

RESEARCH ARTICLE

# Adversarial agent-learning for cybersecurity: a comparison of algorithms

Alexander Shashkov<sup>1</sup> , Erik Hemberg<sup>2</sup> , Miguel Tulla<sup>2</sup> and Una-May O'Reilly<sup>2</sup>

<sup>1</sup>Williams College, Williamstown, MA 01267, USA; e-mail: [aes7@williams.edu](mailto:aes7@williams.edu);

<sup>2</sup>MIT CSAIL, Cambridge, MA 02139, USA; e-mails: [hembergerik@csail.mit.edu](mailto:hembergerik@csail.mit.edu), [mtulla@mit.edu](mailto:mtulla@mit.edu), [unamay@csail.mit.edu](mailto:unamay@csail.mit.edu)

**Received:** 19 December 2022; **Revised:** 7 January 2023; **Accepted:** 7 January 2023

## Abstract

We investigate artificial intelligence and machine learning methods for optimizing the adversarial behavior of agents in cybersecurity simulations. Our cybersecurity simulations integrate the modeling of agents launching Advanced Persistent Threats (APTs) with the modeling of agents using detection and mitigation mechanisms against APTs. This simulates the phenomenon of how attacks and defenses coevolve. The simulations and machine learning are used to search for optimal agent behaviors. The central question is: under what circumstances, is one training method more advantageous than another? We adapt and compare a variety of deep reinforcement learning (DRL), evolutionary strategies (ES) and Monte Carlo Tree Search methods within Connect 4, a baseline game environment, and on both a simulation supporting a simple APT threat model, SNAPT, as well as CyberBattleSim, an open-source cybersecurity simulation. Our results show that when attackers are trained by DRL and ES algorithms, as well as when they are trained with both algorithms being used in alternation, they are able to effectively choose complex exploits that thwart a defense. The algorithm that combines DRL and ES achieves the best comparative performance when attackers and defenders are simultaneously trained, rather than when each is trained against its non-learning counterpart.

## 1. Introduction

Advanced persistent threat (APT) actors present a serious challenge for cybersecurity. They exhibit stealth and adaptivity while advancing through the tactics of a kill-chain and deploying adversarial techniques and exploits (Huang & Zhu, 2020). AI approaches to understanding potential APT strategy spaces, and cyber-actor behavior in general, have employed modeling and simulation (ModSim) (Zhu & Rass, 2018; Walter *et al.*, 2021; Engström & Lagerström, 2022). In ModSim cybersecurity platforms, for example CyberBattleSim (Team, 2021), agents that simulate attackers and/or defenders are executed in a model of a network environment or they are competed with an adversary. The agent strategies or behavior are optimized with machine learning in order to analyze and anticipate how an agent responds to different adversarial scenarios. For example ModSim platforms that execute Evolutionary Algorithms (EAs) operate with two adversarial populations of agents, that compete in pairs (Harris & Tauritz, 2021). One challenge is determining which machine learning training method is most effective.

In this paper we investigate the impact of different machine learning (ML) algorithms, with and without planning approaches, on agent behavioral competence in three ModSim platforms. The ModSim platforms have two-player competitive environments where adversarial agents' are players and their moves are guided by a policy, see Section 3. Connect 4<sup>1</sup>, a 2-player board game, is a simple ModSim platform. We introduce a platform named SNAPT which is cybersecurity oriented and more complicated

<sup>1</sup>[https://en.wikipedia.org/wiki/Connect\\_Four](https://en.wikipedia.org/wiki/Connect_Four).

**Cite this article:** A. Shashkov, E. Hemberg, M. Tulla and U. O'Reilly. Adversarial agent-learning for cybersecurity: a comparison of algorithms. *The Knowledge Engineering Review* 38(e3): 1–26. <https://doi.org/10.1017/S0269888923000012>

than Connect 4 both because it incorporates cyber threat concepts, and also because its two players take different roles (with different actions); that of Attacker or Defender. We also enhance CyberBattleSim (Team, 2021), a more complex simulation, by adding a Defender that competes with the pre-existing Attacker.

We compare the algorithms by competing agents trained using different algorithms against one another. The measures of comparison vary by ModSim platform: with Connect 4 agents are evaluated when they play first and when they play second. In SNAPT and CyberBattleSim all defenders trained using different algorithms compete against all attackers trained using different algorithms. In SNAPT, the comparison measure is competitions, that is ‘games’ won. In CyberBattleSim, the appropriate measure is an agent’s average reward over all competitions.

All algorithms are described in more detail in Section 4. One algorithm is an example of Deep Reinforcement Learning (DRL) where an agent’s policy is optimized temporally, for example for (maximum) expected value in the long term (Arulkumaran *et al.*, 2017; Macua *et al.*, 2021). We use the Actor-Critic approach, employing two functions: the policy function, and independently, the value function. The former returns a probability distribution over actions that the agent can take based on the given state. The value function determines the expected return for an agent starting at a given state and acting according to a particular policy forever after. These functions are implemented as Neural Networks (NNs) with predefined NN architectures. We use the Advantage Actor-Critic (A2C) algorithm to update NN parameters during training.

We also evaluate Evolutionary Algorithms. This is a class of search-based optimization algorithms inspired by biological evolution. Some EAs, for example Evolutionary Strategy (ES) algorithms, have been shown to achieve levels of success similar to those of Deep Reinforcement Learning (DRL) (Salimans *et al.*, 2017). For a fair comparison, both the EAs and DRL train agent policies represented with the same NN architectures. In an ES each NN’s weight parameters are represented as a multi-variate Gaussian distribution and updated by a gradient-free decision heuristic during training. Our comparison includes two ES algorithms where the current solution is compared to one sample. If the sample is accepted, it is used to update the distribution. The (1+1)-ES’s update formula uses average reward and contrasts with Fitness-Based ES (FB-ES)’s, which uses smoothed average reward as update formula. We also include the Cross-Entropy Method (CEM), which works with sample sizes greater than one. Because the number of competitions in CEM scale quadratically with sample size, we compare CEM to a variant where the number of competitions are conducted with round-robin play to reduce training time.

In addition, we craft two combinations of DRL and ES (Pourchot & Sigaud, 2018; Lee *et al.*, 2020) because DRL has been observed to be unstable in some cases (Mnih *et al.*, 2015), and ES tends to suffer from poor sample efficiency (Sigaud & Stulp, 2019). One combination trains alternating iterations of A2C and CEM. The other combination alternates A2C and FB-ES.

In the cyber domain, tree-based planning algorithms are used to automate adversaries, for example in the CALDERA framework (The MITRE Corporation, 2020). This motivates our consideration of policy learning in the context of planning. In the planning context a policy that directs tree search is trained and, after learning the policy, it is used directly or with look ahead supported by in-competition tree search. We evaluate AlphaZero, an algorithm that trains its policy with DRL using tree search. AlphaZero has been successful in complex board games. It relies upon Monte Carlo Tree Search (MCTS), a probabilistic and heuristic driven search algorithm (Silver *et al.*, 2018). One of our algorithms uses the Actor-Critic method (gradient-based learning) to train the policy as in the, original AlphaZero, while another trains it with CEM, that is gradient-free learning.

Because of the combinations of algorithms and platforms, we are able to perform a variety of comparisons, see Sections 5 and 6. On Connect 4 and SNAPT, all policies are represented by NNs. We consider tree search and two algorithms that train a policy using it—AlphaZero with gradient training and AlphaZero with gradient-free training. We compare these with algorithms that train policies directly, that is without tree search. We also compete all the trained policies while allowing all of them, in competitions, to look ahead with tree search. Because of our interest in cybersecurity, we further

analyze SNAPT agents to better understand the complexity of the trained policies. On CyberBattleSim, we compare all algorithms. All policies are represented by a NN and we use a more complex NN for the Attacker (than in SNAPT or Connect 4). We further investigate two aspects of the comparison. We first inquire whether changing CEM to use round-robin competition play impacts the effectiveness of CEM. As well, we compare design options for the Attacker's NN architecture.

The contributions of this paper are:

- We introduce a simple network APT environment, SNAPT, for evaluating agent-based cybersecurity attackers versus defenders.
- We enhance CyberBattleSim by introducing Defender agents.
- We introduce a specialized NN architecture for Attacker play in CyberBattleSim.
- We perform a number of quantitative comparisons and observe:
  - In Connect 4 when none of the players look ahead with MCTS during competitions with each other, the player trained with (1+1)-ES wins the most competitions on average as the first player and loses the fewest as the second player. When players can look ahead with MCTS, the two algorithms that train with tree search are superior to those that do not.
  - In SNAPT, we get a different result. The players trained with AZ-ES using tree search win more competitions as Attackers. However, Attackers do not win uniformly against all Defenders or vice versa. This suggests the existence of different strategies arising from the trained policies and asymmetry of the Attacker and Defender action spaces. Our analysis into this reveals that Attackers trained by DRL and ES, or combinations of them, are able to choose advantageous and complex exploits. We also observe that an algorithm combining DRL and ES trains the best Attacker and Defender.
  - In CyberBattleSim, we find that A2C is the best algorithm to train Attackers. For Defenders, the best is a hybrid gradient trained algorithm—A2C combined with FB-ES, which is gradient-free. When we explore CEM using round-robin play, we find that round-robin play performs better. We also find evidence that the complex NN architecture we introduced is better than a simple one, while we find no difference by the end of training between the two variants of the complex architectures.

We proceed as follows: Section 2 covers related work including Cybersecurity simulations and games, AI Planning, Deep Reinforcement Learning, Evolutionary Strategies, and Coevolution. Section 3 describes Connect 4, SNAPT, and CyberBattlesim. Section 4 describes our methodology. We give details on our training and evaluation setup in Section 5. In Section 6 we present our results. Finally, we discuss our results and future work in Section 7.

## 2. Related work

We provide an overview of related work in cybersecurity simulations and games (Section 2.1) and algorithms for training (Section 2.2) connected to cybersecurity games.

### 2.1 Cybersecurity simulations and games

Many approaches have been taken to simulating APT threats, which often take the form of a two-player game. In Zhu and Rass (2018), a multi-stage game is proposed with different action and rewards in each phase. In Luh *et al.* (2019), a physical game is used with two stages. Many similarities exist in these games, namely they tend to involve two players (an attacker and a defender) alternating moves, discrete action spaces, and asymmetric information between attacker and defender. Other cybersecurity simulation and game approaches include (Rush *et al.*, 2015; Liu *et al.*, 2018; Yang *et al.*, 2018; Baillie *et al.*, 2020; Molina-Markham *et al.*, 2021; Reinstadler, 2021).

CyberBattleSim contains many of these elements, and is also implemented using OpenAI Gym, making it well-suited for reinforcement learning (Brockman *et al.*, 2016). It also contains additional complexity compared to these environments (Team, 2021) and is one of the few that has been publicly released. Deceptive elements, including honeypots and decoys, were incorporated into CyberBattleSim in Walter *et al.* (2021).

Some of these authors have implemented various policy search algorithms in order to train automated attackers and defenders in these environments, such as AI planning or deep reinforcement learning. We describe these methods below.

## 2.2 Algorithms for training

We describe AI Planning using Monte Carlo Tree Search (MCTS), Deep Reinforcement Learning (DRL) work related to cybersecurity games, and Evolutionary Strategies (ES).

**AI Planning** Planning is a long established field of AI research focused on training agents to make optimal decisions (Jiménez *et al.*, 2012). Planning has been successfully applied in reinforcement learning domains (Partalas *et al.*, 2012). Recently, AlphaZero (Silver *et al.*, 2018) combined planning with deep learning, and was able to successfully master several complex board games. In combination with evolutionary algorithms, planning is used to solve RL environments in Olesen *et al.* (2021). Planning has seen some use in cybersecurity. In the domain of automated penetration testing, planning has been used for attack graph discovery (Applebaum *et al.*, 2016; Backes *et al.*, 2017; Falco *et al.*, 2018; Reinstadler, 2021). However, planning has not been used extensively to train agents within cybersecurity platforms.

**Deep Reinforcement Learning (DRL)** Deep reinforcement learning (DRL) is a subfield of RL focusing on using deep neural networks to represent a policy and gradient-based ML methods to train the network parameters. Interest in DRL has exploded in recent years due to breakthrough papers such as Mnih *et al.* (2015) and Lillicrap *et al.* (2016). For a survey of DRL methods, see Arulkumaran *et al.* (2017). DRL has also been applied in cybersecurity domains, for a survey see Nguyen and Reddi (2021). However, most of these environments involve a small number of actions for the attackers, and only some involve coevolving attackers and defenders together. In the initial release of CyberBattleSim (Team, 2021), automated attackers were trained using Q-learning methods without any defenders. Additionally, the action space was simplified to reduce complexity. We develop a neural network architecture that allows attackers to select complex actions within the CyberBattleSim environment and train attackers and defenders together. Our approach is similar to the action selection process in AlphaStar (Vinyals *et al.*, 2019), which was designed to select complex actions in the video game StarCraft.

**Evolutionary Strategies** Evolutionary strategies (ES) are a subset of evolutionary algorithms, a broad class of population-based optimization algorithms modeled on genetic evolution (Goldberg, 1989). ES are commonly used with a population (sample) size of one, although the Cross-Entropy method uses a sample size greater than one. For simulations where a solution has to be competed to determine its fitness, all solutions in a sample can be competed with all adversaries, resulting in a quadratic number of competitions. To reduce the number of competitions to polynomial complexity, small sample subsets can be formed and competition can be restricted to a subset. We call this round-robin play. Evolutionary strategies (ES) have seen heightened interest in RL environments due to the success of authors such as Salimans *et al.* (2017), where ES are applied to MuJoCo and Atari RL environments and shown to outperform some deep learning algorithms. Some authors have also combined ES with DRL methods; in Lee *et al.* (2020), an asynchronous framework is proposed and shown to outperform pure DRL methods in MuJoCo. DRL is also combined with population-based and exploration methods from evolutionary computation in Prince *et al.* (2021). However, to our best knowledge there are few precedents that apply ES to cybersecurity simulations.

Coevolution, broadly speaking, refers to environments where agents are trained with different (possibly competing) objective functions. Evolutionary algorithms have long been applied to coevolutionary

environments, see Potter and Jong (2000) for instance. They have also been used in predator-prey environments with two opposing agents (Simione & Nolfi, 2017; Harris & Tauritz, 2021). ES have also been applied to coevolution in two-player board games; for example in Schaul & Schmidhuber (2008), where they were able to learn some complex strategies. Planning with tree search (Liu *et al.*, 2016) and deep reinforcement learning (Klijn & Eiben, 2021) have also been utilized in coevolutionary RL environments. Finally, competitive fitness functions for one population were investigated in Panait and Luke (2002), where a one-population fitness function is optimized using a round-robin method.

### 3. Environments

We describe the environments on the platforms we use: the Connect 4 board game (Section 3.1), the Simple Network APT (SNAPT) cybersecurity environment (Section 3.2), and the CyberBattleSim cybersecurity environment (Section 3.3).

#### 3.1 Connect 4

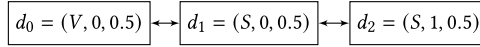
Connect 4 is a simple two-player zero-sum game which has previously been used for reinforcement learning investigations (Allis, 1988). Connect 4 is played on an upright board with six rows and seven columns. Players alternate dropping tokens into a column until one of them wins by having a line of four tokens vertically, horizontally or diagonally. If all seven columns are filled without a winner, then the game ends in a tie. While the game is simple enough to have been solved, perfect play requires complex strategies (Bräm *et al.*, 2020). Note, players in Connect 4 are neither attacker or defender, instead players are evaluated based on who plays first and second.

#### 3.2 Simple Network APT (SNAPT)

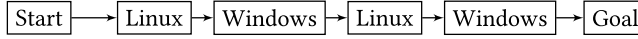
PenQuest (Luh *et al.*, 2019) provides an APT threat model designed to simulate attacker/defender behavior in an abstract network with imperfect information and is implemented as a board game. We design a simpler APT threat model inspired by PenQuest to evaluate our algorithms. It is called Simple Network Advanced Persistent Threat (SNAPT). SNAPT is implemented as a game between ‘attacker’ and ‘defender’, who alternate taking actions. The attacker starts and the game ends after a certain number of moves by each player. Note, the attacker’s actions mimic two cyberattack campaign tactics listed in the MITRE ATT&CK Matrix (Corporation, n.d.a). The change in the device’s security state through exploits mimics the tactic of privilege escalation. Attacking of adjacent nodes mimics the tactic of lateral movement. The defender mimics two defensive approaches of the MITRE Engage (Corporation, n.d. b) cyberdefense framework for adversary engagement operations. The detect action mimics the defensive goal of exposing an adversary through the approach of detection. The secure device action mimics the defensive goal of affecting the adversary with the approach of disrupting the adversaries operation.

**SNAPT States** SNAPT models a network as an unweighted graph where nodes are individual devices. Edges indicate that two devices are connected in some way (e.g. by a router). Aside from its connectivity, each device has three properties: its security state  $s$ , which is either ‘secure ( $S$ )’, ‘vulnerable ( $V$ )’, or ‘compromised ( $C$ )’; its value, which is its worth to an attacker; and its vulnerability probability, which is the likelihood that an exploit will succeed if used against it. We represent these properties as a tuple  $(s, v, p)$ , where  $s \in \{S, V, C\}$  is the security state, and  $v, p \in [0, 1]$  are the value and vulnerability probability, respectively. Figure 1 shows an example SNAPT network.

The SNAPT game begins with the one device in the network, representing a device connected to the Internet, being in a ‘vulnerable’ state, and all other devices being ‘secure’. The attacker and defender have asymmetric information on the status of the network. The attacker knows the security state  $s$  of



**Figure 1.** The SNAPT setup used for training with three nodes. Each node contains the triple of data ( $s, v, p$ ) equal to the security state( $s$ ), value( $v$ ) and exploit probability( $p$ ).



**Figure 2.** The network topology of a CyberBattleSim environment. Nodes represent devices and edges represent potential exploits sourced at the tail of the arrow and targeted at the head. The label of each node is its type. At the start of the simulation, the attacker has control of only the ‘Start’ node and needs to control the ‘Goal’ node in order to receive the full reward.

each device, the defender knows the exploit probability  $p_e$  of each device, and both players know the value  $v$  of each device.

**SNAPT actions** During an attacker’s turn, they select a device to target with an exploit. Attackers may only target devices which are in a vulnerable state, or which are connected to a device in a vulnerable state. Exploits succeed with a probability equal to the device’s vulnerability probability and raise the security state of a device from secure to vulnerable or vulnerable to compromised if they succeed.

The defender can take two kinds of mitigation actions: detecting exploits and securing devices. Detection actions are only effective against devices which were successfully compromised in the previous turn. If the defender targets a detection action against a device which was not compromised in the previous turn, the action immediately fails and nothing happens. Otherwise, the action succeeds with probability  $p_d$ . If the action succeeds, the security state of the device is reduced. If the action does not succeed, the network is left unchanged.

Moreover, the defenders have no knowledge of the security state of each device, so they have to ‘guess’ which device the attacker is likely to have targeted when using detection. If the defender targets a securing action at a certain device, the exploit probability of that device is reduced, making an attacker’s exploit less likely to succeed by a fixed value  $\delta_s$ . However, defenders have limited number of resources to secure a device and may only secure a specific device a maximum of three times. Furthermore, the attackers have no knowledge of the exploit probabilities of devices, so they are unaware if a device has been secured.

**SNAPT rewards** At the end of the game, the attacker receives a reward equal to sum of the value of all the devices in a ‘compromised’ state. Given a network with devices  $(d_1, d_2, \dots, d_n)$ , with device  $d_i$  having properties  $(s_i, v_i, p_i)$ , the reward  $R$  is given by

$$R = \sum_{1 \leq i \leq n, s_i=C} v_i \tag{1}$$

If  $R \geq 1$ , the attacker wins, otherwise the defender wins.

### 3.3 CyberBattleSim

CyberBattleSim (Team, 2021) is a network simulation in which an automated attacker attempts to take control of a network, and an automated defender tries to deter the attacker. The network is represented as a directed graph, with each node representing a device and each edge directed from node  $X$  to node  $Y$  representing an exploit sourced at device  $X$  targeting device  $Y$ , for a concrete CyberBattleSim network example see Figure 2. Each machine has a specific set of vulnerabilities which may be exploited, as well as a specific value representing its importance to the attacker and defender.

CyberBattleSim is implemented as a reinforcement learning environment, where at a given time  $t$  an agent receives an observation of the state  $s_t$ , takes an action  $a_t$  and then receives a reward  $r_{t+1}$  as well as

the observation for the next state  $s_{t+1}$ . The agent then takes an action  $a_{t+1}$  and the cycle continues until either a terminal state is reached or the maximum number of time steps is surpassed.

**CyberBattleSim States** In our CyberBattleSim environment, there are two agents, the attacker and the defender, which receive different observations at each time step in order to model the asynchronous information of a real network. Additionally, the observation given to the attacker is dependent on how much of the network the attacker has discovered and controls. For instance, the attacker receives no information on nodes it has not discovered. The specifics on the information given to the attacker and defender can be found in Appendix C.

**CyberBattleSim Actions** The attacker and defender also take different actions representative of their different roles. The attacker can take three kinds of actions at each time step: local exploits (e.g. privilege escalation techniques), remote exploits (e.g. lateral movement techniques), and connections (e.g. lateral movement techniques). These actions are abstracted as tuples:  $(\alpha, T)$  for local exploits,  $(\alpha, T, S)$  for remote exploits, and  $(\alpha, T, S, C)$  for connections, where  $\alpha$  is the action type,  $T$  is the target node,  $S$  is the source node, and  $C$  is the credential used. Note that only  $\alpha$  and  $T$  are necessary information for every type of action, while  $S$  is only necessary for remote exploits and connections and  $C$  is only needed for connections. The defender recovers against the attacker by ‘reimaging’ nodes, and only needs to choose a node  $T$  to target during the reimaging process. When a node is selected to be reimaged, it goes into an ‘offline’ state for a fixed number of time steps, making it unavailable to exploits. After reimaging, nodes are returned to their original state, without any exploits used against them. Only certain nodes are available for reimaging (representing non-critical infrastructure in a network) and the defender may only reimage a node every  $t_i$  time steps. The defender may also choose to leave the network unchanged. The specifics on the information given to the attacker and defender can be found in Appendix C.

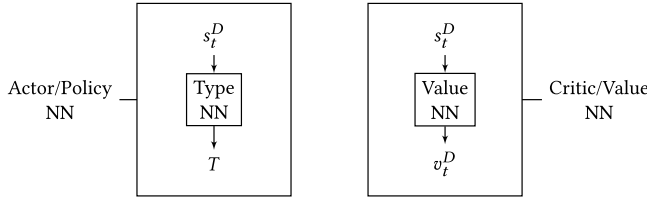
**CyberBattleSim Rewards** The attacker earns rewards for successfully using exploits against nodes in the network, and loses rewards when an exploited node is reimaged. The defender’s reward is the negative of the attacker’s reward. The reward given by reimaging a node is equal to the negative sum of the rewards of all exploits used against that node since its last reimaging. For instance, if an attacker has used 2 exploits against a node  $X$  for a total reward of 100, the attacker will be given a reward of  $-100$  when the node is reimaged and the defender will be given a reward of 100. In the initial release of CyberBattleSim, no rewards were given out due to reimaging, and we make this modification in order to give the defender greater influence over the rewards in the network. Denote the attacker’s reward at time  $t$  by  $r_t^A$  and the defender’s reward by  $r_t^D$ . As the defender only takes an action every  $t_i$  time steps,  $r_t^D = 0$  if  $t \not\equiv 0 \pmod{t_i}$ . If  $t \equiv 0 \pmod{t_i}$ , then the reward given to the defender is the negative of the attacker’s reward from the previous  $t_i$  time steps:

$$r_t^D = - \sum_{i=t-t_i+1}^t r_i^A$$

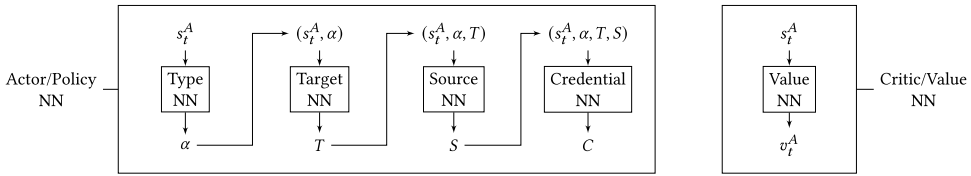
The simulation ends when the attacker reaches a winning state or the current time  $t$  is greater than some predefined threshold  $t_{\max}$ . Winning states are those where the attacker has complete control of the network or the number of nodes in an ‘offline’ state due to reimaging exceeds a predefined threshold, in which case a reward is given to the attacker and a penalty given to the defender. Thus, the defender’s goal is to reimage nodes strategically to deter the attacker while making sure that enough of the network is online so that the simulation does not end.

#### 4. Methods

We implement the policy of the adversaries on each platform environment as a neural network (NN). Section 4.1 describes NN architectures. Section 4.2 describes the policy learning algorithms.



**Figure 3.** The neural network (NN) architecture used for the defender. When choosing an action, the actor/policy NN picks a node  $T$  to reimagine or to leave the network unchanged (this can be seen as picking a ‘null’ node). The value is chosen separately by the critic/value NN.



**Figure 4.** The multi-stage neural network (NN) architecture used for the attacker. When choosing an action, each element of the tuple is selected individually and passed onto the next stage of the actor/policy neural network. The ordering of the Type NN and Target NN may be swapped, and the Source NN and Credential NN are only used for certain action types. The value is chosen separately by the critic/value NN.

**4.1 Neural network architectures**

**Connect 4** We represent each player’s policy with an actor-critic approach which uses two feed-forward NNs with two hidden layers. They map state observation vectors to action probabilities and values. The value  $v_t$  of a state is the expected reward of that state, see Figure 3.

**SNAPT** For SNAPT the Defender’s policy and value neural networks are two feed-forward NNs with two hidden layers, similar to Connect 4, see Figure 3.

**CyberBattleSim** In CyberBattleSim, due to a more complex action space, Attackers use a multi-stage NN similar to the approach introduced in Metz *et al.* (2017) and implemented in Vinyals *et al.* (2019). Each stage is a feed-forward neural network with two hidden layers. The policy network first passes the state  $s_t^A$  into the ‘attack type neural network’ which returns a probability vector for the attack type  $\alpha$  (see Section 3.3). Once  $\alpha$  is chosen using the given probabilities,  $\alpha$  is concatenated with  $s_t^A$  and passed into the ‘target neural network’ which returns a probability vector for the target node  $T$ . The order of these steps: Type before Target, can be switched to Target before Type. We compare the two orderings in our experiments.

Should a source decision be needed,  $T$  is concatenated onto the state vector (along with  $\alpha$ ) and passed into the ‘source neural network’. If the action type is ‘connection’, then finally  $\alpha$ ,  $T$ , and  $S$  are concatenated onto  $s_t^A$  and passed into the ‘credential neural network’.

This approach decomposes the attacker’s action selection into at most four steps, each with a significantly smaller output vector. A summary of the attacker neural network design is shown in Figure 4.

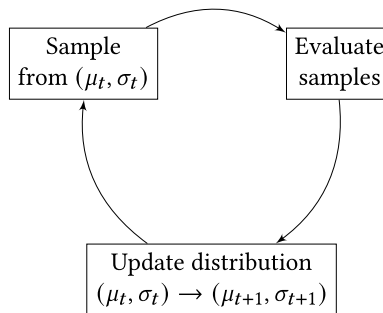
**4.2 Policy learning algorithms**

We characterize the different algorithms we use to train our policies in Table 1. The algorithms vary in parameter updates during training, the number of neural network samples used, and the use of tree search.



**Table 1.** Overview of training methods. The methods are chosen based on how they update the neural network. The training choices are gradient descent and/or weighted average, the number of samples (#Samples) used, and if tree search is used

Name	Training	#Samples	Tree search
A2C	Gradient descent	1	No
AZ	Gradient descent	1	Yes
(1 + 1)-ES	Best sample	2	No
AZ-ES	Best sample	2	Yes
CEM	Best sample	N	No
A2C+CEM	Gradient descent & best sample	N	No
FB-ES	Smoothed best sample	2	No
A2C+FB-ES	Gradient descent & smoothed best sample	2	No



**Figure 5.** The iterative process used for evolutionary strategies (ES). At iteration  $t$ , a set of samples is taken from the distribution  $(\mu_t, \sigma_t)$ . These samples are then evaluated by executing episodes of CyberBattleSim. Sample performance is determined by average reward from these episodes. The distribution is then updated towards the better performing samples. The process then begins again with the new distribution  $(\mu_{t+1}, \sigma_{t+1})$ .

#### 4.2.1 Advantage Actor-Critic (A2C)

Advantage Actor-Critic (A2C) is a common deep reinforcement learning (DRL) algorithm (Grondman *et al.*, 2012). After completing a predetermined number  $G_{A2C}$  of rollouts utilizing an attacker and defender that are not updated, the A2C policy gradient for the attacker and defender is calculated and back-propagated to update the attack and defense policies through gradient descent.

#### 4.2.2 Evolutionary Strategies (ES)

Evolutionary strategies (ES) are a class of optimization algorithms based on biological evolution (Rechenberg, 1989). A high-level overview of ES algorithms can be found in Figure 5. We utilize three ES algorithms: (1 + 1)-ES (Droste *et al.*, 2002), fitness-based ES (Lee *et al.*, 2020), and the cross-entropy method (CEM) (Hansen, 2016). All three rely on iterating a multi-variate Gaussian distribution of the parameters of the attacker and defender network. The distributions at time  $t$  are given by  $(\mu_t^A, \sigma_t^A)$  and  $(\mu_t^D, \sigma_t^D)$  for the attacker and defender, respectively, where  $\mu_t$  is the mean and  $\sigma_t$  is the covariance matrix. We will sometimes omit the superscript  $A$  or  $D$  in writing operations which are performed on both distributions. In order to reduce computational complexity, we constrain  $\sigma_t$  to be diagonal as in

Pourchot and Sigaud (2018). Thus  $\sigma_t$  can be interpreted as a vector, and we write  $\sigma_t^2$  to mean the square of each element in  $\sigma_t$ . Our ES methodology is derived from Lee et al. (2020).

**(1+1)-ES (Droste et al., 2002)** In (1 + 1)-ES, a single-sample  $z^A$  and  $z^D$  is taken from the distributions  $(\mu_t^A, \sigma_t^A)$  and  $(\mu_t^D, \sigma_t^D)$ . A fixed number  $G_{(1+1)}$  of simulations are rolled out for each of the four attacker-defender pairings  $(\mu_t^A, \mu_t^D)$ ,  $(\mu_t^A, z^D)$ ,  $(z^A, \mu_t^D)$ ,  $(z^A, z^D)$  and the average reward for each agent over the simulations is calculated. In this way the mean and sample face each opponent the same number of times, allowing for a more balanced evaluation. We are also able to evaluate attackers and defenders simultaneously this way, reducing the number of competitions executed in each training step. If the sample  $z$  outperforms  $\mu_t$  (i.e. it has a higher average reward), the distribution is updated by

$$\mu_{t+1} = z \quad \text{and} \quad \sigma_{t+1}^2 = (\mu_t - z)^2 \tag{2}$$

In order to guarantee convergence, we also introduce an update to  $\sigma_t$  with  $\sigma_{t+1} = 0.99\sigma_t$  if  $\mu_t$  performs the same or better as  $z$ .

**Fitness-based ES (Lee et al., 2020)** Fitness-based ES (FB-ES) is similar to (1 + 1)-ES, as one sample  $z^A$  and  $z^D$  are taken from each distribution and a fixed number  $G_{fit}$  of simulations with each of the four possible pairings are rolled out. Then,  $\mu_t$  is updated if and only if the sample  $z$  outperforms it. However, instead of utilizing the somewhat aggressive update strategy of (1+1)-ES, the distribution is updated using a smoother formula. When the sample  $z$  outperforms  $\mu_t$  the distribution is updated by

$$\mu_{t+1} = (1 - p_z)\mu_t + p_z \cdot z \quad \text{and} \quad \sigma_{t+1}^2 = \sigma_t^2 + \frac{(z - \mu_t)^2 - \sigma_t^2}{n_z} \tag{3}$$

where  $p_z$  and  $n_z$  depend on the relative performance of  $z$  and  $\mu_t$ . Let  $f(z)$  and  $f(\mu_t)$  denote the average reward of  $z$  and  $\mu_t$ , respectively. We use a *relative fitness baseline* to calculate  $p_z$  and Rechenberg’s method (Rechenberg, 1989) to calculate  $n_z$ :

$$p_z = \frac{f(z) - f(\mu_t) + b}{f(z) - f(\mu_t) + 2b} \quad \text{and} \quad n_z = \max\left(\frac{1 - p}{p}, 1\right) \tag{4}$$

Here,  $b^A$  and  $b^D$  are fixed parameters known as *fitness baselines*. These specifications were chosen due to their success in Lee et al. (2020).

**CEM (Hansen, 2016)** In the cross-entropy method (CEM), two samples  $\{z_1^A, z_2^A, \dots, z_N^A\}$  and  $\{z_1^D, z_2^D, \dots, z_N^D\}$ , each of predetermined size  $N$  are taken from the attacker and defender distributions, respectively. Each attacker executes a fixed number,  $G_{CEM}$ , of competitions against each defender and the average reward determines their fitness. Note that setting up every attacker or defender to plays against the same field of adversaries supports unbiased evaluation. It also supports evaluating the attackers and defenders in unison, halving the training time. Lastly, since each agent plays against a variety of adversaries, they may be more robust.

The attacker and defender distributions are updated by

$$\mu_{t+1} = \sum_{i=1}^K \lambda_i x_i \quad \text{and} \quad \sigma_{t+1}^2 = \sum_{i=1}^K \lambda_i (x_i - \mu_t)^2 + \epsilon \tag{5}$$

where  $\lambda_i$  is a weight and  $\epsilon$  is a fixed noise value to prevent the variance from vanishing. We set  $\lambda_i = \frac{1}{i \cdot H_K}$ , where  $H_K = \sum_{i=1}^K \frac{1}{i}$  is the  $K$ -th harmonic number. Here,  $K$  is a fixed parameter known as the *elite size* and  $x_i^A$  and  $x_i^D$  are the  $i$ -th best-performing attacker and defender, respectively.

The setup of this system means that a total number of  $G_{CEM}N^2$  episodes must be executed during each training iteration, making it expensive for large values of  $N$ . An example of the standard and round-robin average reward (mean expected utility) calculation can be found in Table 2.

**Table 2.** Example calculations (using mock numbers) for the round-robin system introduced in Section 4.2.2 with  $N = 3$ . Each cell gives the average reward from executing  $G_{CEM}$  episodes between an attacker (given by the row) and defender (given by the column). The average for each attacker and defender is given at the end of each row and column. The best-performing attacker is  $z_2^A$  and the best-performing defender is  $z_3^D$ , so these samples will be relabeled as  $x_1^A$  and  $x_1^D$  and have the largest weight when calculating  $\mu_{t+1}^A$  and  $\mu_{t+1}^D$ , respectively

Attacker/Defender	$z_1^D$	$z_2^D$	$z_3^D$	Mean
$z_1^A$	100	97	94	97
$z_2^A$	102	99	102	101
$z_3^A$	95	92	89	89
Mean	99	96	95	

#### 4.2.3 DRL and ES in combination

We combine DRL and ES methods by alternating between executing one iteration of A2C with one iteration of the two single-sample ES algorithms as in Pourchot and Sigaud (2018).

#### 4.2.4 Planning with policy-guidance

**AlphaZero (AZ)** (Silver *et al.*, 2018) is an algorithm relying on planning which has been successful in complex board games. During a rollout, AlphaZero plays itself utilizing the policy and value network to calculate move probabilities and position values through Monte Carlo tree search (MCTS). After completing the tree search, the NN is updated according to a loss function

$$\ell = (z - v)^2 + \pi^T \cdot \log p \tag{6}$$

where  $z$  is the value calculated from the tree search,  $v$  is the value given by the value network,  $\pi$  is the action probabilities calculated from the tree search, and  $p$  is the action probabilities given by the policy network. The NN is then updated through gradient descent on the loss function.

**AlphaZero-ES (AZ-ES)** This substitutes a gradient-free method for updating the NN, during the last step of the AlphaZero algorithm. We take  $N$  samples from a distribution  $(\mu_t, \sigma_t)$  and select the  $K$  samples with the lowest loss function. We then update the distribution with the cross-entropy method

$$\mu_{t+1} = \sum_{i=1}^K \lambda_i z_i \quad \text{and} \quad \sigma_{t+1}^2 = \sum_{i=1}^K \lambda_i (z_i - \mu_t)^2 + \epsilon. \tag{7}$$

### 5. Experimental setup

In this section we give the setup of the training algorithms and environments used in the ModSim platforms. We also cover our policy evaluation procedures. All relevant code can be found in Group (2022). In Table 3 we provide an overview of how we evaluate the training methods.

**Computational Hardware** We train and compare all algorithms using a 4-core Intel Xeon CPU E5-2630 v4 at 2.2 GHz with 8GB of RAM.

**Connect 4 & SNAPT** We train the NNs of each algorithm in Table 4 for one hour in the environment settings listed. During training, the AlphaZero methods had the fewest iterations with fewer than 100 in each case, while CEM was typically able to complete several hundred iterations. A2C and (1 + 1)-ES both completed several thousand iterations. In order to compare the algorithms, each trained NN plays

**Table 3.** Overview of training method evaluation. Column shows the evaluation setup and results section. Row shows the training method. CBS is CyberBattleSim

Training method	Connect 4 Section 6.1		SNAPT Section 6.2		CBS Section 6.3
	No MCTS	MCTS	No MCTS	MCTS	No MCTS
AZ	✓	✓	✓	✓	
AZ-ES	✓	✓	✓	✓	
A2C	✓	✓	✓	✓	✓
(1 + 1)-ES	✓	✓	✓	✓	✓
CEM	✓	✓	✓	✓	✓
Random	✓	✓	✓	✓	✓
A2C+CEM					✓
FB-ES					✓
A2C+FB-ES					✓
None (no defender)					✓
CEM round-robin					✓
A2C+CEM target first					✓
A2C+CEM simple					✓

100 matches twice against each of the other trained NN, once as the first player (1st) and once as the second (2nd).

The SNAPT platform's competition environment is a simple network of three nodes with topology as shown in Figure 1. The various algorithms and environments have many hyperparameters, shown in Table 4.

**CyberBattleSim** We train the NNs of each algorithm for 24 hours. The NNs each have two hidden layers of size 512. Each algorithm requires a set of constants which we list in Table 5. We set the constants so that on each training iteration 64 episodes of CyberBattleSim are executed. The number of iterations executed during the 24 hour training period for each algorithm is shown in Table B.1.

The CyberBattleSim environment is a network with 6 nodes of 4 different types (we use the built-in 'CyberBattleChain' Group, 2022). A visualization of the network topology can be found in Figure 2. The attacker's observations have dimension 114 and the defender's observations have dimension 13, details in Appendix C. There are 15 total attack types (5 local exploits, 2 remote exploits, and 8 connections), meaning the attack type neural network has an output dimension of 15 (see Appendix C). As there are six nodes the target and source neural networks have output dimension 6. There are 5 total credentials in the environment so the credential neural network has output dimension 5. In total there are 1542 unique actions available to the attacker. There are 7 actions available to the defender, one for each node and one 'do nothing' action. The maximum number of steps in each simulation  $t_{\max}$  is 100 and each node takes 15 time steps to reimagine.

During training, we record the reward for each attacker when simulated against the coevolving defender as well as in an environment without a defender. The difference in these quantities can be interpreted as the quality of the defender. In order to compare the attackers and defenders, we execute 100 simulations with each attacker and defender pairing and report the average reward. In addition, we introduce a control by testing each training method against an untrained attacker and defender which choose each action randomly. Finally, we also test each attacker in an environment without a defender.

**Table 4.** Environment and algorithm parameters used in Connect 4 and SNAPT

Environment	Parameter	Value
Connect 4	board width	7
	board height	6
SNAPT	player moves	20
	total moves	40
	exploit probability ( $p_e$ )	0.7
	detect probability ( $p_d$ )	0.7
	probability decrease ( $\delta_s$ )	0.1
Algorithm	Parameter	Value
A2C	Adam learning rate	0.001
(1 + 1)-ES	$G$	50
CEM	$N$	8
	$K$	4
	$\epsilon$	$10^{-6}$
AZ	$G_{AZ}$	10
	MCTS iterations	50
AZ-ES	$N$	8
	$K$	4
Random	Probability	Uniform

**Table 5.** Environment and algorithm parameters used with CyberBattleSim

Algorithm	Parameter	Value
A2C	$G_{A2C}$	64
	Adam learning rate	$10^{-4}$
(1 + 1)-ES	$G_{(1+1)}$	16
FB-ES	$G_{fit}$	16
	$b^A$	200
	$b^D$	-200
CEM	$G_{CEM}$	1
	$N$	8
	$K$	4
	$\epsilon$	$10^{-6}$

**Table 6.** Comparison of Connect 4 training algorithms without players using tree search (top) and with all players using tree search (bottom). Rows are the first player, and columns the second. Each cell shows how many games out of 100 the first player won against the second. The final column gives the average of each row and the final row gives the average of each column. The largest value in each column is **bolded** to highlight the best-performing attacker. The smallest value in each row is underlined to highlight the best-performing defender

Connect 4 games won by 1st player. No use of tree search							
1st/2nd	A2C	AZ	AZ-ES	CEM	(1 + 1) – ES	Random	Mean
A2C	59	55	61	48	<u>38</u>	48	51.5
AZ	63	54	64	<u>48</u>	<u>48</u>	60	56.2
AZ-ES	<u>43</u>	52	60	44	47	59	50.8
CEM	59	61	50	53	<u>53</u>	<b>72</b>	58.0
(1 + 1)ES	<b>66</b>	<b>62</b>	<b>69</b>	<b>69</b>	<u>44</u>	67	<b>61.3</b>
Random	54	51	52	48	<u>40</u>	59	50.7
Mean	57.3	55.8	59.3	51.7	<u>45.0</u>	60.8	
Connect 4 games won by 1st player. Both players use tree search							
1st/2nd	A2C	AZ	AZ-ES	CEM	(1 + 1)ES	Random	Mean
A2C	72	51	<u>30</u>	55	58	64	55.0
AZ	72	55	<u>46</u>	73	76	71	65.6
AZ-ES	<b>84</b>	<b>75</b>	<u>47</u>	<b>91</b>	<b>84</b>	<b>85</b>	<b>77.7</b>
CEM	68	36	<u>33</u>	56	58	69	53.3
(1 + 1)ES	72	51	<u>30</u>	55	58	64	44.7
Random	69	61	<u>30</u>	38	48	58	49.2
Mean	72.8	54.8	<u>36.0</u>	61.3	63.7	68.5	

## 6. Results

We present results from in Connect 4 (Section 6.1), then SNAPT (Section 6.2), and finally in CyberBattleSim (Section 6.3). For an overview of the experiments and see Figure A.1.

### 6.1 Connect 4

The Connect 4 player training algorithm comparisons can be found in Table 6. In the games where players did not use look-ahead tree search (MCTS), the player trained by (1 + 1)-ES won the most games on average when playing first and lost the fewest on average when playing second. This ranking is followed by CEM training. When utilizing Monte Carlo Tree Search (MCTS) during play, the player trained by AlphaZero-ES, followed by AlphaZero, won the most games on average when playing first and lost the fewest on average when playing second. We also tested using MCTS for only one opponent and each algorithm was able to win more than 90% of its games against opponents that did not use tree search, so these results are not shown.

### 6.2 SNAPT

The SNAPT algorithm comparisons can be found in Table 7. In the competitions where the adversaries did not use look-ahead tree search, the AZ-ES-trained Attacker won the most competitions, with

**Table 7.** Comparison of SNAPT Attacker vs. Defender competitions without agents using tree search (top) and with both adversaries using tree search (bottom). Rows are the Attackers and columns the Defenders. Each cell shows how many competitions out of 100 the Attacker won against the Defender. The final column gives the average of each row and the final row gives the average of each column. The largest value in each column is **bolded** to highlight the best-performing attacker. The smallest value in each row is underlined to highlight the best-performing defender

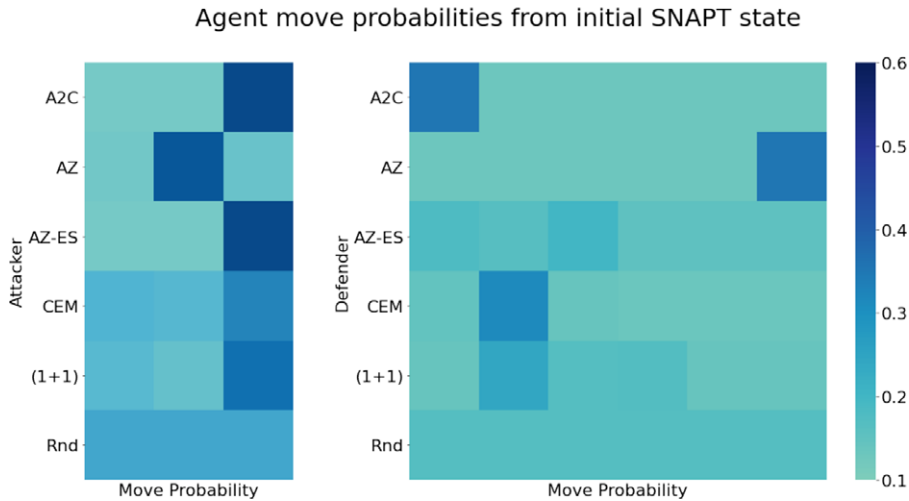
SNAPT competitions won by Attacker. No use of tree search							
Attacker/Defender	A2C	AZ	AZ-ES	CEM	(1 + 1)ES	Random	Mean
A2C	94	95	89	<b>33</b>	<u>22</u>	42	62.5
AZ	65	77	29	<u>4</u>	8	31	35.7
AZ-ES	<b>100</b>	<b>100</b>	<b>100</b>	30	<u>29</u>	49	<b>68.0</b>
CEM	13	18	14	<u>12</u>	28	47	22.0
(1 + 1)ES	<u>11</u>	14	<u>11</u>	18	<b>32</b>	38	20.7
Random	<u>4</u>	6	6	6	19	37	13.0
Mean	47.8	51.7	41.5	<u>17.2</u>	23.0	40.7	
SNAPT competitions won by Attacker. Both adversaries use tree search.							
Attacker/Defender	A2C	AZ	AZ-ES	CEM	(1 + 1)ES	Random	Mean
A2C	<u>25</u>	41	27	32	40	38	33.8
AZ	<u>27</u>	<b>42</b>	35	33	35	30	33.2
AZ-ES	<b>39</b>	38	35	<u>30</u>	38	34	35.8
CEM	<u>28</u>	31	<b>38</b>	<u>35</u>	<b>43</b>	37	35.3
(1 + 1)ES	<u>29</u>	31	30	36	39	38	33.8
Random	37	36	37	<b>40</b>	<u>31</u>	<b>50</b>	38.5
Mean	<u>30.8</u>	36.5	33.7	34.0	37.7	37.8	

an average of 68.0 wins, followed by the Attacker trained by A2C with 62.5. These two algorithms were particularly successful against the Defenders trained with A2C, AZ, and AZ-ES, with the Attacker trained by AZ-ES winning all of its competitions against these Defenders.

Among Defenders, the CEM-trained agent performed the best, losing on average 17.2 out of 100 games, followed by agents trained by AZ and (1 + 1)-ES with an average of 23.0. Like the Attackers, the success of the Defenders was not consistent. The A2C, AZ, and AZ-ES-trained Defenders performed better than the random Defender against the CEM-trained, (1 + 1)-ES-trained, and Random-trained Attackers, while performing worse than the random Defender against the A2C-trained, AZ-trained, and AZ-ES-trained Attackers.

When using MCTS, all the models performed similarly, with the Random attacker winning the most games on average. One possible explanation for this behavior is that the MCTS was much stronger than the models themselves, so the relative difference in their performance was not apparent when utilizing MCTS.

**Initial States Output Probabilities** The inconsistent success of the Attackers and Defenders suggests that the trained models use different strategies. For instance, if all of the Attackers were converging towards one global optima in training, the best Attackers would likely perform the best against all of the Defenders. To gain more insight into this, we examined move selections made by the trained models on the initial state. A heatmap showing the probability of each agent taking a given move from the



**Figure 6.** A heatmap showing the move (output) probabilities for the trained SNAPT agents at the initial state. The row is the Attacker or Defender and the column is the move (output), with darker shades indicating the agent is more likely to make a certain move. The Attacker has 3 possible moves, and the Defender has 6 possible moves.

initial state is given in Figure 6. The heatmap shows that different agents favor different moves in the initial state. Random, as expected, is the only one that has uniform probabilities. While only looking at the initial state does not fully explain the performance of the agents against each other, it does suggest the emergence of varying strategies even in a simple SNAPT environment. See Section 7 for further discussion.

### 6.3 CyberBattleSim

Table 8 presents the average reward from executing 100 episodes of CyberBattleSim. A larger value between agents indicates that one agent outperformed the other. While the absolute rewards offer no interpretation, the relative rewards reflect comparative performance of the algorithms. The standard deviations for these data are given in Table C.1.

The best-performing attacker is A2C, with an average reward of 349 across all defenders, followed by A2C+FB-ES with 345 and A2C+CEM with 340. The pure ES methods FB-ES and CEM also perform well with 323 and 293, respectively, and (1+1)-ES was only able to achieve an average reward of 98, while the untrained random attacker had an average reward of 38.

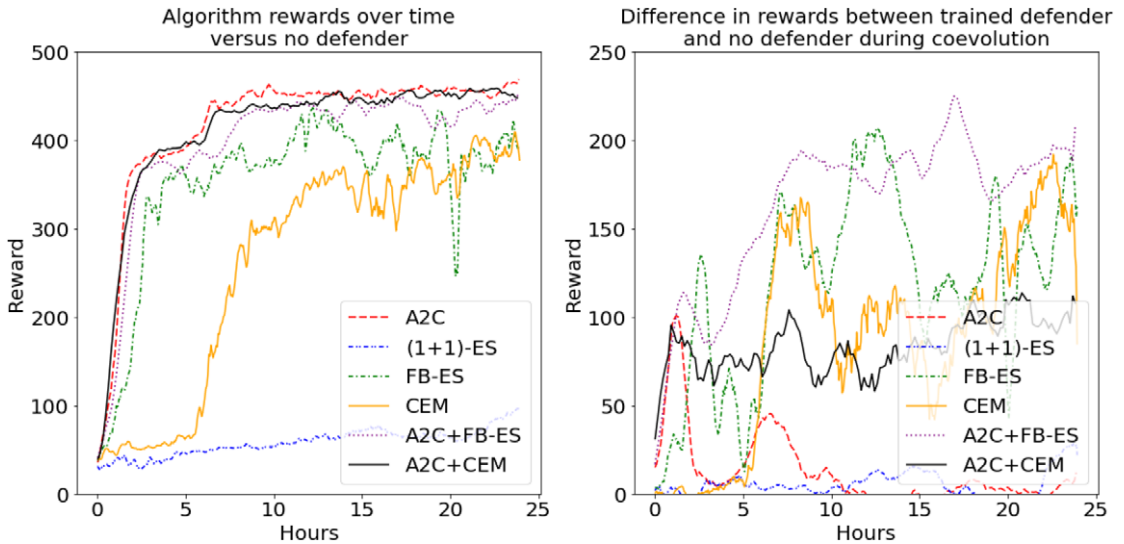
Among defenders, A2C+FB-ES performs best, allowing an average reward of 189 (or, alternatively, achieving an average reward of  $-189$ ), followed by FB-ES, CEM and A2C+CEM which allow an average reward of 205, 216 and 233, respectively. Notably, A2C+FB-ES also was the best-performing defender against each individual attacker as well. A2C and (1+1)-ES allowed an reward of 296 and 300, performing worse than the untrained random defender which has an average of 279 but better than no defender which has an average of 325. Overall, it appears that A2C+FB-ES is the best-performing defender and is among the best attackers (along with A2C and A2C+CEM).

**Reward over time** The reward over time versus no defender for the attackers trained using all the algorithms and the regular Type First neural network is shown in Figure 7 (Left). All algorithms except for (1+1)-ES appear to train the attacker to similar final rewards, with A2C, A2C+FB-ES and A2C+CEM having the best stability. The reward over time for each attacker versus the coevolving defender is somewhat uninformative as it depends on the performance of both the attacker and defender. However, by



**Table 8.** The average rewards (rounded to the nearest integer) from 100 episodes in each attacker and defender pairing. The column gives the defender and the row gives the attacker. We include an untrained attacker and defender which randomly select each action as a control as well as an environment with no defender. The averages of each row and column are given at the end. The largest value in each column is **bolded** to show which attacker performed best against a given defender. The smallest value in each row is underlined to show which defender performed the best against a given attacker

Attacker/Defender	A2C	(1+1)-ES	FB-ES	CEM	A2C+ FB-ES	A2C+ CEM	Random	None	Mean
A2C	<b>422</b>	412	270	266	<u>255</u>	322	382	<b>467</b>	<b>349</b>
(1+1)-ES	101	97	98	97	<u>94</u>	100	97	102	98
FB-ES	357	380	275	275	<u>244</u>	293	362	396	323
CEM	339	341	215	254	<u>194</u>	283	340	374	293
A2C+FB-ES	410	410	260	<b>304</b>	<u>250</u>	298	<b>386</b>	444	345
A2C+CEM	404	<b>414</b>	<b>277</b>	274	<u>255</u>	295	353	452	340
Random	38	46	41	40	<u>33</u>	38	<u>33</u>	38	38
Mean	296	300	205	216	<u>189</u>	233	279	325	255



**Figure 7.** (Left) The attacker's reward over time versus no defender for each algorithm used in training. (Right) The difference in reward between the attacker versus the trained defender and the attacker versus no defender during coevolution. The rewards are smoothed by averaging consecutive terms.

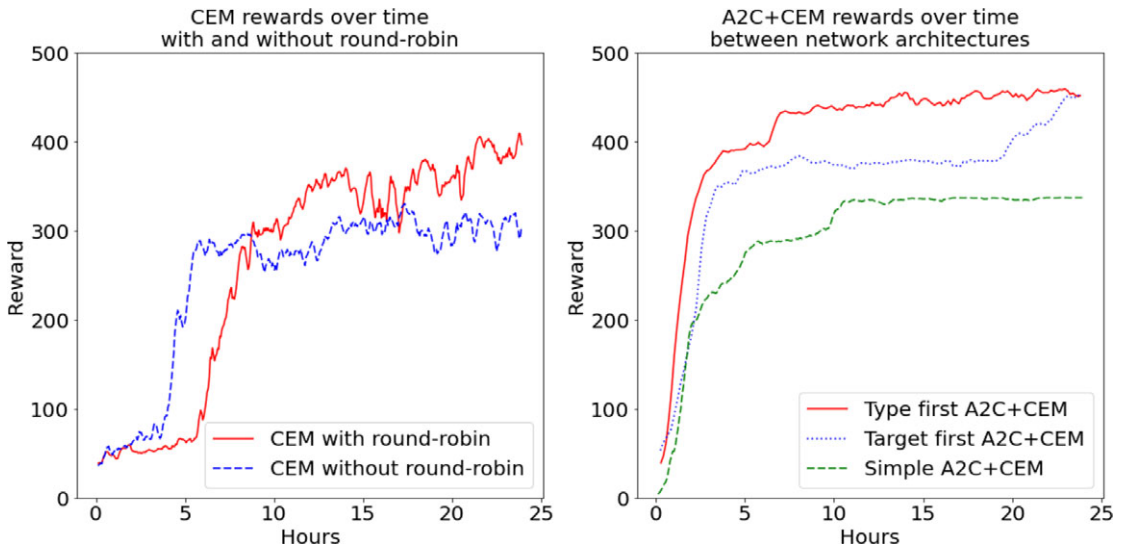
calculating the difference between the attacker's reward with no defender and the attacker's reward with the defender, we can obtain a rough heuristic for the defender's performance over time. This graph is shown in Figure 7 (Right). While significantly more noisy than the previous graph, it appears that A2C+FB-ES performs the best and with the most stability over time, with FB-ES, CEM and A2C+CEM also creating a significant difference in reward. A2C and (1+1)-ES are close to zero for much of the training period.

### 6.3.1 Coevolutionary round-robin training

The reward over time versus no defender for the attackers trained using CEM with and without the round-robin is shown in Figure 8 (Left). The CEM with round-robin appears to overtake the CEM without round-robin during the course of training and finish at a higher average reward. When testing each attacker against both defenders trained with CEM for 100 episodes each, the CEM with round-robin earned an average reward of 234 while the CEM without round-robin earned an average reward of 194.

### 6.3.2 Attacker neural network architectures

The reward over time versus no defender for the three attacker architectures trained using A2C+CEM are shown in Figure 8 (Right). The Type First and Target First models appear to train the attacker to a similar final average reward, although the regular Type First neural network outperforms the alternate Target First neural network for most of the training period. The Simple neural network architecture does not appear to perform as well as the other two. We also compare the performance of the three neural networks trained with A2C+CEM by executing 100 simulations between each pairing of three attackers and three defenders. The average rewards from these pairings are shown in Table 9. The Type First neural network performs best with an average reward of 295, closely followed by the Target First neural network with an average reward of 287. The Simple neural network had an average reward of 217.



**Figure 8.** (Left) Attacker's reward at various checkpoints during the 24 hours of training by CEM with and without the round-robin method. (Right) The Attacker's reward at various checkpoints during the 24 hours of training of A2C+CEM with the 'Type First', 'Target First', and 'Simple' neural network approaches described in Section 4.1. The rewards are smoothed by averaging consecutive terms.

## 7. Discussion and future work

This section provides a discussion about the results from the experimental comparisons (Section 7.1) and the limitations (Section 7.2) of our work, as well as future work (Section 7.3).

### 7.1 Experimental comparisons

Our results suggest that the AlphaZero algorithm, which incorporates planning, could be effective in training agent-based Attackers and Defenders both with deep reinforcement learning methods and with gradient-free methods. The different strategies used by the different trained models suggest the emergence of varying strategies, even in a simple SNAPT environment. More work is needed to determine whether the different training algorithms tend to converge to consistent but different strategies. Additionally, these results suggest that a training approach using multiple agents, such as a population-based method, might result in robust Attackers and Defenders that are successful against a variety of different strategies. In addition, a game-theoretic analysis to determine optimal strategies may also be possible.

In CyberBattleSim, the Attacker trained with CEM not training in round-robin style was strong. When round-robin competitions were compared, they outperformed the Attacker trained with CEM not training in round-robin style. Note the pairing system used in (1+1)-ES and fitness-based ES (FB-ES) is essentially this round-robin system with only two attackers and defenders. This first suggests the ES with sample size greater than one is advantageous and round-robin play offers even greater advantage.

In the investigation of A2C NN architectures, there was no significant performance difference between the Type First and Target First NNs trained with A2C+CEM. This suggests that this NN architecture design is invariant to a change in element selection. Both the Type First and Target First NNs outperformed the Simple NN architecture, suggesting the multi-stage architecture provides advantage to Attacker policy training in CyberBattleSim. Applying this multi-stage neural network design to agents in other cybersecurity ModSim platforms could provide additional evidence for this hypothesis. We also note that the multi-stage neural network architecture utilizes about 300 000 weights,

**Table 9.** The average rewards (rounded to the nearest integer) from 100 episodes for each pairing of trained attackers and defenders from the three attacker neural network architectures trained with A2C+CEM. The row gives the attacker and the column gives the defender. The average reward for each attacker is given in the Mean column. The largest value in each column is **bolded** to show which attacker performed best against a given defender

Attacker/Defender	Type First	Target First	Simple	Mean
Type First	<b>275</b>	<b>325</b>	284	<b>295</b>
Target First	<b>275</b>	290	<b>296</b>	287
Simple	220	218	212	217

**Table 10.** CyberBattleSim settings. The feature vector observations given to the attacker and defender. The Agent column gives whether the observation is given to the attacker or defender, the Type column gives whether the observation is global or device-specific, the Observation column gives the name of the observation (see Section C), and the Dimension column gives the dimension of the observation. For device-specific observations, the dimension is multiplied by 6 as there are 6 nodes in the network. The total size of the attacker and defender observations is given in the Total row in the observation column

Agent	Type	Observation	Dimension	
Attacker	Global	Credentials	5	
		Time	1	
	Device-specific	Discovered	1( $\times$ 6)	
		Installed	1( $\times$ 6)	
		Privilege level	1( $\times$ 6)	
		Attack count	1( $\times$ 6)	
		Discovered properties	14( $\times$ 6)	
			<i>Total</i>	114
Defender	Global	Time	1	
		Reimage count	1( $\times$ 6)	
	Device-specific	Running status	1( $\times$ 6)	
				<i>Total</i>

about 3 times fewer than the 900 000 weights used by the Simple neural network architecture. The extent of this complexity difference would be even greater in larger, more complex simulated environments.

We see that among the algorithms, the best-performing attackers appear to be those trained with deep reinforcement learning (DRL) either solely or in combination with ES. One somewhat surprising result is the apparent instability of the ES algorithms relative to DRL, contradicting the idea that ES may be a more stable alternative to DRL (Mnih *et al.*, 2015). Among Defenders, the algorithms training with ES solely or in combination with A2C performed best (except for (1+1)-ES). The algorithm only utilizing A2C performed quite poorly relative to the others, possibly due to the relative instability of the environment from the Defender's perspective. (1+1)-ES performed poorly in both Attacker and Defender settings, likely due to its aggressive update strategy (Lee *et al.*, 2020). Overall, the combined method A2C+FB-ES was among the best algorithms for training both Attackers and Defenders. However we are hesitant to make conclusions about the statistical significance of such results without more information on the stability of each algorithm across training runs.

Our results suggest that combining DRL and ES may provide more stable and robust algorithms for training agents in a competitive environment. We find that these algorithms are able to coevolve autonomous attackers to take actions in CyberBattleSim, and that introducing certain specializations such as round-robin evaluation and a multi-stage network architecture may improve performance.

### 7.2 Limitations and threats to validity

Based on the experimental setup which had each algorithm train only a single neural network, it is premature to draw conclusions about the statistical significance of the results without more information on the stability of each algorithm. While it appears the trained SNAPT models converged to different strategies, it is unclear to what extent this is due to the different training methodologies as opposed to random noise. Training multiple models with each algorithm could provide more insight into the stability of the models.

**Simple Network APT (SNAPT)** has many simplified assumptions. First, the use of a device (node) value is a simplifying assumption. For example the attribution of events is notoriously hard, both post-incident and during, and without knowledge of the attacker intent it is difficult for a defender to know the value of a device. In addition, it is difficult to ascribe devices intrinsic value to attackers, although information stored on a device, or the abilities provided by the device can provide quantifiable value.

The fixed number of moves make the environment susceptible to the horizon effect that is, an agent who can delay an opponent from reaching a device till past the max number of moves can avoid consequences on their reward. Moreover, the exploit probability is a simple approximation of the notion of a security upper bound, with a securing action not able to exceed the bound. Furthermore, the limited resources of defenders is a very general concept and cannot capture actions that would be obvious in actual networks. For example, if one device controls access to the network or contains the classified document server, the defender would allocate all their securing actions to that device.

**CyberBattleSim** The formulation of the reward as a zero-sum game is a simplified assumption to the complexity of cybersecurity realities. From a practical view the available actions for a defender in CyberBattleSim somewhat limits the practical use of discovered defender policies, as in practice compromised services are often patched, not only shut down or rebooted. Additionally, currently the agents only work in the particular network they are trained on, and their strategies would be ineffective in a different topology. From a more theoretical standpoint the CyberBattleSim environment has not been analyzed too rigorously, and it has over 1,000 available actions to the attacker which makes analysis quite complex.

### 7.3 Future work

In future work we will expand and improve the training methods used for better performance in larger, more complex simulations. The input dimension for both attacker and defender neural networks grows linearly with the number of devices in the simulated network, and introducing a recurrent neural network design similar to that used in natural language processing (Palangi *et al.*, 2016) may provide an effective design whose input dimension size remains small even in large networks. Such a design may also allow for the model to learn general strategies independent of network topology. We will also expand the scope of the algorithms used, including methodologies such as AI planning or population-based evolutionary algorithms which have found success in other domains, in addition to other coevolution setups (Popovici *et al.*, 2012). Moreover, the ability to train agents on multiple scenarios and to transfer policies to unseen scenarios will be investigated. The practical application of the policy search method could be improved by analyzing the discovered policies. Finally, the hyperparameters of the algorithms and the setup can be explored further, for example the training time until convergence.

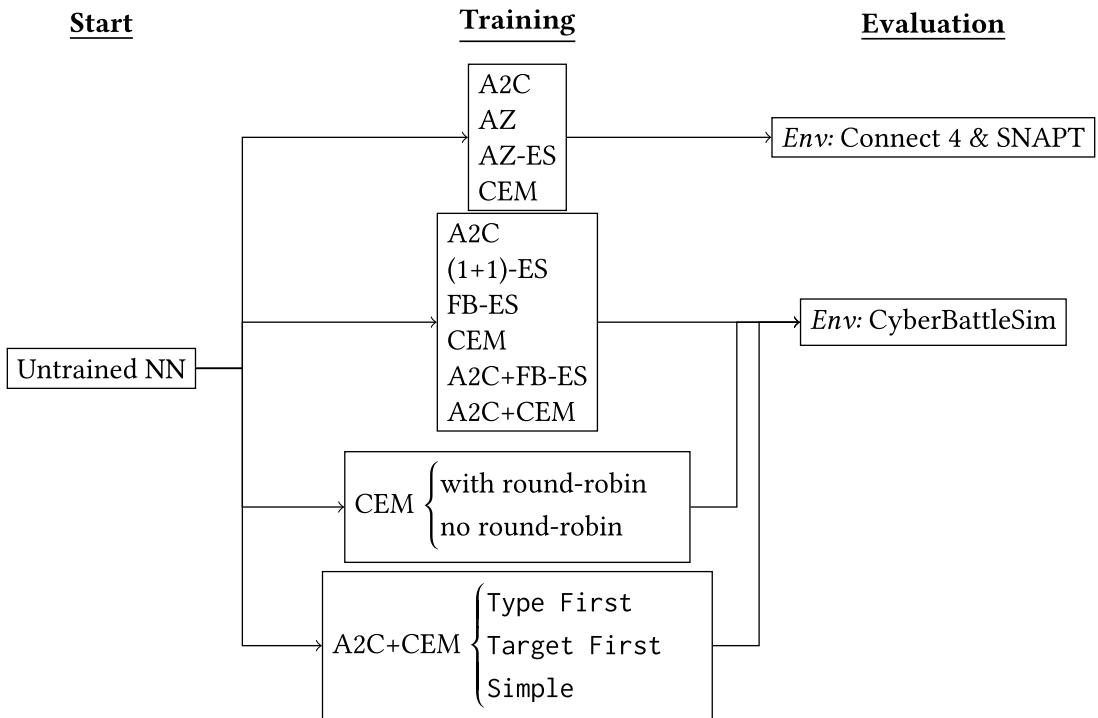
**Conflicts of interest.** The author(s) declare none.

## References

- Allis, L. V. 1988. A knowledge-based approach of connect-four. *Journal of the International Computer Games Association* **11**, 165.
- Applebaum, A., Miller, D., Strom, B., Korban, C. & Wolf, R. 2016. Intelligent, automated red team emulation. In *Proceedings of the 32nd Annual Conference on Computer Security Applications*, 363–373.
- Arulkumaran, K., Deisenroth, M. P., Brundage, M. & Bharath, A. A. 2017. Deep reinforcement learning: a brief survey. *IEEE Signal Processing Magazine* **34**(6), 26–38.
- Backes, M., Hoffmann, J., Künnemann, R., Speicher, P. & Steinmetz, M. 2017. Simulated penetration testing and mitigation analysis. ArXiv [abs/1705.05088](https://arxiv.org/abs/1705.05088).
- Baillie, C., Standen, M., Schwartz, J., Docking, M., Bowman, D. & Kim, J. 2020. Cyborg: An autonomous cyber operations research gym. ArXiv [abs/2002.10667](https://arxiv.org/abs/2002.10667).
- Bräm, T., Brunner, G., Richter, O. & Wattenhofer, R. 2020. Attentive multi-task deep reinforcement learning. In *Machine Learning and Knowledge Discovery in Databases*, Brefeld, U., Fromont, E., Hotho, A., Knobbe, A., Maathuis, M. & Robardet, C. (eds). Springer International Publishing, 134–149.
- Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J. & Zaremba, W. 2016. Openai gym. ArXiv [abs/1606.01540](https://arxiv.org/abs/1606.01540).
- Corporation, T. M. (n.d.a). Mitre att&ck. <https://attack.mitre.org>.
- Corporation, T. M. (n.d.b). Mitre engage. <https://engage.mitre.org>.
- Droste, S., Jansen, T. & Wegener, I. 2002. On the analysis of the (1+1) evolutionary algorithm. *Theoretical Computer Science* **276**(1), 51–81. <https://www.sciencedirect.com/science/article/pii/S0304397501001827>.
- Engström, V. & Lagerström, R. 2022. Two decades of cyberattack simulations: a systematic literature review. *Computers & Security*, 102681.
- Falco, G., Viswanathan, A., Caldera, C. & Shrobe, H. 2018. A master attack methodology for an ai-based automated attack planner for smart cities. *IEEE Access* **6**, 48360–48373.
- Goldberg, D. E. 1989. *Genetic Algorithms in Search, Optimization and Machine Learning*, 1st edition. Addison-Wesley Longman Publishing Co., Inc.
- Grondman, I., Buşoniu, L., Lopes, G. & Babuška, R. 2012. A survey of actor-critic reinforcement learning: standard and natural policy gradients. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* **42**, 1291–1307.
- Group, A. 2022. Adversarial agent-learning for cybersecurity. [https://github.com/ALFA-group/adversarial\\_agent\\_learning\\_for\\_cybersecurity](https://github.com/ALFA-group/adversarial_agent_learning_for_cybersecurity).
- Hansen, N. (2016). The CMA evolution strategy: a tutorial. ArXiv [abs/1604.00772](https://arxiv.org/abs/1604.00772).
- Harris, S. N. & Tauritz, D. R. 2021. *Competitive Coevolution for Defense and Security: Elo-Based Similar-Strength Opponent Sampling*. Association for Computing Machinery, 1898–1906. <https://doi.org/10.1145/3449726.3463193>.
- Huang, L. & Zhu, Q. 2020. A dynamic games approach to proactive defense strategies against advanced persistent threats in cyber-physical systems. *Computers & Security* **89**, 101660.
- Jiménez, S., De La Rosa, T., Fernández, S., Fernández, F. & Borrajo, D. 2012. A review of machine learning for automated planning. *The Knowledge Engineering Review* **27**(4), 433–467.
- Klijin, D. & Eiben, A. E. 2021. *A Coevolutionary Approach to Deep Multi-Agent Reinforcement Learning*. Association for Computing Machinery, 283–284. <https://doi.org/10.1145/3449726.3459576>.
- Lee, K., Lee, B.-U., Shin, U. & Kweon, I. S. 2020. An efficient asynchronous method for integrating evolutionary and gradient-based policy search. In *Advances in Neural Information Processing Systems*, Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M. F. & Lin, H. (eds), **33**. Curran Associates, Inc., 10124–10135. <https://proceedings.neurips.cc/paper/2020/file/731309c4bb223491a9f67eac5214fb2e-Paper.pdf>.
- Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N. M. O., Erez, T., Tassa, Y., Silver, D. & Wierstra, D. 2016. Continuous control with deep reinforcement learning. ArXiv [abs/1509.02971](https://arxiv.org/abs/1509.02971).
- Liu, J., Pérez-Liébana, D. & Lucas, S. M. 2016. Rolling horizon coevolutionary planning for two-player video games. In *2016 8th Computer Science and Electronic Engineering (CEECE)*, 174–179.
- Liu, L., Yasin Chouhan, A., Li, T., Fatima, R. & Wang, J. 2018. Improving software security awareness using a serious game. *IET Software* **13**, 159–169.
- Luh, R., Temper, M., Tjoa, S., Schrittwieser, S. & Janicke, H. 2019. Penquest: a gamified attacker/defender meta model for cyber security assessment and education. *Journal of Computer Virology and Hacking Techniques* **16**, 19–61.
- Macua, S. V., Davies, I., Tukiainen, A. & De Cote, E. M. 2021. Fully distributed actor-critic architecture for multitask deep reinforcement learning. *The Knowledge Engineering Review* **36**, e6.
- Metz, L., Ibarz, J., Jaitly, N. & Davidson, J. 2017. Discrete sequential prediction of continuous actions for deep RL. ArXiv [abs/1705.05035](https://arxiv.org/abs/1705.05035).
- Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M. A., Fidjeland, A., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S. & Hassabis, D. 2015. Human-level control through deep reinforcement learning. *Nature* **518**, 529–533.
- Molina-Markham, A., Winder, R. K. & Ridley, A. 2021. Network defense is not a game. ArXiv [abs/2104.10262](https://arxiv.org/abs/2104.10262).

- Nguyen, T. T. & Reddi, V. J. 2021. Deep reinforcement learning for cyber security. *IEEE Transactions on Neural Networks and Learning Systems*, 1–17.
- Olesen, T. V. A. N., Nguyen, D. T. T., Palm, R. B. & Risi, S. 2021. Evolutionary planning in latent space. In *Applications of Evolutionary Computation*, Castillo, P. A. & Jiménez Laredo, J. L. (eds). Springer International Publishing, 522–536.
- Palangi, H., Deng, L., Shen, Y., Gao, J., He, X., Chen, J., Song, X. & Ward, R. 2016. Deep sentence embedding using long short-term memory networks: analysis and application to information retrieval. *IEEE/ACM Transactions on Audio, Speech, and Language Processing* **24**, 694–707.
- Panait, L. & Luke, S. 2002. A comparison of two competitive fitness functions. In *Proceedings of the 4th Annual Conference on Genetic and Evolutionary Computation*, GECCO'02, Morgan Kaufmann Publishers Inc., 503–511.
- Partalas, I., Vrakas, D. & Vlahavas, I. 2012. Reinforcement learning and automated planning: a survey. In *Artificial Intelligence for Advanced Problem Solving Techniques*.
- Popovici, E., Bucci, A., Wiegand, R. P. & Jong, E. D. 2012. Coevolutionary principles. In *Handbook of Natural Computing*.
- Potter, M. A. & Jong, K. A. D. 2000. Cooperative coevolution: an architecture for evolving coadapted subcomponents. *Evolutionary Computation* **8**, 1–29.
- Pourchot, A. & Sigaud, O. 2018. CEM-RL: combining evolutionary and gradient-based methods for policy search. ArXiv [abs/1810.01222](https://arxiv.org/abs/1810.01222).
- Prince, M. H., McGehee, A. J. & Tauritz, D. R. 2021. Edm-drl: toward stable reinforcement learning through ensembled directed mutation. In *Applications of Evolutionary Computation*, Castillo, P. A. & Jiménez Laredo, J. L. (eds). Springer International Publishing, 275–290.
- Rechenberg, I. 1989. Evolution strategy: nature's way of optimization. In *Optimization: Methods and Applications, Possibilities and Limitations*, Bergmann, H. W. (ed.). Springer Berlin Heidelberg, 106–126.
- Reinstadler, B. 2021. *Ai Attack Planning for Emulated Networks*. Master's thesis, Massachusetts Institute of Technology.
- Rush, G., Tauritz, D. R. & Kent, A. D. 2015. 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*. Association for Computing Machinery, 859–866. <https://doi.org/10.1145/2739482.2768429>.
- Salimans, T., Ho, J., Chen, X. & Sutskever, I. 2017. Evolution strategies as a scalable alternative to reinforcement learning. ArXiv [abs/1703.03864](https://arxiv.org/abs/1703.03864).
- Schaul, T. & Schmidhuber, J. 2008. A scalable neural network architecture for board games. In *2008 IEEE Symposium on Computational Intelligence and Games, CIG 2008*, 357–364.
- Sigaud, O. & Stulp, F. 2019. Policy search in continuous action domains: an overview. *Neural Networks: The Official Journal of the International Neural Network Society* **113**, 28–40.
- Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., Lillicrap, T., Simonyan, K. & Hassabis, D. 2018. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science* **362**(6419), 1140–1144. <https://www.science.org/doi/abs/10.1126/science.aar6404>.
- Simione, L. & Nolfi, S. 2017. Achieving long-term progress in competitive co-evolution. In *2017 IEEE Symposium Series on Computational Intelligence (SSCI)*, 1–8.
- Team, M. D. R. 2021. Cyberbattlesim. <https://github.com/microsoft/cyberbattlesim>. Created by Christian Seifert, Michael Betser, William Blum, James Bono, Kate Farris, Emily Goren, Justin Grana, Kristian Holsheimer, Brandon Marken, Joshua Neil, Nicole Nichols, Jugal Parikh, Haoran Wei.
- The MITRE Corporation 2020. Caldera. <https://github.com/mitre/caldera>.
- Vinyals, O., Babuschkin, I., Czarnecki, W., Mathieu, M., Dudzik, A., Chung, J., Choi, D., Powell, R., Ewalds, T., Georgiev, P., Oh, J., Horgan, D., Kroiss, M., Danihelka, I., Huang, A., Sifre, L., Cai, T., Agapiou, J., Jaderberg, M., Vezhnevets, A., Leblond, R., Pohlen, T., Dalibard, V., Budden, D., Sulsky, Y., Molloy, J., Paine, T., Gulcehre, C., Wang, Z., Pfaff, T., Wu, Y., Ring, R., Yogatama, D., Wünsch, D., McKinney, K., Smith, O., Schaul, T., Lillicrap, T., Kavukcuoglu, K., Hassabis, D., Apps, C. & Silver, D. 2019. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature* **575**(7782), 350–354.
- Walter, E. C., Ferguson-Walter, K. J. & Ridley, A. D. 2021. Incorporating deception into cyberbattlesim for autonomous defense. ArXiv [abs/2108.13980](https://arxiv.org/abs/2108.13980).
- Yang, L.-X., Pengdeng, L., Zhang, Y., Yang, X., Xiang, Y. & Zhou, W. 2018. Effective repair strategy against advanced persistent threat: a differential game approach. *IEEE Transactions on Information Forensics and Security* **14**(7), 1713–1728.
- Zhu, Q. & Rass, S. 2018. On multi-phase and multi-stage game-theoretic modeling of advanced persistent threats. *IEEE Access* **6**, 13958–13971.

**Appendix A. Experiment Overview**



**Figure A.1.** An overview of our experimental setup. We start with an untrained network and apply one of the training algorithms described in Section 4.2. For Connect 4 and SNAPT we compare ES, DRL and MCTS. For CyberBattleSim we then compare the CEM algorithms with and without round-robin, A2C+CEM with the three different attacker network architectures, and all the algorithms using a Type First neural network.

**Appendix B. CyberBattleSim Training Iterations**

Table B.1 shows the number of training iterations for CyberBattleSim.

**Table B.1** The number of training iterations executed during the 24 hour training period for each algorithm for CyberBattleSim. For the combined methods (A2C + FB-ES and A2C + CEM) one iteration is counted as one step of both A2C and the ES method

Algorithm	Iterations
A2C	877
(1 + 1)-ES	1359
FB-ES	1425
CEM	1281
A2C + FB-ES	600
A2C + CEM	631



## Appendix C. CyberBattleSim RL Environment Specifications

### C.1. CyberBattleSim RL Input

Here we give the details on the feature vector input, observations, given to the attacker and defender. The information was chosen based on what information might be available to a real life attacker or defender in a network and is thus asymmetric. Some observations are specific to each device and some are global and relate to the network as a whole. Some pieces of information, such as the time, are expressed as integers, and some, such as whether a credential has been discovered, are boolean. The global information given to the attacker is which credentials it has discovered and the number of steps taken so far (the time). The device-specific information corresponds to whether or not a device has been discovered, whether or not malware has been installed on the device, the attacker's privilege level on the device, the number of times the device has been attacked, and the discovered properties of the device. The only global property given to the defender is the time. The device-specific information is the number of times each device has been reimaged and whether each device is running or not. A summary of this information can be found in Table 10.

### C.2. CyberBattleSim Environment

From Group (2022) Chain pattern environment. In the environment the local vulnerabilities are: Scan bash history, Scan explorer recent files, sudo attempt, Crack keep PassX and Crack keep Pass. Remote vulnerabilities are probe Linux or Windows. Connections are: HTTPs, GIT, SSH, RDP, PING, MySQL, SSH-key and su.

**Table C.1** The standard deviations (rounded to the nearest integer) from 100 episodes in each attacker and defender pairing. The column gives the defender and the row gives the attacker. We include an untrained attacker and defender which randomly select each action as a control as well as an environment with no defender. The averages of each row and column are given at the end.

Attacker/Defender	A2C	(1+1)-ES	FB-ES	CEM	A2C+ FB-ES	A2C+ CEM	Random	None	Mean
A2C	80	81	93	99	90	84	100	21	81
(1+1)-ES	24	23	19	26	18	22	16	36	23
FB-ES	117	85	78	81	92	104	103	116	97
CEM	128	129	107	114	92	105	106	131	114
A2C+FB-ES	89	78	93	90	92	94	90	63	86
A2C+CEM	83	71	88	93	90	94	92	38	81
Random	30	38	27	31	27	28	34	25	30
Mean	79	72	72	76	72	76	77	62	73