Exploring aspects of network security using adversarial multi-agent reinforcement learning.

Panagiotis Lymperopoulos

December 12, 2019

1 Abstract

Securing a computer system, particularly a network of computers, is a process with a number of inherent trade-offs. High accessibility and availability often bring security vulnerabilities but at the same time perfect security is not feasible for any practically useful system. Here, we explore security trade-offs that arise in computer networks by simulating an abstract two-player game between a network attacker and a defender. The attacker tries to gain access to sensitive information by exploiting vulnerabilities in the network while the defender attempts to thwart the attacker by investing resources in different aspects of the network's security. The two agents learn to play the game and continually adapt their strategy using reinforcement learning algorithms.

2 Introduction

This work presents the development and results of a multi-agent reinforcement learning simulation that aims to explore trade-offs in network security. The simulation consists of two players, an attacker and a defender, acting adversarially on a static graph. The defender is tasked with securing one specific node of the graph that represents a computer in the network that stores sensitive information. The attacker aims to gain access to that node by starting out from its own computer and making its way through the network by attacking computers connected to its current node. Each player partially perceives the environment and acts based on that perception. Through this simulation I explore the behavior of the two agents in different scenarios, which I attempt to map to real problems and trade-offs in cybersecurity.

3 To the Community

As computer systems get larger and more complicated, it is becoming intractable to manually maintain security on all levels of the system. For this reason, the community should explore ways of intelligently automating not only low-level tasks such as intrusion detection, but higher-level security management as well. This work explores the application of reinforcement learning for network security and should serve as a reminder that the fundamental problems of security remain, regardless of whether security decisions are made by humans or by computers.

4 Related Work

The recent rapid increase of interest in machine learning and deep learning have brought interest in applying these tools in cyber-security. Xin et al. outline a number of such applications, with examples including malware analysis and intrusion detection [5]. While Xin et al. focus mostly on the supervised and

unsupervised paradigms of machine learning, reinforcement learning [4], with its focus on open environments and agent-based modelling, provides a natural framework for cyber-security applications.

Reinforcement learning problems. in their simplest form, are formulated as Markov Decision Processes, which are defined by a set of states that an agent can be in, a set of actions that can be taken from each state, along with a transition function that maps state-action pairs to next states, and a set of rewards received by an agent when entering a state. The aim of a reinforcement learning algorithm is to find a policy, a mapping between states to actions, that maximizes the long term reward of an agent in the MDP. This framework can be extended to a multi-agent scenario where multiple agents operate within the same environment and may have aligned or unaligned goals [1]. In the context of cyber-security, Nguyen et al. identify a number of ways in which deep reinforcement learning (a sub-field of RL that uses deep neural networks for function approximation) can been applied to traditional cyber-security problems [3]. Finally, Elderman et al. design a cyber-security game and apply multi-agent reinforcement learning on it [2]. The cyber-security game presented in the present work is an adaptation and further exploration of Elderman et al's game.

5 The Game

The game we explore in this work takes place is an undirected graph. Each node in the graph is associated with V vulnerabilities, a security value $v_i \in \{0, 1, 2\}$ for each one and a base detection probability $p = 0.1$. One node in the graph is the attacker's initial position and is unsecured ($v_i = 0$) for all V vulnerabilities. All other nodes have different levels of security for each vulnerability, with the requirement that at least one vulnerability is unsecured in each node. A second node in the graph is the target node, the one the attacker aims to reach. At each timestep, the attacker observes the security levels of the nodes adjacent to its current position and selects a node, and selects an attack. Attacks have a target node and vulnerability and are characterized by an intensity $i \in \{1, 2\}$. If the intensity of the attack is higher than the security value of the target node vulnerability, then the attacker moves to that node. A successful attack with intensity i has a probability $p * i$ to trigger a detection, which results in the attacker losing the game and receiving a reward of -100. If the attacker moves to the target node without being detected, it receives a reward of +100 and wins the game. At every timestep, after the attacker acts, the defender also selects an action. The defender's action consists of a node and a vulnerability whose security is modified by the formula: $v_i = max(2, v_i + 1)$. If all vulnerabilities in the network (except for the attacker's node) reach a security value of 2 or if the attacker is detected, the defender receives a reward of +100 and wins the game. If the attacker reaches the target node, the defender receives a reward of -100 and loses the game. Additionally, to encourage the attacker to quickly find a path to the target node, the attacker receives a reward of -1 for everuy

timestep in which its position is not the target node.

Although this game abstracts away most technical details of network security, its individual components are meant to map to some real-world concepts. The graph represents a computer network, with nodes being individual computers and edges being connections between computers. While communication in real networks is handled by different protocols at different layers, in the game two computers are connected permanently and no new connections are established. The set of vulnerabilities associated with each node in the graph are meant to represent the various ways computers can communicate with each other, along with the inherent security risks involved. For instance, a security level of 0 for vulnerability v_0 may represent a computer accepting ftp connections with default credentials, while the field with value 2 may correspond to sftp connections using encryption and secure credentials. The intensity of attacks also has a corresponding concept in cyber security. Attackers attempting to crack passwords may use a small list of 100 common passwords (intensity 1), or continually run brute force for three days on a gpu cluster (intensity 2). The detection probability is meant to reflect the value of reconnaissance,information gathering and monitoring. A small increase in the value of p, reflecting more attentive network monitoring, can drastically hinder an attacker's ability to use high intensity attacks.

6 Learning

6.1 Algorithm

Both agents make use of function approximators to estimate the action-value function Q and use an e-greedy algorithm for exploration [4]. The simulation allows for a choice of approximators for both agents, with possible models being a linear model and a neural network with 1 hidden layer as a non-linear option. The input to the model is a state observation and a candidate action, and the output of the model is an estimate of the long term reward that this action will yield. The agent then picks the action with highest value with (1-e) probability and a random action with probability $e = 0.1$. The use of an e-greedy policy guarantees that exploration continues throughout the simulation which is important as the environment is constantly changing due to the simultaneous learning of both agents.

The agents are trained using the Monte Carlo estimation algorithm. They play one round of the game and record all of their state-action-reward triplets. At the end of the game, which happens when either one of the players win or 100 timesteps have taken place, the agents update ther estimate of the long term rewards achieved by each state-action pair using their experience from that round. The process is repeated for multiple timesteps and the behavior of agents is observed.

6.2 Proof of learning

Before the agents are placed in the multi-agent scenario, where both are learning at the same time and their behavior is constantly changing, there needs to be proof that the agents can in fact learn to maximize their long-term rewards in a simpler scenario. To obtain such proof, each agent is put against a trivial opponent, and its rewards are recorded for 500 episodes (rounds of the game). The process is repeated for 50 agents and the results per episode are averaged.

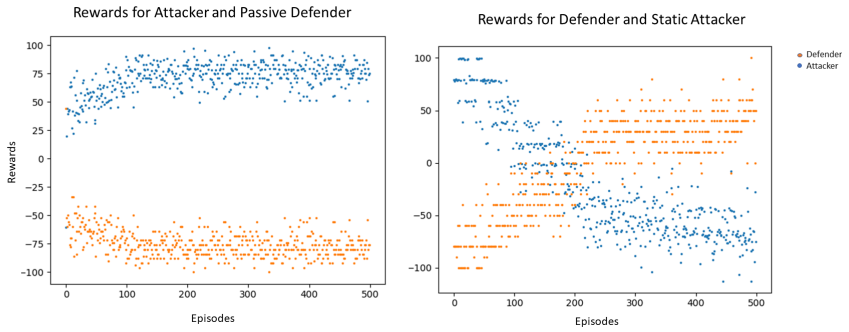


Figure 1: Rewards per episode for 500 episodes averaged over 50 agents against trivial opponents.

Figure 1 shows the rewards achieved on average by 50 agents over 500 episodes when playing against a trivial opponent. The graph on the left corresponds to the case where the attacker is learning to play while the defender does not take any actions. Over a few episodes the attacker learns to navigate to the target node and so the average reward increases. The defender's reward decreases, not because of its own actions (since it is not taking any) but because the attacker learns how to reach the target node. On the right the defender is the one learning to act but the attacker is still taking actions. The reason for this is that the defender's success depends solely on the attacker's failure, it has no other goal to achieve other than preventing the attacker from reaching its own. as a result, to test whether the defender is able to learn, we put it against a static opponent, one who does not learn and so behaves the same way every time. The static behavior chosen is to select the "correct" action (take the shortest pah to the target node) at every time step. Even under these conditions, the defender eventually learns to block the attacker from doing so.

7 Results

Now that evidence of the agent's learning has been established, the agents are pitted against each other while learning. The results shown here are obtained by having the agents play for 500 episodes on different graphs that reflect dif-

ferent cyber-security scenarios. While the previous results were averaged over 50 repeats, the results presented here are for a single pair of agents. Although this makes the results more difficult to interpret because of random effects, it is valuable to examine them so as to observe the way that agents respond to each other's behavior, a feature that would be lost if averaged over many runs. Still, care was taken to ensure that the results shown here are qualitatively representative of the results obtained when running the simulation under similar conditions.

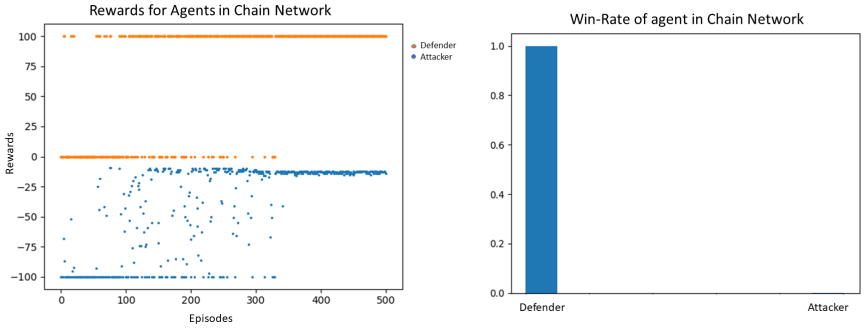


Figure 2: Graph (top), Rewards (left) and Win-rate (right) of agents playing in the chain graph (See appendix for structure and node labels)

Figure 2 represents a difficult case for the attacker. The attacker starts at node 0 and the target node is node 3. The defender dominates throughout the game, as before the attacker figures out where to attack, the defender as already learned to block access to the target node. In fact, under optimal play by both players, the defender will always win, and it learns to do so.

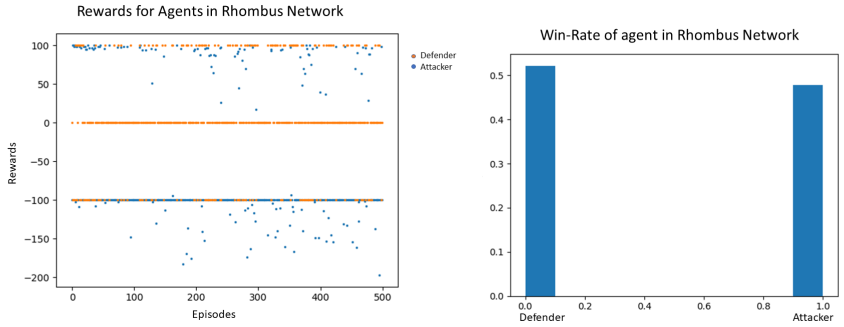


Figure 3: Graph (top), Rewards (left) and Win-rate (right) of agents playing in the rhombus graph. Multiple paths favor attacker, small network favors defender.(See appendix for structure and node labels)

Figure 3 represents a case where the attacker has multiple paths towards the target, but the network is still small. The attacker starts at node 0 and the target node is node 3. The attacker and the defender seem to oscillate between winning and losing, indicating that one is adapting to a change in strategy by the other.

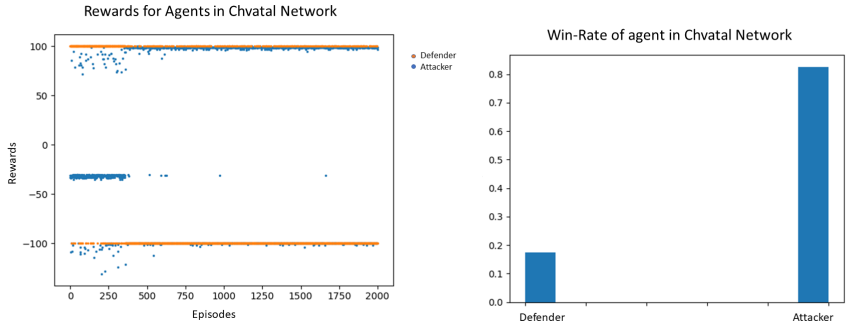


Figure 4: Graph (top), Rewards (left) and Win-rate (right) of agents playing in the Chvatal graph. Multiple paths and a large network favor the attacker.(See appendix for structure and node labels)

Figure 4 represents a case where the attacker has multiple paths towards the target and the network is large. The attacker starts at node 0 and the target node is node 11. The attacker dominates the game as there are too many nodes to defend and too many possible paths for the attacker to take.

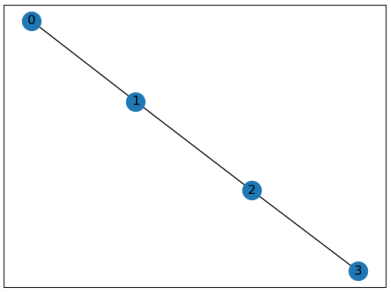
8 Conclusion

The simulation developed in this work allows us to explore the dynamics of an abstract cyber-security scenario in ways that map effectively onto the real world. For one, the game captures the fundamental problem faced by cyber-security professionals who work on defending systems from attacks, namely that defense is inherently reactive while attacks are proactive. In our game, the attacker always acts first and the defender needs to adapt. In cases where networks are small and sparsely connected (chain graph), there are fewer vulnerabilities and so its easy to defend, but the usefulness of the network (as determined by its accessibility) suffers greatly. Once the network becomes slightly more densely connected, defending becomes much harder, as the potential vulnerabilities increase quickly. In cases of large densely connected networks (Chvatal graph), like the internet, defense becomes incredibly hard, as attackers have numerous routes of attack available.

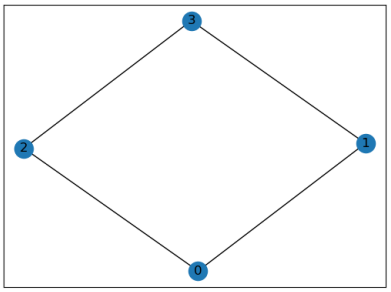
9 Appendix

Source code for this simulation can be found on github at https://github.com/panlybero/MARL-POMDP-CYBERSECURITY

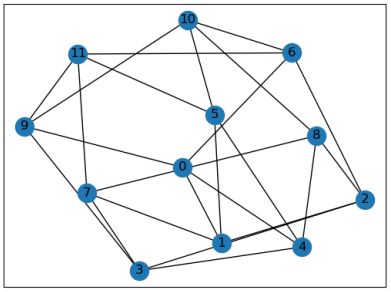
Chain Graph:



Rhombus Graph:



Chvatal Graph:



References

[1] Lucian Busoniu, Robert Babuska, and Bart De Schutter. A comprehensive survey of multiagent reinforcement learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 38(2):156–172, mar 2008.

[2] Richard Elderman, Leon J. J. Pater, Albert S. Thie, Madalina M. Drugan, and Marco M. Wiering. Adversarial reinforcement learning in a cyber security simulation. In *Proceedings of the 9th International Conference on Agents and Artificial Intelligence*. SCITEPRESS - Science and Technology Publications, 2017.

[3] Thanh Thi Nguyen and Vijay Janapa Reddi. Deep reinforcement learning for cyber security.

[4] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[5] Yang Xin, Lingshuang Kong, Zhi Liu, Yuling Chen, Yanmiao Li, Hongliang Zhu, Mingcheng Gao, Haixia Hou, and Chunhua Wang. Machine learning and deep learning methods for cybersecurity. *IEEE Access*, 6:35365–35381, 2018.