Analyzing Multi-Agent Reinforcement Learning and Coevolution in Cybersecurity

Matthew J. Turner
mjturner@mit.edu
MIT
Cambridge, MA, USA

Erik Hemberg
hembergerik@csail.mit.edu
MIT
Cambridge, MA, USA

Una-May O'Reilly
unamay@csail.mit.edu
MIT
Cambridge, MA, USA

ABSTRACT

Cybersecurity simulations can offer deep insights into the behavior of agents in the battle to secure computer systems. We build on existing work modeling the competition between an attacker and defender on a network architecture in a zero-sum game using a graph database linking cybersecurity attack patterns, vulnerabilities, and software. We apply coevolution to this challenging environment, and in a novel modeling approach for this problem, interpret each population as a distribution over fixed strategies to form a mixed strategy Nash equilibrium. We compare the results to solutions generated by multi-agent reinforcement learning and show that evolutionary methods demonstrate a considerable degree of robustness to parameter misspecification in this environment.

CCS CONCEPTS

• **Security and privacy** → **Intrusion/anomaly detection and malware mitigation**; • **Computing methodologies** → **Multi-agent systems**; **Game tree search**; **Markov decision processes**; **Genetic algorithms**; • **Networks** → Network simulations.

KEYWORDS

cybersecurity, coevolution, evolutionary algorithms, machine learning, Nash equilibrium

ACM Reference Format:
Matthew J. Turner, Erik Hemberg, and Una-May O'Reilly. 2022. Analyzing Multi-Agent Reinforcement Learning and Coevolution in Cybersecurity. In *Genetic and Evolutionary Computation Conference (GECCO '22), July 9–13, 2022, Boston, MA, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3512290.3528844

1 INTRODUCTION

Cyberattacks pose a material and substantial threat to both governments and industry. The Solarigate attack of 2020 exposed the data of thousands of individuals, corporations, and even the U.S. government to malicious hackers[16]. The CryptoWall ransomware attack was able to extort over $18 million in its first year of operation[4]. The WannaCry ransomware attack, developed by North Korea, indiscriminately affected more than 80 hospital systems worldwide, and is part of a growing trend of attacks that aim for maximum damage against a host of targets[29]. Given the immense risks to both safety and property, it is in the public interest to better understand the behavior of attackers in order to develop effective defensive strategies.

In mitigating and preventing cyberattacks, network administrators have limited options. They can engage in vulnerability scanning, or simply enumerating all potential vulnerabilities on a network[1]. However, with one database showing an average of 255 known vulnerabilities per software configuration, this approach does little to identify the most pressing vulnerabilities to mitigate[12]. Penetration testing involves dedicating a team to discover weaknesses by conducting, for example, capture-the-flag exercises on a network. Although important, this form of testing costs between $10,000 and $100,000, is often only conducted biannually, and can result in network downtime if an attack is successful[1].

Breach and simulation (BAS) tools attempt to provide an automated solution that addresses the drawbacks of each of these methods. BAS tools benefit from an understanding of real attacker behavior, providing insight into which vulnerabilities are most pressing. Since attacks are simulated, these benefits can be realized without the cost and risk of penetration testing. BAS tools do not eliminate the need for penetration testing completely, as they rely on existing data and are not robust to *zero-day* or new attacks[1]. However, they offer a necessary intermediate providing an automated, context-aware prioritization for vulnerability mitigation. These tools can be seen as a form of *cyber hunting*, or proactively anticipating and mitigating attacks based on a model of the attacker's behavior[18].

In this paper, we use a BAS framework to model and simulate a game between attackers and defenders on a network. We analyze two potential models for agents and show that coevolution results in a strong and diverse solution set while reducing computational overhead. Simulations such as this could lead to a better understanding of attacker behavior and the development of more effective and efficient mitigations.

In line with prior research, we model a cyberattack as a zero-sum game played between an attacker and defender on a network environment[27]. Specifically, the attacker selects one or more *attack patterns* to leverage against the defender's network. Simultaneously, the defender selects software(s) to *patch*, or upgrade, to the next software iteration. The reward for the attacker is computed as the sum of the risk scores of the product configurations affected by their attack; the defender's reward is the negation of this. We model these as simultaneous actions, for, in a real-world setting, an attacker may not know the complete network structure before launching an attack, and the defender certainly cannot know an attacker's strategy, tactics, or techniques a priori. Furthermore, no

GECCO '22, July 9–13, 2022, Boston, MA, USA

ACM ISBN 978-1-4503-9237-2/22/07.
https://doi.org/10.1145/3512290.3528844

internal state is revealed after either action, since a failed attack does not uncover new insights into the network structure. Unlike other BAS tools, which primarily focus on modeling attacker behavior, bringing in the defender can help us better understand zero-day attacks, a traditional weakness of BAS[1].

We note the challenging nature of these environments, which can be viewed from several angles, including solving for Nash equilibria, combinatorial optimization, Markov decision processes (MDPs) with adversarial rewards, or reinforcement learning (RL). In any case, the problem is NP-hard, and any solution will be approximate. With the infeasibility of finding an optimal solution, we compare two different approaches: *multi-agent reinforcement learning* (MARL), or a competition between two RL agents, and *coevolution*, an evolutionary competition between two populations. Figure 1 contains an overview of the system. In our implementation, we use grammatical evolution (GE) as the evolutionary search mechanism[22].

Past research on this problem formulation has focused on applying coevolution as a means to solve for specific Nash equilibrium (NE) solutions among a population, and can be extended to population members that represent *mixed strategy NE*, or NE composed of a distribution over fixed strategies[31]. We take a slightly different approach and make a connection between coevolution and RL in the process. Instead of looking for individual equilibrium solutions among the population, we model the populations itself as a distribution over fixed strategies, so the group as a whole becomes a mixed strategy NE. Under this model, the mean population reward has a beautiful interpretation as the expected reward obtained by selecting a strategy to play uniformly at random from the population. Rather than hand-picking a Nash equilibrium from the population, we let the evolutionary dynamics, which seek to maximize the average population reward, select the distribution of strategies for us. The population as a whole can now be compared to an RL agent which selects actions based on some learned probability distribution.

We primarily focus on mixed strategy equilibria because our knowledge of the environment tells us that, intuitively, there exist very few stable fixed strategy NE. To see why this is the case, take, for example, an attacker who threatens a certain product configuration with an attack pattern. The best response of the defender is to patch that product, thus mitigating the attack and resulting in a lower reward for the attacker. In this scenario, it may become advantageous for the attacker to randomize their actions according to some probability distribution over attack patterns. These dynamics are also seen in practice[21]. We also note that the set of mixed strategy NE is a superset of fixed strategy NE, so we gain, rather than lose, generalizability through this assumption.

Through this simulation, we hope to answer the following research questions. *How do the solutions generated using a mixed strategy coevolutionary model compare with a standard RL algorithm?* Since we are operating in an *online learning* setting in which agents learn by interacting directly with the environment, the tradeoff between exploration of new strategies and exploitation of existing ones bearing high reward is key. As a subquestion, we add: *How do exploration/exploitation decisions vary between the RL and GE algorithms?* In answering these questions, we offer the following contributions:

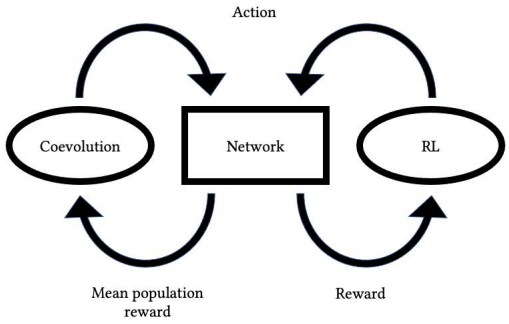

Figure 1: A representation of the system. Both coevolutionary and multi-agent RL systems interact with the environment (a network) and learn through observed rewards.

- Proposing a novel approach to modeling mixed strategy Nash equilibria as a coevolutionary population in cybersecurity simulations
- Comparing the results of such a coevolution to reinforcement learning agents competing in the same cybersecurity environment

We proceed as follows. Section 2 details the background necessary to understand our contribution. Section 3 describes our method and evaluation. Section 4 presents our results. Section 5 analyzes these results, and Section 6 proposes several avenues for future research.

2 BACKGROUND

A variety of approaches exist to simulating cyberattacks. To play optimally, agents must find best response functions which lead to a Nash equilibrium. We begin by highlighting the general landscape of cybersecurity simulations, define Nash equilibria on a game, and introduce both algorithms for analysis.

2.1 Cybersecurity Simulation

Much research has been done to simulate the competition between attackers and defenders in cyber warfare. These methods vary in their algorithmic variant, participant behaviors, environment, and objective function, but the primary differentiator is the choice of action space, or range of possible actions an agent may take.

The breadth of action spaces utilized in prior research can be placed on a spectrum from very concrete actions on specific systems to more general, network-wide abstractions. One class of simulations has focused on operating on particular code bases and could prove useful in training models for threat detection and monitoring. Coev-Malware and ArmsRace involved injecting lines of malicious code while a vendor aimed for early detection of the security bug[8, 26]. In the middle of the spectrum lies RIVALS-DDOS, which simulates a DDOS environment while not actually running any malware itself[23]. Taking a broader view, a recent development, EvoAPT, creates a competition using abstract attack patterns on a simulated network[27]. We describe more details of the EvoAPT framework in Section 2.4.

2.2 Nash Equilibria

A *Nash equilbrium* occurs when two players are playing their best response to the strategies of their opponent(s), and no player can deviate to achieve a higher payoff. Let (S, r) be a game of n players, where $S = (S_1 \dots S_n)$ contains the strategy set S_i for each player and $r : x \rightarrow \mathbb{R}$ is the reward function. Each player selects a strategy $x_i \in S_i$ from their strategy set. For simplicity, we denote x_{-i} as the selected strategies of all other players except player i. A set of chosen strategies x^* is a Nash equilbirium if

$$\forall i, x_i \in S_i, r(x_i, x_{-i}) \leq r(x_i^*, x_{-i}^*)$$

In other words, no player can deviate from their equilibrium strategy x_i^* and receive a higher payoff. In the context of our simulation, this corresponds to selecting a distribution of attack patterns for which no further improvement in reward can be made, given the defender's mitigation strategy, and vice versa. The key is finding an intersection of each player's best response function, such that no player would deviate to achieve a higher payoff.

To find a player's best response function exactly, one could draw a game tree, enumerate all possible branches, and find the optimal action via backwards induction. The problem then becomes one of performing brute force search over the entire state space. In our case, this amounts to search over all combinations of attack patterns and software patches. Heuristics such as alpha-beta pruning improve the search time by proactively pruning branches, but solving still requires search and full knowledge of the opponent's actions[11]. This approach becomes infeasible when the number of possible states or actions becomes too large.

2.3 Multi-Agent Reinforcement Learning

Reinforcement learning consists of one or more agents who interact with an environment consisting of a set of states $\mathcal{X}$, actions $\mathcal{A}$, and a reward function $r : \mathcal{X}x\mathcal{A} \rightarrow \mathbb{R}$. The goal of the agent is to maximize its reward. RL environments typically rely on the Markov assumption that the environment is essentially memoryless, and that the present state and reward only depends on the previous state and action. Reinforcement learning agents are often evaluated based on their *regret R*, which is defined as

$$R = \sum_{t=1}^{T} r_t^{\pi_t} - \sum_{t=1}^{T} r_t^{\pi^*}$$

where the agent acts in $t = 1 \dots T$ rounds, $r_t^{\pi_t}$ is the average reward achieved in round t from following policy π_t, and $r_t^{\pi^*}$ is the average reward achieved in round t from following the optimal policy π^*. Intuitively, regret is the difference in expected rewards between following the learned policy and some optimal policy π^*.

Recent research has focused on multi-agent reinforcement learning (MARL), including problems in which two or more agents either cooperatively or competitively act in an environment[30]. A special case arises when agents neither observe their opponents' actions nor their impact on the agent's state. In this case, the problem of finding a best response can be reduced to solving a Markov decision process with bandit feedback (where an agent only receives information about the reward for actions taken) and adversarial rewards. A recent algorithm was introduced which led to the lowest known regret bound in this challenging environment, but their work has not been implemented due to the need to solve a global optimization problem using Online Mirror Descent after every environmental interaction[13]. A recent publication introduced dilated bonuses to maintain optimistic reward estimators in the face of adversarial rewards, a key step to ensuring that agents continue to explore new actions. This algorithm, however, requires the explicit enumeration of the entire state and action space unless specific assumptions are made, something that may be infeasible for the large action spaces we are modeling[15]. For example, our task of selecting three attack patterns simultaneously leads to over 100 million possible actions–more rows than we want to store in a table.

An alternative to directly constructing this table is to use a neural net to approximate each entry, in a field known as deep reinforcement learning. Some work has been done on training robust deep reinforcement learning agents, but it has mainly focused on bounded (and not adversarial) changes in the reward function, so using this approach currently means setting aside theoretical guarantees on the regret bound[20].

2.4 Coevolution

An alternative is evolution strategies, which have been used to solve several traditional RL problems, including Atari and humanoid walking[25]. Two evolutionary populations can be adversarially paired in coevolution, and this has been used to solve for optimization problems in which the environment can change, just like our adversarial rewards[14]. Coevolution presents several key advantages over RL models, including eliminating the need for gradient updates and value function estimation[25]. Evolutionary methods are much more flexible and pose few restrictions on the types of environments that may be used.

EvoAPT applied coevolutionary algorithms to a competitive cybersecurity environment[27]. The EvoAPT framework is noteworthy for its method of quantifying the objective function for optimization. While other simulations have focused on detection time, accuracy, and disruption as measures of success, the EvoAPT utilizes the Common Vulnerability Scoring System (CVSS) scores, an extension of the Common Vulnerability Enumeration (CVE) database, as a measure of the success of a particular attack[3, 5, 27]. Linking several necessary databases, including Common Attack Pattern Enumeration and Classification (CAPEC) attack patterns and Common Platform Enumeration (CPE) software configuration identifiers is possible through a graph database called BRON[2, 6, 12]. Using CVSS scores is advantageous as it allows quantifying the impact of attacks by their risk level, thus meeting the goal of BAS tools to rank vulnerability risk based on anticipated attacker behavior. We borrow the reward function and simultaneous attack and defense environment from the EvoAPT framework.

Prior work on finding Nash equilibria in cyber simulations has focused on finding strategies that are evolutionarily stable and therefore correspond to Nash equilibria[31]. The results from such attempts were mixed, and did not show a clear advantage to using coevolutionary algorithms over other techniques for finding Nash equilibria. To our knowledge, no work has been done treating the coevolutionary populations as a mixed strategy in the cybersecurity simulation domain.

3 METHOD

We begin by detailing the modeling assumptions of the environment. We then describe each algorithm in turn and the modifications to the environment necessary for its function.

We define our simulation environment as follows:

- The game proceeds in rounds. In a single round, both an attack and a defense are made simultaneously, independently, and without knowledge of the other player's actions. This models a real-world scenario, where an attacker may not know the network structure before launching an attack, and the defender certainly cannot know which attack patterns will be selected. At the end of the round, both players see their own reward, but do not learn the other player's actions.
- An attack is the selection of three attack patterns from the CAPEC repository[6]. There may be 0, 1, or more than 1 software configurations affected by each attack pattern. Currently, there are 546 CAPECs resulting in 162,771,336 potential combinations.
- A defense is the selection of three software configurations, as identified by CPE, to patch on the network[2]. A patch increments every instance of the particular software to the next available version, similar to how a network administrator would roll out a systemwide update. There is no guarantee the upgrade will fix any security risks to particular attack patterns, and may introduce new ones. In our simulations, we used a network with 20 unique software configurations for a total of 8,000 patch combinations.
- The environment is a model of an enterprise network. It contains a map between each software configuration and the number of occurrences on the network. We used a single network to keep all experiments consistent.
- The reward is the sum of the CVSS scores for every software configuration on the network affected by the selected attack patterns. This is a minimax game, so the attacker receives a positive reward with increased risk, while the defender receives the negation of that reward. The defender's max reward is 0, while the attacker's max reward depends on the network.

In this environment, a Nash equilibrium strategy involves a distribution over attack patterns (patches) that results in the highest (lowest) average reward, given the opponent's strategy.

We now describe the application of this game to each learning algorithm.

3.1 Coevolution

Due to the versatility of coevolution, the process of encoding this game in a grammar is relatively straightforward. The grammar used for attacker-defender interactions is found in Listing 1. The chosen hyperparameter combinations for evaluation are displayed in Table 2. *GE baseline* and *GE elite* are based on default arguments for the coevolution[10]. *GE low mutation* and *GE low mutation, elite* are based upon prior research[27]. An interpretation of these hyperparameters may be found in Table 1.

In a novel modeling approach, we treat each population as an agent (either attacker or defender) and its members as strategies in a mixed strategy NE. To understand the motivation for this

Listing 1: Coevolution grammar. The attacker selects three attack patterns (denoted ap in this listing) while the defender selects three application identifiers to patch. A single round involves input from both <attack> and <defense>.

```
\# ATTACK GRAMMAR
<attack> ::= <ap>, <ap>, <ap>
<ap> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 ...

\# DEFENSE GRAMMAR
<defense> ::= <software>, <software>, <software>
<software> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | ...
```

Table 1: Interpretation of coevolutionary hyperparameters. A high mutation probability resulted in higher turnover in the population, leading to exploration. An increase in crossover probability resulted in greater recombination among solutions in the population, leading to faster adoption of strong solutions. A larger elite size created more consistency in the population over multiple generations, thus dampening oscillations in the population reward.

Hyperparameter	Interpretation
Mutation probability	Exploration parameter
Crossover probability	Exploitation parameter
Elite size	Convergence pressure

model, we recall that a mixed strategy is simply a distribution over fixed or deterministic strategies. Assume we have a mixed strategy we would like to play by sampling elements from a population of strategies. We could play this mixed strategy by sampling each strategy from the population according to our desired distribution. If we wanted to store fewer strategies, we could approximate the full distribution by limiting the size of the population to some upper bound k by taking the top k strategies with the highest probability of play.

Now assume that we can only sample from our population with uniform probability. If we had a near-infinite population size ($k = \infty$), we could approximate any continuous distribution, or mixed strategy, by allowing duplicate strategies in our population and assigning slots in the population according to the desired probability distribution. As k decreases, the approximation becomes more rough, but we can still sample uniformly from the population and get an approximation for our mixed strategy distribution. Just as each representative in a democratic government stands for a portion of the people, each representative in the population stands for a portion of the strategies in our distribution.

This formulation has two advantages. First, there is no need to hand-pick strategies from the population, as the evolutionary dynamics allocate members of the population in order to maximize individual fitness, which in turn maximizes the collective fitness. In other words, the dynamics are already present to select for members of the population which approximate the optimal mixed strategy NE. Secondly, the mean population reward obtained from the coevolutionary algorithm is exactly equal to the expected reward from

Table 2: Coevolution hyperparameters. All experiments were conducted with a population size of 10 and tournament size of 2.

Algorithm Name	Mutation probability	Crossover probability	Elite size
GE baseline	0.5	0.25	0
GE elite	0.5	0.25	4
GE low mutation	0.1	0.8	0
GE low mutation, elite	0.1	0.8	4

playing a round with a single randomly selected strategy from the population. This means we have an exact metric to measure performance, and a fair way to compare a coevolutionary population to a single RL agent operating under its own learned distribution over actions.

For all algorithms we keep the tournament size constant at 2 in line with previous research[31]. For the population size, we had two warring factors that had to be balanced. To model the entire continuous space of mixed strategy Nash equilibria, we would need an infinite population size. However, the most expensive aspect of the coevolution is the $O(n^2)$ fitness evaluations that must be conducted, where n is the population size. We selected a population size of 10 as the largest possible while maintaining reasonable runtimes in our implementation.

3.2 Reinforcement Learning

For the reinforcement learning case, we had to be a bit more careful in our design. Standard RL algorithms, such as those present in the StableBaselines3 package which we used, require a specific environmental API, which does not allow the opponent's actions to be taken in parallel[24]. In fact, since each agent is only taking a single action and observes a reward, without seeing the action of its opponent, our environment is much more like a multi-armed bandit problem with adversarial rewards than a standard MARL environment. With this in mind, we model each agent as acting in an MDP with adversarial rewards. We developed two separate OpenAI gym environments–one for each agent–and updated each with changes to the opponent's strategy[7].

We also altered the action spaces of our environment in order to ensure compatibility with the RL algorithms, as RL algorithms generally perform poorly when presented with large discrete action spaces[9]. To handle the large number of attack pattern combinations, we converted the action space for the selection of attack pattern triplets into a continuous action space represented by a cube of edge length 2 centered at the origin. The agent's action was represented by the selection of a coordinate within this cube. To convert this point into three attack patterns, we simply partitioned each axis into equal-sized intervals, with each interval representing a single attack pattern. Thus, we could determine the attack pattern $c(x_i)$ for coordinate x_i of dimension i by taking

$$c(x_i) = CAPECS\left[\left\lfloor \frac{x_i + 1}{2} len(CAPECS) \right\rfloor\right]$$

where $CAPECS$ is some zero-indexed list of CAPECs, len denotes the length function, and $x[i]$ denotes the ith element of list x. Note that this is equivalent to partitioning the size-2 cube into $len(CAPECS)^3$ smaller cubes of side length $2/len(CAPECS)$, where each represents a unique combination of 3 CAPECs. We recognize that this structure could imply some sort of spatial relation between CAPECs that may adversely affect the model's learning. On the whole, however, we hypothesize that this spatial relation was to the algorithm's advantage, since CAPEC identifiers are often grouped by similarity in the CAPEC taxonomy[6].

We encountered several design decisions when selecting an RL algorithm for comparison. We selected the A2C model due to its fast wall clock running time, versatility in state and action space representations, and well-tested implementation in the Stable Baselines3 package[19, 24]. Due to the NP-completeness of the problem, it would be impossible to compare the quality of the RL solutions relative to the unknown set of Nash equilibria. Since our goal was primarily to compare our grammatical coevolution approach to some other baseline, we simply used the default learning rate of 0.0007 provided in the StableBaselines3 package[24].

Next, we needed to derive our own adversarial training algorithm. We trained both the attack and defense agents in an alternating schedule using a single environmental sample (ie a batch size of 1) for each agent per episode. Each agent would learn using an environmental sample, update its internal model, make a prediction, and update their opponent's reward function based on their adversarially chosen action. Our variation stands in contrast to some other adversarial machine learning methods, such as Generative Adversarial Networks (GANs), in which generally one agent, say the generator, is trained using a much larger batch size before alternating to train the discriminator, and vice versa. This is usually done for performance reasons, since training a GAN using a batch size of 1 would waste too much time in performing gradient updates. The drawback to such time-saving measures lies in issues such as mode collapse, where one agent can quickly evolve and fail to provide useful gradients to its opponent[17]. We anticipated similar issues with larger batch sizes in our case, including agents that could specialize in thwarting a single attack from their opponent and catastrophically forget a more diverse strategy set. Our training algorithm is presented in Algorithm 1.

3.3 Comparison

Our overarching research question as stated in Section 1 was to compare both coevolution and RL approaches in solving our cybersecurity problem. In order to answer this question, we needed a basis to quantify the amount of learning each algorithm had performed to accurately compare the results. For this purpose, we chose to evaluate each algorithm based on the number of internal parameter updates made. In the case of coevolution, this would correspond to 10 (the population size) updates per generation for both the attacker and defender; for RL, one update is accumulated every

Algorithm 1 MARL Training Algorithm. The algorithm takes in the attacker and defender reinforcement learning agents and environments, and executes each over n episodes with a batch size of 1. This version stores cumulative reward; modifications can be made to log individual rewards

Input:
a: Agents **e**: Environments n: Episodes, $n > 0$ N: Network
Output: r cumulative reward

$r \leftarrow 0$ // Initialize reward
for $i \in [1, \ldots, n]$ **do** // for each episode
  learn($\mathbf{a}_a, \mathbf{e}_a$) // Learn attack agent on environment for 1 step using one environmental sample
  $(c_1, c_2, c_3), r_a \leftarrow p(\mathbf{a}_a)$ // Predict CAPEC attack patterns from agent and get reward
  $r \leftarrow r + r_a$ // Update cumulative reward
  e_d.set_capecs(c_1, c_2, c_3)) // Update defense environment reward function based on CAPEC attack patterns, i.e adversarial reward choice
  $\mathbf{a}_a \leftarrow$reset($\mathbf{a}_a$) // Reset attack environment
  learn($\mathbf{a}_d, \mathbf{e}_d$) // Learn defense agent on environment for 1 step using 1 environmental sample
  $(p_1, p_2, p_3), r_d, N_p \leftarrow p(\mathbf{a}_a, N)$ // Predict patches, modify network from agent, and get reward
  $r \leftarrow r - r_d$ // Update cumulative reward. Minimax, so r_a is negative of r_a.
  e_a.set_network(N_p) // Update attack environment reward function based on modified network, i.e adversarial reward choice
  $\mathbf{a}_d \leftarrow$ reset($\mathbf{a}_d$) // Reset defense environment
end for

return r // Cumulative reward

episode, since our batch size was 1. We therefore grouped every 10 RL episodes into a *generation*, where we use the term loosely here as a matter of convenience to compare to the coevolution. We note that other heuristics for comparison could have been used, including number of environment calls and CPU time.

4 RESULTS

We expected both the reinforcement learning and coevolutionary algorithms to achieve rewards on the same order of magnitude, but thought that the RL algorithm would achieve a modestly higher cumulative reward for the attacker. This is because RL algorithms are tuned to minimize regret, or, in other words, to optimize the decision between exploration and exploitation. In our model, the attacker had the largest state space to search, so we would expect the benefits of RL to be more pronounced in this challenging case, leading to a higher cumulative reward. We also expected more stability in the RL rewards, as coevolutionary population dynamics can vary significantly between iterations.

Our hypothesis about oscillations in the coevolutionary reward were correct. In some cases, the reward oscillated wildly during training, as shown in Figure 2. This is because the population size for coevolution is somewhat analogous to a batch size for RL, since the entire population of attack strategies is updated before an update to the defense strategies is made, and vice versa. Several hyperparameters could be tuned to minimize such oscillations. We found the best results with a low mutation and high crossover rate in line with prior research[31]. We believe this combination performed well since the low mutation probability preserved strong solutions, while the high crossover probability leveled the playing field between individual solutions in the population, in line with our model of the population as a probability distribution over individual strategies. To a lesser extent, a positive elite size helped convergence by maintaining consistency in the population across generation.

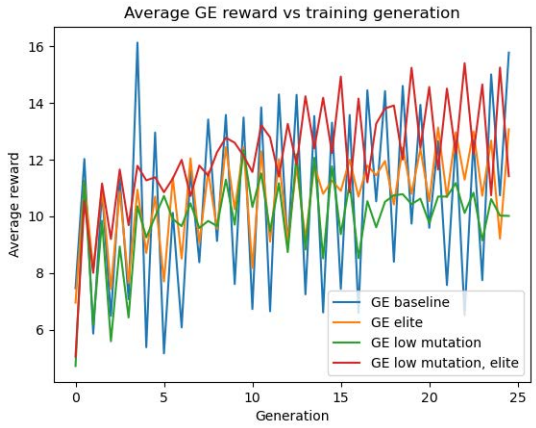

Figure 2: Comparing the average reward as a function of training generation for the coevolutionary algorithm under four sets of hyperparameters. A low mutation rate seemed to offer the strongest dampening on the reward. Hyperparameter values for each variant may be found in Table 2.

As an aside, we note that although the choice in coevolutionary hyperparameters had a large impact on the strength of the reward oscillations, in most cases, it had little impact on the cumulative reward achieved, as shown in Figure 3. This suggests that coevolution is robust to some hyperparameter misspecification[28]. This could prove to be a huge benefit in training, due to the complexities involved in training adversarial machine learning methods in general and MARL methods in particular[30].

Contrary to our expectations, the coevolutionary and RL algorithms seemed to converge to very different equilibria, as shown

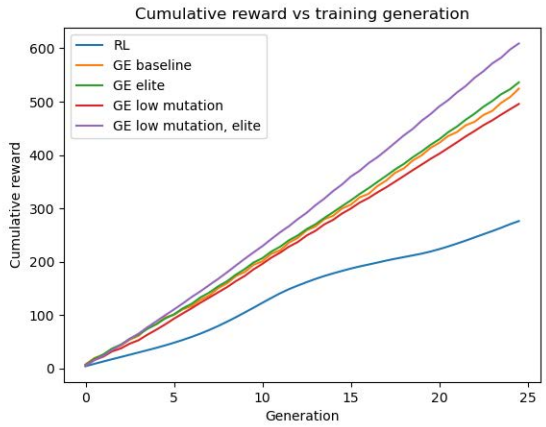

Figure 3: Cumulative reward as a function of training generation. Although the choice in hyperparameters had a big impact on the strength of the reward oscillations for coevolution, in most cases, it had little impact on the cumulative reward. Note that, although the optimal policy is not known, a higher cumulative reward means a lower regret for the attacker—hence, all GE variants were able to achieve a better attacker regret bound than the RL variant under these parameters.

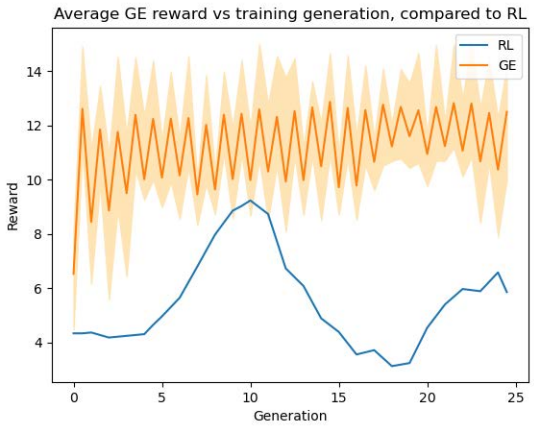

Figure 4: RL reward compared to average *GE low mutation* reward over 5 runs with min and max ranges shaded. In most cases, the average RL reward does not meet the minimum GE reward.

in Figure 4. Specifically, the attacker in general performed much better through coevolution than the RL variant. We break down this difference into several potential causes. The first could be related to our necessary adjustment to the attacker's action space in the RL variant. Because the algorithm had no clear delimiter between combinations in the continuous space, it may have wasted time exploring small perturbations that resulted in the same attack patterns

Table 3: Number of unique CAPEC attack patterns explored by algorithm. The RL algorithm explored more than all but the GE baseline.

Algorithm	Number unique CAPECs explored
GE baseline	282
GE elite	192
GE low mutation	112
GE low mutation, elite	82
RL	248

being selected. If this were the case, however, we would expect the RL algorithm to have explored fewer unique attack patterns, not more as shown in Table 3 A more likely explanation stems from the fact that the RL algorithm relies on optimistic estimators for the rewards of particular actions in order to regulate the level of exploration. As shown in Table 3, the RL algorithm explored more unique attack patterns than all but the baseline coevolutionary algorithm. It seems that the adversarial rewards place a downward bias on the optimistic reward estimators, pushing the RL algorithm toward excessive exploration.

In many cases, the RL reward lay below even the minimum reward range over multiple runs of the GE algorithm, further proving that these are indeed very different equilibria. Due to the nature of the problem and the possibility of multiple NE, it would be impossible for us to make a value judgment as to which solutions was "better." In this specific scenario, and under the specific hyperparameters we used, however, we were able to find several certificates of non-optimality demonstrating that the RL agents had not converged to some stable equilibrium. If the RL attacker were performing optimally under a mixed strategy, we would expect its reward to sometimes lie above the minimum GE realized reward. This is because we would expect the attacker to occasionally select an attack pattern combination for which the defender does not yet have a good defense, leading to a high attack reward. This is not the case, and suggests that the RL algorithm did not achieve an optimal ratio between exploration and exploitation under these parameters. We also found many iterations in which the RL defense agent patched the same software multiple times in a single action, a policy that could not lead to further improvement in reward. The defender could at least diversify its portfolio of mitigations to safeguard against new attacker exploration. It seems that excessive exploration led to suboptimal performance for the RL algorithm over training. The cumulative effect of this was a lower reward for the RL attacker, as shown in Figure 3.

5 DISCUSSION

Our goal was to compare the results of both an RL and a coevolutionary algorithm in solving for Nash equilibria in a competitive cybersecurity environment. Finding an optimal selection of strategies can be framed as a game tree search or combinatorial optimization problem, which is NP-hard, and infeasible for all but the smallest of problems. To compare these two approaches, we developed a novel method for framing evolutionary populations as mixed strategy NE,

and we created an approach to adversarially train two reinforcement learning agents over the same cybersecurity environment as coevolution.

In the absence of a concrete knowledge of the local equilibria in our reward function to which we could compare our solutions, it was impossible to rank the approaches and qualify the performance. This was not our goal. In fact, what is better for the attacker is worse for the defender, and vice versa. We were left instead to compare algorithms and create benchmarks, which is what we did. We relied on indicators of optimality: the reward to which the algorithm converged after many iterations, the cumulative reward as a proxy for (negative) regret, and the distribution and quality of the solutions themselves.

These indicators suggest that, with the hyperparameter combinations we used, the MARL approach did not achieve high quality solutions. This was likely due to the fact that the reward estimators used are not robust to adversarial attacks–a known limitation of the algorithm we utilized. This is not to say that there does not exist some combination of hyperparameters under which the RL algorithm may have fared better. On the contrary, we are convinced that there are. We are instead focused on the purpose of our investigation, which was practical: to develop the tools to compare these two approaches, which will allow us to make a judgment call on which variant to use for a specific network model on a case by case basis.

Among all the potential equilibria, and without theoretical guarantees of optimality from either algorithm, it may be this process of comparison that affords us the highest quality solutions. Just as the RL results served as a baseline for which we could compare various GE hyperparamter configurations, maybe the GE results could serve as guidance to tuning RL hyperparameters in the future, sort of like an alternating optimization method. GE's robustness to some level of hyperparamter misspecification may make it especially suitable for this process, as one could quickly arrive at a representative solution for coevolutionary algorithms as a baseline for comparison.

In the face of this challenging environment, we note the versatility of coevolution. Without any assumptions about the adversary's behavior, environment, or the shape of the rewards, we were able to achieve high quality solutions that demonstrated a trend toward convergence. There was no need to alter the action space as necessary in RL, and both the attacker and defender could simultaneously act in the same environment. Finally, the simplicity of the model and the absence of the need for gradient updates can be advantageous.

One previous drawback of coevolution in this context was the need to select a Nash equilibrium strategy from the population. By modeling the population as a mixed strategy NE, we were able to sidestep this difficulty altogether. We found that, in our model, a high crossover probability among the population helped to stabilize the reward of the solutions, likely because it allowed for a more diverse mixture of attack pattern and patch combinations to be played. One limitation of this approach is the fact that a finite-size population is only a rough approximation for a distribution over strategies. Another limitation is the possibility of the crossover events creating strategies that are less optimal (such as duplicating attack patterns), but this seems to be outweighed by the benefits toward convergence. The key result is that this model allows coevolution to create mixed, probabilistic strategies in a manner similar to RL while requiring fewer assumptions and with greater robustness to hyperparameter misspecification in our environment.

6 FUTURE WORK

There are several avenues we hope to explore in future work. On the subject of coevolution, the mutation probability is the key hyperparameter that controls the exploration/exploitation balance. Research could be done to determine the relationship between this parameter, the convergence of solutions, and the cumulative reward achieved. Heuristics could be developed for setting this parameter based on environmental conditions, such as the size of the action space. Finally, modifications could be made to bar crossover events from resulting in repeat attack patterns in the same strategy, for example.

The next major RL-related development to pursue would be to tune its hyperparameters and compare the results to those obtained from coevolution. We anticipate that in our model, we could improve the attacker's cumulative reward to be more in line with that of the coevolution. A linkage could be found between the optimal hyperparamters for GE and RL, allowing one to train the coevolution, which generally runs much faster on a CPU, and use those results to tune the RL hyperparameters.

A major weakness we identified in RL was the lack of theoretical guarantees on robustness to adversarial losses. We hope to implement the dilated bonuses algorithm discussed in section 2 to see if this improves outcomes. The combination of this theoretical grounding with investment in hyperparameter tuning could lead the RL agents to be a basis for comparison and evaluation of the coevolutionary solutions.

Future work could also expand upon the environment in order to more realistically model cyber attack progressions. The option of using taps as a form of mitigation on certain software could be added. We also hope to create more realistic network models using a data driven approach on which we could evaluate simulations and observe trends.

ACKNOWLEDGMENTS

This material is based upon work supported by the DARPA Advanced Research Project Agency (DARPA) and Space and Naval Warfare Systems Center, Pacific (SSC Pacific) under Contract No. N66001-18-C-4036.

REFERENCES

[1] [n. d.]. 2022 Frost & Sullivan White Paper on Breach and Attack Simulation. https://info.xmcyber.com/frost1
[2] [n. d.]. NVD - CPE. https://nvd.nist.gov/products/cpe
[3] [n. d.]. NVD - Vulnerability Metrics. https://nvd.nist.gov/vuln-metrics/cvss
[4] 2018. A History of Ransomware Attacks: The Biggest and Worst Ransomware Attacks of All Time. https://digitalguardian.com/blog/history-ransomware-attacks-biggest-and-worst-ransomware-attacks-all-time
[5] 2021. https://cve.mitre.org/
[6] 2022. CAPEC - Common Attack Pattern Enumeration and Classification (CAPEC™). https://capec.mitre.org/
[7] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. 2016. OpenAI Gym. arXiv:arXiv:1606.01540
[8] Rapahel Bronfman-Nadas, Nur Zincir-Heywood, and John T. Jacobs. 2018. An Artificial Arms Race: Could it Improve Mobile Malware Detectors?. In *2018*

Network Traffic Measurement and Analysis Conference (TMA). 1–8. https://doi.org/10.23919/TMA.2018.8506545

[9] Gabriel Dulac-Arnold, Richard Evans, Peter Sunehag, and Ben Coppin. 2015. Reinforcement Learning in Large Discrete Action Spaces. *CoRR* abs/1512.07679 (2015). arXiv:1512.07679 http://arxiv.org/abs/1512.07679

[10] Hemberg Erik. 2022. donkey_ge. https://github.com/flexgp/donkey_ge

[11] Timothy Hart and Daniel Edwards. 1963. *The Alpha-Beta Heuristic*. Technical Report. USA.

[12] Erik Hemberg, Jonathan Kelly, Michal Shlapentokh-Rothman, Bryn Reinstadler, Katherine Xu, Nick Rutar, and Una-May O'Reilly. 2020. Linking Threat Tactics, Techniques, and Patterns with Defensive Weaknesses, Vulnerabilities and Affected Platform Configurations for Cyber Hunting. https://doi.org/10.48550/ARXIV.2010.00533

[13] Chi Jin, Tiancheng Jin, Haipeng Luo, Suvrit Sra, and Tiancheng Yu. 2020. Learning Adversarial Markov Decision Processes with Bandit Feedback and Unknown Transition. In *Proceedings of the 37th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 119)*, Hal Daumé III and Aarti Singh (Eds.). PMLR, 4860–4869. https://proceedings.mlr.press/v119/jin20c.html

[14] Xiaofen Lu, Ke Tang, Stefan Menzel, and Xin Yao. 2019. Competitive Coevolution as an Adversarial Approach to Dynamic Optimization. arXiv:1907.13529 [cs.NE]

[15] Haipeng Luo, Chen-Yu Wei, and Chung-Wei Lee. 2021. Policy Optimization in Adversarial MDPs: Improved Exploration via Dilated Bonuses. In *Advances in Neural Information Processing Systems*, M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan (Eds.), Vol. 34. Curran Associates, Inc., 22931–22942. https://proceedings.neurips.cc/paper/2021/file/c1b8bf9e071c0dabb899e7a27f353762-Paper.pdf

[16] Robert McMillan. 2020. Suspected Russian Cyberattack Began With Ubiquitous Software Company. *Wall Street Journal* (Dec. 2020). https://www.wsj.com/articles/suspected-russian-cyberattack-began-with-a-little-known-but-ubiquitous-software-company-11608036495

[17] Luke Metz, Ben Poole, David Pfau, and Jascha Sohl-Dickstein. 2017. Unrolled Generative Adversarial Networks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net. https://openreview.net/forum?id=BydrOIcle

[18] Sadegh M Milajerdi, Birhanu Eshete, Rigel Gjomemo, and VN Venkatakrishnan. 2019. Poirot: Aligning attack behavior with kernel audit records for cyber threat hunting. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. 1795–1812.

[19] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. 2016. Asynchronous Methods for Deep Reinforcement Learning. In *Proceedings of The 33rd International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 48)*, Maria Florina Balcan and Kilian Q. Weinberger (Eds.). PMLR, New York, New York, USA, 1928–1937. https://proceedings.mlr.press/v48/mniha16.html

[20] Tuomas Oikarinen, Wang Zhang, Alexandre Megretski, Luca Daniel, and Tsui-Wei Weng. 2021. Robust Deep Reinforcement Learning through Adversarial Loss. In *Advances in Neural Information Processing Systems*, M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan (Eds.), Vol. 34. Curran Associates, Inc., 26156–26167. https://proceedings.neurips.cc/paper/2021/file/dbb422937d7ff56e049d61da730b3e11-Paper.pdf

[21] Hamed Okhravi, Thomas Hobson, David Bigelow, and William Streilein. 2013. Finding focus in the blur of moving-target techniques. *IEEE Security & Privacy* 12, 2 (2013), 16–26.

[22] M. O'Neill and C. Ryan. 2001. Grammatical evolution. *IEEE Transactions on Evolutionary Computation* 5, 4 (2001), 349–358. https://doi.org/10.1109/4235.942529

[23] Una-May O'Reilly, Jamal Toutouh, Marcos Pertierra, Daniel Prado Sanchez, Dennis Garcia, Anthony Erb Luogo, Jonathan Kelly, and Erik Hemberg. 2020. Adversarial Genetic Programming for Cyber Security: A Rising Application Domain Where GP Matters. *Genetic Programming and Evolvable Machines* 21, 1–2 (jun 2020), 219–250. https://doi.org/10.1007/s10710-020-09389-y

[24] Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. 2021. Stable-Baselines3: Reliable Reinforcement Learning Implementations. *Journal of Machine Learning Research* 22, 268 (2021), 1–8. http://jmlr.org/papers/v22/20-1364.html

[25] Tim Salimans, Jonathan Ho, Xi Chen, Szymon Sidor, and Ilya Sutskever. 2017. Evolution Strategies as a Scalable Alternative to Reinforcement Learning. https://doi.org/10.48550/ARXIV.1703.03864

[26] Sevil Sen, Emre Aydogan, and Ahmet I. Aysan. 2018. Coevolution of Mobile Malware and Anti-Malware. *IEEE Transactions on Information Forensics and Security* 13, 10 (2018), 2563–2574. https://doi.org/10.1109/TIFS.2018.2824250

[27] Michal Shlapentokh-Rothman, Jonathan Kelly, Avital Baral, Erik Hemberg, and Una-May O'Reilly. 2021. *Coevolutionary Modeling of Cyber Attack Patterns and Mitigations Using Public Datasets*. Association for Computing Machinery, New York, NY, USA, 714–722. https://doi.org/10.1145/3449639.3459351

[28] Moshe Sipper, Weixuan Fu, Karuna Ahuja, and Jason H Moore. 2018. Investigating the parameter space of evolutionary algorithms. *BioData mining* 11, 1 (2018), 1–14.

[29] William Smart. 2018. *Lessons Learned review of the WannaCry ransomware cyber attack*. Technical Report. National Health Service. https://www.england.nhs.uk/wp-content/uploads/2018/02/lessons-learned-review-wannacry-ransomware-cyber-attack-cio-review.pdf

[30] Kaiqing Zhang, Zhuoran Yang, and Tamer Başar. 2021. Multi-Agent Reinforcement Learning: A Selective Overview of Theories and Algorithms. In *Handbook of Reinforcement Learning and Control*. Springer International Publishing, Cham, 321–384.

[31] Linda Zhang and Erik Hemberg. 2019. Investigating algorithms for finding nash equilibria in cyber security problems. 1659–1667. https://doi.org/10.1145/3319619.3326851