5_324. URL https://doi.org/10.1007/978-1-4419-5906-5_324.

Google. Https encryption on the web.

Heinrich, Clemens. *Secure Socket Layer (SSL)*, pp. 1135–1139. Springer US, Boston, MA, 2011. ISBN 978-1-

4419-5906-5. doi: 10.1007/978-1-4419-5906-5_223. URL https://doi.org/10.1007/978-1-4419-5906-5_223.

Mnih, Volodymyr, Kavukcuoglu, Koray, Silver, David, Rusu, Andrei A, Veness, Joel, Bellemare, Marc G, Graves, Alex, Riedmiller, Martin, Fidjeland, Andreas K, Ostrovski, Georg, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

Mukkamala, Srinivas, Janoski, Guadalupe, and Sung, Andrew. Intrusion detection using neural networks and support vector machines. In *Neural Networks, 2002. IJCNN'02. Proceedings of the 2002 International Joint Conference on*, volume 2, pp. 1702–1707. IEEE, 2002.

Sutton, Richard S and Barto, Andrew G. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.