TU Delft

Delft University of Technology

What Is Your MOVE

Modeling Adversarial Network Environments

Knezevic, Karlo; Picek, Stjepan; Jakobovic, Domagoj; Hernandez-Castro, Julio

DOI
10.1007/978-3-030-43722-0_17

Publication date
2020

Document Version
Accepted author manuscript

Published in
Applications of Evolutionary Computation

Citation (APA)
Knezevic, K., Picek, S., Jakobovic, D., & Hernandez-Castro, J. (2020). What Is Your MOVE: Modeling Adversarial Network Environments. In P. A. Castillo, J. L. Jiménez Laredo, & F. Fernández de Vega (Eds.), *Applications of Evolutionary Computation : 23rd European Conference, EvoApplications 2020, Held as Part of EvoStar 2020, Proceedings* (pp. 260-275). (Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics); Vol. 12104). Springer. https://doi.org/10.1007/978-3-030-43722-0_17

Important note
To cite this publication, please use the final published version (if applicable).
Please check the document version above.

Copyright
Other than for strictly personal use, it is not permitted to download, forward or distribute the text or part of it, without the consent of the author(s) and/or copyright holder(s), unless the work is under an open content license such as Creative Commons.

Takedown policy
Please contact us and provide details if you believe this document breaches copyrights.
We will remove access to the work immediately and investigate your claim.

This work is downloaded from Delft University of Technology.
For technical reasons the number of authors shown on this cover page is limited to a maximum of 10.

What is Your MOVE: Modeling Adversarial Network Environments

Karlo Knezevic[1], Stjepan Picek[2], Domagoj Jakobovic[1], and Julio Hernandez-Castro[3]

[1] Faculty of Electrical Engineering and Computing, University of Zagreb, Croatia
[2] Delft University of Technology, Delft, The Netherlands
[3] University of Kent, Canterbury, UK

Abstract. Finding optimal adversarial dynamics between defenders and attackers in large network systems is a complex problem one can approach from several perspectives. The results obtained are often not satisfactory since they either concentrate on only one party or run very simplified scenarios that are hard to correlate with realistic settings. To truly find which are the most robust defensive strategies, the adaptive attacker ecosystem must be given as many degrees of freedom as possible, to model real attacking scenarios accurately. We propose a coevolutionary-based simulator called MOVE that can evolve both attack and defense strategies. To test it, we investigate several different but realistic scenarios, taking into account features such as network topology and possible applications in the network. The results show that the evolved strategies far surpass randomly generated strategies. Finally, the evolved strategies can help us to reach some more general conclusions for both attacker and defender sides.

Keywords: Coevolutionary algorithms · Network security · Attack/defense strategies

1 Introduction

Cyber attacks are becoming more powerful, dangerous, and prevalent due to constant improvements in attackers' strategies. There are powerful attacks like Advanced Persistent Threats (APTs) that can hide their presence, conduct reconnaissance, and run exploits [15]. Defender mechanisms, at the same time, try to detect possible threats as early as possible and run defensive actions to thwart the attacker's moves. At the same time, the defender's actions are constrained by not being disruptive to the normal operation of the network. In most modern systems, there is a significant asymmetry between attackers and defenders. That notion of asymmetry is a well-known phenomenon with many attempts to make such a relationship more balanced [16,2]. Unfortunately, while the defender needs to protect the whole system at all times, for an attacker, it is enough to find a single weakness at a certain moment in time.

As an example, consider a defender strategy where one is randomly changing the network topology to deceive an attacker (for instance, implementing a Moving Target Defense (MTD) [4]). Such a strategy will thwart many attackers, but will not be capable of adapting to any specific scenario. Ideally, we want to develop strategies that are: 1) powerful enough to fulfill their goals (e.g., mounting a successful exploit or defending against it), 2) general enough to encompass many scenarios, and 3) adaptive, to react to new not previously observed scenarios. Developing such strategies can present multiple benefits, from simply testing the limits of a system (for example, we develop a system and we want to test how resilient it is against an adaptable attacker) to finding new strategies not previously used or even considered.

In this paper, we propose a new simulator (MOVE) based on coevolutionary algorithms, which can model adversarial network cybersecurity scenarios. To do so, we abstract networks with graphs and we define several capabilities for each side.

When considering adaptive scenarios, we start with seminal work by J. Miller, where he investigated the coevolution of strategies in the repeated prisoner's dilemma [10]. To find new strategies, Miller used genetic algorithms and modeled strategies in the form of finite automata (Moore machine) with individuals encoded as strings of bits. Winterrose et al. evolved attackers' strategies against moving target defenses [17]. There, the attackers' goal is to find the optimal scheduling strategy for introducing exploits to a system by using genetic algorithms. At the same time, the defenders switch between several platforms, i.e., they use migration-based techniques, to reduce the attackers' chances of mounting a successful exploit. Due to the limited number of considered resources and the limitation to only two defender's strategies, genetic algorithms were able to find highly successful strategies in all cases. Rush et al. presented CANDLES, a system designed to coevolve attacker and defender agent strategies and evaluate potential solutions with an abstract computer network defense simulation [12]. As far as we are aware, this is the first system considering coevolutionary algorithms for security scenarios. Garcia et al. considered modeling real-world attackers and defenders in a peer-to-peer network [6]. They develop a system able to generate defense strategies where the defenders can choose one of three different network routing protocols: shortest path, flooding, and a peer-to-peer ring overlay to try to maintain their performance.

Duan et al. presented a random route mutation technique, which by randomly changing the route of the multiple flows in a network, simultaneously defends against attackers while preserving network usability [4]. Zaffarano et al. investigated the influence of moving target defenses for network environments and discussed different metrics designed to evaluate the success of MTD [18]. Achleitner et al. developed a network reconnaissance system based on SDNs to achieve deception by simulating virtual network topologies [2]. There, the authors discussed several deceptive methods to slow down the scanning attempts by an attacker who needs to scan the whole network. Prado Sánchez explored several variants of coevolutionary algorithms to model adversarial behavior in

cybersecurity domains [13]. Kelly et al. developed a system based on a coevolutionary genetic algorithm, which produces a virtual network topology that delays the attacker and detects the attacker's scan as quickly as possible [8]. Hemberg et al. used coevolutionary algorithms to examine a defensive measure called network segmentation, which divides a network into enclaves serve as threat isolation units [7].

The main contributions of this work are:

1. Design of a coevolutionary scheme able to model a wide range of realistic adversarial dynamics in complex networks. The developed simulator MOVE enables running simulations to find the best strategies for both attackers and defenders. MOVE is an open-source and publicly available project under development [1].
2. Besides modeling attackers and defenders, we also take into account the normal network traffic, which puts important constraints on defense strategies. Consequently, a defender's strategy will not only try to incapacitate attackers but simultaneously opt not to disrupt normal network operation.
3. We propose a new metric called *Network Spatial Topology* to measure changes in the network from the defender's perspective.
4. To better describe the adversarial dynamics, we add a temporal component to our strategies. As a consequence, the opponents play against each other in a series of games, where each game takes place after both players have performed their actions. The defender can maximize the impact of his actions by considering the network state at various moments in time.

Our simulator significantly differs from previous works (e.g., CANDLES [12] and RIVALS [6]), notably in contributions (2), (3), and (4) which, to the best of our knowledge, were never considered before. Naturally, the actions we define for the attacker and defender are also different from those considered in previous works.

2 Problem Definition

2.1 Network Model

Let NM be a network model consisting of a finite number of elements (e.g., switches, hosts, firewalls, honeypots, etc.). Accordingly, the network consists of a number of elements that are used in actual traffic but also some elements that are used to deceive an attacker. From the operational perspective, we distinguish between two types of elements: firewalls and everything else. The difference is that the attacker cannot conduct actions on nodes behind a firewall until he has the firewall under his control (i.e., he successfully performed an exploit on that node). Without loss of generality, we continue by calling all nodes hosts. The host elements are connected to form a network and each host has certain resources related to it. For instance, one resource can be an operating system running on a station. Each host must have at least one resource allocated to it. Finally, each node has a set of ports to serve as endpoints for communication.

We represent our network NM as an n-tuple: $< H, R, C, P >$. Here, H represents the set of hosts, R represents the set of resources, C represents the connections between hosts (i.e., their adjacencies), and P denotes the set of ports. Next, we represent every host H as an n-tuple: $< RH, PH, Firewall, Honeypot >$, where RH represents the resources allocated on the host, PH represents the ports available on a specific host, $Firewall$ denotes whether a node is a firewall (1) or not (0), and finally, $Honeypot$ denotes whether a node is a honeypot (1) or not (0).

In our model, hosts are connected with edges that can be undirected or directed (e.g., a firewall that allows traffic in one direction but not in the other). To model the connections among hosts, we use an adjacency matrix. There is a finite number of possible resources in a network, and there is a degree of similarity between some of those resources. The resource similarity is defined with the resource similarity matrix. As an example, consider resources R_1, R_2, and R_3.The resources R_1 and R_2 have a similarity equal to 0.7, while resources R_1 and R_3 have a similarity equal to 0.5. This means that if there is a successful attack on resource R_1, then there is a greater probability that the same attack will work for resource R_2 than for resource R_3. Since not all resources are equally likely to be observed in a network, we model this phenomenon through the resource popularity table that defines the probability of a resource to occur in a network. Finally, each node has a number of ports where a certain port number is reserved to identify specific services.

2.2 Attacker Model

We assume that the attacker has already infiltrated the network and is controlling one internal host (location is not known, and at the beginning of each run, it is assigned uniformly at random). The attacker does not know the network topology, the resources allocated to each host, or which ports are used. From inside the network, he can use various scan techniques to discover details about the network topology and to run remote exploits to move laterally within the network. Note that by combining these actions, we encompass the main functionalities of the Discovery phase of a cyber attack [14]. This set of actions enable the attacker to obtain (partial) information about the network, i.e., n-tuple $< H, R, C, P >$ and a number of nodes, i.e., n-tuples $< RH, PH, Firewall, Honeypot >$. Then, he attempts to exploit specific resources (e.g., operating system, services, protocols) associated with hosts/ports. The attacking approach is to find a strategy that will maximize the amount of information about hosts gathered, and consequently, the success of the exploits deployed. To allow the attacker to (theoretically) assume control of any part of a network, we assume that each resource can be exploited (with a certain probability). The set of actions that are available to the attacker are:

Scan a node of the network. This is the basic move for the attacker and the only reliable way to explore the whole network – identifying hosts, used ports, and resources as well as the information on the connectivity between the hosts. Results from the literature show that the network scanning is often the

predecessor of an attack, and consequently represents an integral part of our model [11]. We start each simulation with the assumption that the attacker does not know anything about the network, except for the node he is located at. Only once a node is scanned (e.g., using Nmap [9]), the attacker knows the potential weaknesses (resources) present there. To scan the network, the attacker can use either a horizontal scan or a vertical scan. A *horizontal scan* is a scan performed against a group of nodes for a single given port. A *vertical scan* is a scan where a single node is scanned for multiple ports. Note, we assume that the attacker can rescan nodes to keep an updated database to perform more successful attacks. It is not possible to scan the nodes behind a firewall before mounting a successful exploit on the firewall. There is a cost of scanning a node/port. To model it, we define those costs as a cost of accessing a node and the cost of scanning a port. The cost amounts can be set arbitrarily to match the actual network properties.

Attempt of an exploit. A successfully mounted exploit is the end goal of the attacker. The exploit can be attempted on both previously scanned hosts and hosts that were not scanned before. In the former case, the attacker attempts to run the exploit corresponding to the most observed resource in the scanned network. Naturally, he will attempt the exploit only on the nodes that have that resource. Additionally, the attacker will also try to mount exploits on resources that are similar to the most observed resource. In the latter case, we distinguish two strategies. The first strategy is *Maximize*, where the attacker attempts to use the exploit that is the most "popular". The motivation for this type of exploit is that the attacker can use his previous experience or available information on the network to attempt the exploit he deems most likely to be successful. If the host does not have that specific resource, the attack is not successful. The second strategy is called *Diversify* and it differs from the previous one by using random exploits. Similarly, the host must have at least one of the corresponding resources for the exploit to be successful.

We assume there is a cost of running exploits where we assign these costs uniformly at random for each resource (but this can easily be configured differently). Note, the notion of exploit cost can be interpreted as a set of actions necessary for the attacker to run an exploit (the concept used in [17]). Regardless of the interpretation, we can work without loss of generality with a notion of a *budget* that the attacker must spend to perform a scan or to launch a successful exploit.

2.3 Defender Model

The defender's goal is to minimize the success of an attacker by either preventing him from mounting exploits or at least slowing him down. To do so, the defender has at his disposal several actions that will enable him to deceive the attacker with incorrect information. Since each of the hosts can be infected, we assume that our defense strategy can deploy different defense mechanisms for each host. By utilizing these actions, the defender changes the n-tuple corresponding to the actual network (i.e., "True View Network") $TV_N =< H, R, C, P >$ into n-tuples belonging to the Virtual View Network $VV_N =< H', R', C', P' >$ and

n-tuple corresponding to host $TV_H =< RH, PH, Firewall, Honeypot >$ into $VV_H =< RH', PH', Firewall, Honeypot >$. With these actions, the defense deceives the attacker since he has access only to the virtual views. Consequently, when the attacker runs the attack, he will obtain knowledge only of a small part of a true network, which will reduce the chances of his successful exploits. Actions available to the defender are:

Add/remove a honeypot. A honeypot is isolated and monitored by the defense system, and users accessing it can be immediately labeled as attackers, since legitimate users do not enter honeypots. Each honeypot has resources/ports of the same type as real hosts and there is a limit on the maximal number of honeypots in the network.

Add/remove a connection. This action corresponds to re-routing traffic between hosts.

Add/remove a resource. This action corresponds to migration-based techniques.

Move ports. This action corresponds to replacing a resource from the present port to another randomly chosen unused one.

Note that, by default, all actions affect a randomly selected part of the network. The defender has an additional goal connected with the network usability: he must maintain the normal operation of the network, which we encode as not being possible to remove nodes (or their corresponding connections) that participate in the normal (true) traffic. The set of nodes that must not be affected by defender actions is given at the beginning of the simulation.

3 Experimental Setup

Both the defender and the attacker have at their disposal a certain *budget* for their actions. With this, we try to avoid that the evolution finds solutions that are trivial or not particularly insightful (e.g., scanning the whole network at once or adding thousands of honeypots to a physical network of only ten hosts). Different actions have different costs which limit their use. The actual values can be defined in the simulator by the user to reflect the current network model.

We differentiate three scenarios concerning the capabilities of adversaries: 1) static defender, adaptive attacker, 2) adaptive defender, static attacker, and 3) adaptive defender, adaptive attacker. In all scenarios, both defender and attacker populations are initialized at random and at the beginning each member of both populations is evaluated in simulation against every member of the opposing population. This way, we obtain an initial estimate of the quality of randomly generated attack and defense strategies.

In the first scenario where the defense is static, we then select only the best strategy from the defender population and run the evolution on the attacker population only. In every iteration of the GA, each attacker is evaluated by simulating its actions against the preselected defender. The attacker population is then subject to evolution, trying to find the best strategy against the static defender. In the second scenario, the same principle is applied inversely; the best

attacker from the initial population is used in the evolution of improved defense strategies.

Finally, in the most general case, both attacker and defender populations undergo evolution; in every iteration of the GA, we simulate the actions of each attacker against each defender and vice versa separately, and their fitness value is accumulated over all simulations. In this case, the aim is to evolve a strategy that would perform well against a wide range of opponents.

In all the scenarios, a single simulation is performed in one or more *games*: a single game includes both the attacker and the defender performing their actions up to exhausting the allotted budget. After a game is completed, their budgets are restored to the initial amount and the next game is commenced. In our experiments, a default number of 5 games is used in all experiments.

In MOVE, we use a simple genetic algorithm (GA) with a 3-tournament selection [5]. With the 3-tournament selection, three solutions are selected randomly and the worst one is discarded. From the remaining two solutions, one offspring is created by the crossover operator. We note that this algorithm has the property of elitism, which means that the best solution will always remain intact in the next population [5].

The initial population is created uniformly at random. As a stopping criterion, we use the number of generations. Each simulation is run for R runs where each run is independent and consists of G generations. The size of the attacker population is P_A and that of the defender population P_D. The simulation is run in a setting where time advances in discrete steps.

We use parameters defined for the coevolutionary algorithms as given in Table 1. Note, the mutation rate is per individual, which means there is a p_m chance an individual will be mutated, but the mutation happens only on a single, randomly selected gene.

Table 1: Coevolutionary algorithm parameters.

Parameter name	Parameter value
Number of runs R	30
Number of generations G	150
Number of games	5
Attacker (defender) population size A (B)	30
Tournament size k	3
Mutation rate per individual p_m	0.3

3.1 Encoding of Solutions

To encode strategies efficiently, we use integer representation, where each gene represents one part of the strategy. Each solution (i.e., an individual) is represented as an array of integer values – integer-based encoding. The encodings of

solutions for attackers and defenders differ, but we define one that ensures a minimal number of non-coding genes, as explained in the next section. For both sides, the values for each gene are in the range $[0, 100]$. Each value represents a *relative probability* (as a percentage) of choosing a specific action. The relative percentages are always scaled during genotype decoding so that the total sum of probabilities of all considered actions equals 100%. After an action is chosen based on those probabilities, it is performed either in full (depending on the action type) or until the budget is exhausted. If there is a remaining budget after the current action is completed, the next action is chosen and performed in the same way until the budget is depleted. Note that although it may seem possible to run an exhaustive search over the set of all possible strategies, besides the number of strategies, one also needs to take into account the network layout and all possible choices in running the strategies, making exhaustive search infeasible.

To encode the attacker, we require seven genes. The first gene determines the amount of the budget spent on the *scan* action. The *scan* action can occur only on accessible nodes (i.e., those that are not behind a firewall). After a scan action is chosen, it can be performed either via a horizontal or vertical scan, and the relative probabilities of these variants are encoded in gene two and gene three, respectively.

The fourth gene decides the probability of mounting exploits on previously scanned nodes, while the fifth gene determines the choice of exploits attempted without a prior scan. After an exploit without a prior scan is chosen, gene six determines the probability of attempting to mount the most popular exploit on a node accessible by the attacker. Conversely, gene seven determines the relative probability of mounting a random exploit on the accessible nodes of the attacker.

All actions, i.e., scan and exploit are mounted on randomly selected accessible nodes. We randomly select a known resource and a known related port to be attacked on a previously scanned node. In the case when no scan action has been performed on a node, we take a resource from a list of the most popular resources and a random ordering of ports for the exploit attempt.

To clarify the encoding, we give a small example next. For easing the interpretation, we normalize the values. Let us see what coding $|15||35||65||75||10||72||28|$ represents: The first gene corresponds to the scan action, which is chosen with 15% probability, 75% on exploits after scanning and 10% on exploits without scan. From the scan budget, we choose a horizontal scan with 35% and a vertical scan with 65%. Finally, from the budget allocated to exploit without scan, we choose to diversify with 72% and maximize with 28%. Note that not all genes need to contribute to the fitness (e.g., if the exploit without scan is 0, then the values for gene five and six would not contribute to the strategy).

The defender encoding consists of nine genes.The first gene (at position 0) decides the probability of add actions. Genes two to four decide the relative probability for adding paths, hosts, and resources, respectively. Gene five decides the probability for remove actions. Analogously, genes six to eight decide the amount of remove budget to be spent on removing paths, hosts, and resources,

respectively. Finally, gene nine decides the probability for moving ports (more precisely, replacing a resource from the present one to another randomly chosen port that is not used).

In both attacker and defender encodings, we use well-known genetic operators; namely, a simple mutation that alters a randomly chosen gene with uniform probability over the gene values, and one-point crossover between two parents.

3.2 Fitness Functions

The goal of the attacker is to mount as many as possible successful exploits and at the same time minimize the chances of being discovered by the defense system. Accordingly, he aims to maximize the following fitness function:

$$fitness_A = \#Successful\_Exploits. \tag{1}$$

If the attacker is simulated against multiple defenders (scenario 3), then the total fitness is simply the sum of all fitness values from separate simulations. The goal of avoiding being discovered is encoded implicitly in this fitness function since the attacker will not gain a reward for exploits mounted on honeypots.

The goal of the defender is to thwart the attacker by making the changes in the network where those changes cannot interfere with the normal operation of the network. To measure the changes in the network, we propose a metric called Network Spatial Topology – $NST(NM)$ that encompasses changes on hosts, paths, and resources. This metric is defined in three separate components:

$$NST(NM) = \left(\frac{n}{max\_network\_size}, \frac{number\_of\_paths}{n^2}, RSN \right). \tag{2}$$

RSN considers the number of resources on each host and their similarities:

$$RSN = \sum_{i=1}^{n} \left(\sum_{j=1}^{m} RH_{j,i} + \sum_{k=j+1}^{m} S\left(R_j, R_k\right) \right), \tag{3}$$

where n is the number of hosts, m is the maximal number of resources per host, $RH_{j,i}$ is the resource j on host i, and $S(R_j, R_k)$ is the similarity between resources R_j and R_k. To capture the effect of changes, we include a time component into the metric and compare the values of $NST(NM)$ at discrete moments t and $t+1$ occurring after each game (each move) in the simulation.

All three components of NST are normalized into the $[0, 1]$ range to avoid one component having a much larger influence than the other components. Besides *maximizing* the changes in the network configuration, the defender has as a goal to *minimize* the number of successful exploits of an attacker. Note, this could be a less realistic goal since the defense will in practice rarely know immediately that a node is compromised. Still, we add this to our fitness function to be able

to evolve strategies that can fight more actively against attackers. The number of successful exploits is normalized by dividing with the maximum number of nodes in the network ($max\_network\_size$). In our fitness function, we maximize NST and minimize the number of successful exploits (we subtract that value since the overall goal is fitness maximization):

$$fitness_D = |NST(NM, t) - NST(NM, t+1)| - \#Successful\_Exploits. \quad (4)$$

Note that we do not add information about honeypot nodes that the attacker visited into the defender fitness. This is because such discovery is a consequence of fitness (i.e., because the fitness is enabling the development of good strategies, the attacker often visits honeypots) and not a behavior encoded in the fitness function.

3.3 Simulation Parameters and MOVE Simulator

In Table 2, we give parameters defining specific scenarios we investigate. *Max network size* represents the largest network size we allow in a simulation. We work with three sizes that represent a small, medium, and large network, respectively. Naturally, we are aware that our large network is small when considering many realistic scenarios (for instance, a university could easily have 20 000 or more hosts). Still, the network sizes we use are either comparable to or larger than those explored in similar literature [2,3]. *Init network size* represents the network size at the beginning of a simulation. $Number\ of\ real\ nodes$ represents the part of the *Init network size* that is composed of real nodes, i.e., not honeypots. For instance, if the $Max\ network\ size$ is 100, then there are 60 hosts at the beginning of the simulation, and of those 60, 54 are real hosts, meaning six hosts are honeypots. Finally, $Number\ of\ firewalls$ is set to 10% of real hosts, which means there are five firewalls in the configuration. We set the costs of any defender action, as well as when the attacker is conducting the scan action (accessing a host, scanning a port) to the value of 1. For both the attacker and defender side, we allocate a budget equal to 10% of the $Initnetworksize$.

We start each run by randomly generating a network with connections/paths between those hosts and resources belonging to each host. The exploit cost for each resource is selected uniformly in the range $[1, 5]$. We report the average values and standard deviation of the fitness for both attacker and defender populations over 30 runs.

4 Results

For each scenario we consider, we give an average and standard deviation values calculated over several experimental runs. Those values are good indicators of the more general behavior and are less affected by a certain choice of experimental parameters. We do not give Min and Max values since they represent the extreme cases that are less interesting from the perspective of having good strategies.

Table 2: Network and adversary parameters.

Parameter name	Parameter value
Max network size	$[100, 500, 1\,000]$
Initial network size	60% of Max network size
Number of real nodes	90% of Initial network size
Number of firewalls	20% of real nodes
Budget for attacker	15% of Initial network size
Budget for defender	15% of Initial network size
Cost of any defender action	1
Cost of accessing a node (attacker)	1
Cost of scanning a port (attacker)	1
Max number of resources	$[10, 20, 30]$
Max number of ports on a node	20
Max number of resources per host	$[3, 5, 7]$

It is to be expected that there will be some defense strategies behaving extremely poor for certain attack strategies and good for some other strategies. Naturally, such behavior is highly dependent on the current network and adversary, so it should not be considered as a good strategy in general. One goal of evolutionary optimization is to attain a set of possible solutions that behave well over a large number of problem instances. Consequently, a smaller deviation of the best fitness values over multiple runs would imply the optimization algorithm is consistently able to perform well.

Note that attacker and defender strategies could have significantly different fitness values; this is a natural consequence of different fitness functions. In Table 3, we present results for all three scenarios we experiment with, and in the next paragraphs, we discuss the obtained results as well as give examples of evolved strategies.

4.1 S1 – Static Defender, Adaptive Attacker

Since the defense is static, we simply create uniformly at random 30 defense strategies (since the population size equals 30) and we investigate how those strategies behave against attackers that can adapt to them. First, we observe that the fitness value of the attacker increases with the network size, while the defender fitness does not show a significant increase with the increase in the network size. At the same time, we see that the standard deviation for the defense is large (the larger the network size, the larger is the standard deviation), which indicates that there are many attack strategies for which a static defense is not that successful. Consequently, for a large number of attack strategies, static defense cannot provide adequate protection. Finally, we observe how attack strategies have a steady increase in their average values, which is expected since, with the increased size of the network, the attacker also has more freedom to select where and what to attack.

Differing from that, the defense strategies do not display consistent behavior with the increase of the network size. More precisely, the best fitness is obtained for the largest network size, which is not surprising since there the defense has the most freedom to change the topology of the network significantly. More surprisingly, the worst average fitness for defense is for the medium sized network. These results simply suggest that with random changes, it is impossible to predict the behavior of the defender.

Table 3: Results for all scenarios. NS denotes the network size and SD is the standard deviation.

NS	Strategy	S1 - SDAA		S2 - ADSA		S3 - ADAA	
		Avg	SD	Avg	SD	Avg	SD
100	attack	3.23	1.17	0.39	6.09	12.70	3.69
	defense	0.77	17.51	3.63	0.80	71.30	3.71
500	attack	4.23	1.10	0.79	12.68	38.73	9.98
	defense	0.66	24.82	4.00	0.82	80.09	4.38
1 000	attack	6.10	1.58	0.94	17.45	53.50	11.40
	defense	1.19	26.41	3.87	0.93	82.05	3.91

When considering adaptive attack strategies, we give a typical high performing example for a large network: |7||55||1||95||5||93||66|. Here, we see that a large part of the budget is spent on mounting exploits after scanning, where strategy almost always prefers a horizontal scan rather than a vertical scan. Attempts to mount exploits without scan actions are deemed not to be very lucrative, fitness-wise. Still, when running the exploit without scan action, maximize strategy is preferred over the diversify strategy. Another fact to observe is the budget spent on scanning: better strategies spend lower budget in scanning, which means that the attacker scans a small part of the network and then conducts several exploits.

4.2 S2 – Adaptive Defender, Static Attacker

Analogously to the first scenario, now we randomly create a population of attack strategies and try to evolve defense strategies that are effective against such random attacker strategies. Here, we can see that having the ability to adapt reduces significantly the standard deviation of defense strategies, which means that we can evolve strategies successful against different attacks. At the same time, large standard deviation values for the attacker strategies indicate that random attacker strategies are not successful against a variety of defense strategies. Small average values for the attacker further indicate that the random strategies are not a good choice to use since such strategies will not give a good performance even for particular cases. We observe how the increase in the network size brings improvement to the fitness of the attacker. Still, while the network size increases

five times, the fitness increases only two times when comparing the small and medium networks. Similarly, when going from medium to large network (double the network size), the fitness increases for only around 20%.

Interestingly, we can see that both the average and standard deviation values for defense strategies remain similar even for significantly different network sizes. Finally, relatively small changes in the average value for the defender strategies indicate that the allocated budget is sufficient even for the largest network and that it could be smaller for the medium and small networks.

From the best evolved defense strategy for a small network |86||50||16||32||49||10||94||91||61|, we observe that add action is more lucrative than the remove action, and adding paths is the best option. When considering the remove action, we see there are many removed hosts and resources. This indicates that the strategy is aiming to produce a much better connected network while reducing the complexity of the elements in the network (by either removing the elements of a network or a number of resources). The move port action is also used relatively often, the success of which will depend on the actual number of port allocated to each host.

4.3 S3 – Adaptive Defender, Adaptive Attacker

Finally, in this setting, we run experiments where both attackers and defenders evolve and adapt over the course of the evolution process. We observe that the average values for both attack and defense strategies are higher than in the first two scenarios. With the network sizes increase, we see that the attacker's average results increase as well. When considering the standard deviation values for the attacker, we see that the values increase with the increase in the network size, which indicates that there is still a number of cases where the attack strategies show significantly various behavior, regardless good or bad. At the same time, there is a very stable behavior in the defense average values and standard deviations. This indicates that defense strategies can cope with attackers (to a certain extent) but also that it is difficult to increase the fitness of the defense by simply using a larger network. Still, the stability of the standard deviation indicates that the evolved defense strategies are indeed able to handle a variety of attacks.

The best obtained defense strategy for a small network |50||99||21||17||34||8||66||80||42| indicates that the adding action is preferred over remove action. Adding paths seem to be the best option, while from the removing side, resources represent the preferred choices. Along with moving resources to different ports, the defender balances between all three types of actions, which seems to be the most beneficial option. Despite having somewhat smaller values than in the second scenario, it is interesting to note that again remove host and remove resource are preferred actions over remove paths.

Finally, one of the best attack strategies for large networks is |1||84||99||84||0||25||31| and indicates that to maximize fitness the attacker should invest a large part of the budget into attempting exploits after scanning. We see that there seems to be no incentive for running exploits without previous scans.

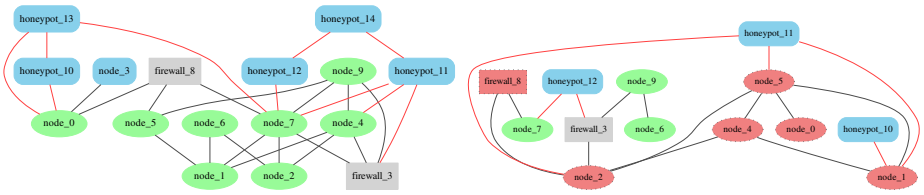
Similarly (but even more pronounced) to the first scenario, we see that the budget to be spent on scan action is very small. There, some more budget is to be spent on a vertical scan, but we can consider both scanning techniques to be similarly represented. This is a reasonable step since now the network size is large when compared to the number of ports, so using any of those options will give comparable results in terms of exploration.

4.4 Discussion

We emphasize that although we consider specific scenarios where results depend on selected parameters, there are some general conclusions we can make. First, we observe that a random strategy (regardless of whether it is defense or attack strategy) cannot compete with an evolved one. This is clear from observing standard deviation for all cases where smaller values mean that the strategy is adapted for a large number of adversaries. Additionally, larger average values indicate that the tested strategies were able to make more impact on the network. As such, we see that random strategies do not provide good adaptation against a variety of different competing strategies.

When considering the defense, we see as a general trend that add action is favored over the remove action. Additionally, we see that the best option seems to be adding paths as much as possible and removing hosts/resources. For attackers, we observe that the scan action is almost not used since the mounting of exploits could be regarded as easy. Naturally, this depends on the perspective one takes since we see that the average values indicate a number of successful exploits. These numbers are still relatively small considering the size of the tested networks but represent a serious security breach. Finally, it is better to first scan (remember, the scan is not favored, so the number of scanned nodes is not large) and then conduct exploit on scanned nodes. Since the chances of the successful exploit are then much better than mounting exploit without prior scan, a vast majority of highly successful attack strategies involve only a smaller budget in the exploit without prior scan action.

In all scenarios, we used the budget equal to 15% of the network size. The results we obtained suggest that the defender requires a larger budget to be able to cope with the attacker. Naturally, these observations only confirm the intuition that the adversarial dynamic is more difficult for the defender since he needs to protect the whole network while the attacker only needs to find some weak hosts to exploit. In Figures 1a and 1b, we depict the network topology after running both attack and defense strategies evolved for 20 generations. The real nodes are depicted in green color, the firewall in grey color, and honeypots in blue color. The nodes that are exploited by the end of the process (i.e., in the final network state) are depicted in red color. The paths that are accessed by the attacker are given in red color while those not traversed by the attacker are in black color. As can be seen, the attacker was able to successfully mount four exploits by the end of the simulation, 3 in real nodes, and 1 in the firewall. Note that the network is rather small and should serve only as an illustrative example.



(a) Initial network state. (b) Final network state

Fig. 1: Initial and evolved networks for scenario 3.

5 Conclusions and Future Work

In this paper, we present the MOVE simulator as a tool to model adversarial behavior in networks. With it, we can construct attacker and defender strategies that offer high competitiveness in networks of various sizes. Our simulator can be used either as a tool to help decide the network layout, early in the design phase or as a driving force for on-the-fly network changes in reconfigurable environments such as software-defined networks. To be able to construct defensive strategies that are more resilient over different attack strategies, we propose a new metric called the Network Spatial Topology. Besides considering the attacker and defender side, our network model also includes normal behavior, which adds additional but realistic constraints on the defender.

Our scenarios consider only cases where all attacker strategies have the same agenda. It would be interesting to add several attackers (i.e., to have more than two populations) in adversarial dynamics where those attackers can be completely independent, but can also either collaborate or compete.

References

1. Move simulator, 2019. `https://github.com/MOVESimulator/`.
2. Stefan Achleitner, Thomas La Porta, Patrick McDaniel, Shridatt Sugrim, Srikanth V. Krishnamurthy, and Ritu Chadha. Cyber Deception: Virtual Networks to Defend Insider Reconnaissance. In *Proceedings of the 8th ACM CCS International Workshop on Managing Insider Security Threats*, MIST '16, pages 57–68, New York, NY, USA, 2016. ACM.
3. Stefan Achleitner, Thomas La Porta, Patrick McDaniel, Shridatt Sugrim, Srikanth V. Krishnamurthy, and Ritu Chadha. Reconnaissance deception system prototype implementation, 2016. `https://github.com/deceptionsystem/master`.
4. Qi Duan, Ehab Al-Shaer, and Jafar Haadi Jafarian. Efficient random route mutation considering flow and network constraints. In *IEEE Conference on Communications and Network Security, CNS 2013, National Harbor, MD, USA, October 14-16, 2013*, pages 260–268. IEEE, 2013.
5. A. E. Eiben and J. E. Smith. *Introduction to Evolutionary Computing*. Springer-Verlag, Berlin Heidelberg New York, USA, 2003.

6. Dennis Garcia, Anthony Erb Lugo, Erik Hemberg, and Una-May O'Reilly. Investigating coevolutionary archive based genetic algorithms on cyber defense networks. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, GECCO '17, pages 1455–1462, New York, NY, USA, 2017. ACM.
7. Erik Hemberg, Joseph R. Zipkin, Richard W. Skowyra, Neal Wagner, and Una-May O'Reilly. Adversarial co-evolution of attack and defense in a segmented computer network environment. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, GECCO '18, pages 1648–1655, New York, NY, USA, 2018. ACM.
8. J. Kelly, M. DeLaus, E. Hemberg, and U. O'Reilly. Adversarially adapting deceptive views and reconnaissance scans on a software defined network. In *2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, pages 49–54, April 2019.
9. Gordon Fyodor Lyon. *Nmap Network Scanning: The Official Nmap Project Guide to Network Discovery and Security Scanning*. Insecure, USA, 2009.
10. John H. Miller. The coevolution of automata in the repeated prisoner's dilemma. *Journal of Economic Behavior & Organization*, 29(1):87 – 112, 1996.
11. S. Panjwani, S. Tan, K. M. Jarrin, and M. Cukier. An experimental evaluation to determine if port scans are precursors to an attack. In *2005 International Conference on Dependable Systems and Networks (DSN'05)*, pages 602–611, June 2005.
12. George Rush, Daniel R. Tauritz, and Alexander D. Kent. Coevolutionary agent-based network defense lightweight event system (candles). In *Proceedings of the Companion Publication of the 2015 Annual Conference on Genetic and Evolutionary Computation*, GECCO Companion '15, pages 859–866, New York, NY, USA, 2015. ACM.
13. Daniel Prado Sánchez. Visualizing adversaries : transparent pooling approaches for decision support in cybersecurity. Massachusetts Institute of Technology, 2018. M. Eng, thesis.
14. Symantec. Preparing for a cyber attack, January 2017.
15. Colin Tankard. Advanced persistent threats and how to monitor and deter them. *Network Security*, 2011(8):16 – 19, 2011.
16. Michael L. Winterrose and Kevin M. Carter. Strategic evolution of adversaries against temporal platform diversity active cyber defenses. In *Proceedings of the 2014 Symposium on Agent Directed Simulation*, ADS '14, pages 9:1–9:9, San Diego, CA, USA, 2014. Society for Computer Simulation International.
17. Michael L. Winterrose, Kevin M. Carter, Neal Wagner, and William W. Streilein. Adaptive attacker strategy development against moving target cyber defenses. *CoRR*, abs/1407.8540, 2014.
18. Kara Zaffarano, Joshua Taylor, and Samuel Hamilton. A Quantitative Framework for Moving Target Defense Effectiveness Evaluation. In *Proceedings of the Second ACM Workshop on Moving Target Defense*, MTD '15, pages 3–10, New York, NY, USA, 2015. ACM.