Coevolutionary Modeling of Cyber Attack Patterns and Mitigations Using Public Datasets

Michal Shlapentokh-Rothman
MIT
Cambridge, MA, USA
mshlapen@alum.mit.edu

Jonathan Kelly
MIT
Cambridge, MA, USA
jgkelly@alum.mit.edu

Avital Baral
MIT
Cambridge, MA, USA
abaral@mit.edu

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

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

ABSTRACT

The evolution of advanced persistent threats (APTs) spurs us to explore computational models of coevolutionary dynamics arising from efforts to secure cyber systems from them. In a first for evolutionary algorithms, we incorporate known threats and vulnerabilities into a stylized "competition" that pits cyber *attack patterns* against *mitigations*. Variations of attack patterns that are drawn from the public CAPEC catalog offering Common Attack Pattern Enumeration and Classifications. Mitigations take two forms: software updates or monitoring, and the software that is mitigated is identified by drawing from the public CVE dictionary of Common Vulnerabilities and Exposures. In another first, we quantify the outcome of a competition by incorporating the public Common Vulnerability Scoring System - CVSS. We align three abstract models of population-level dynamics where APTs interact with defenses with three competitive, coevolutionary algorithm variants that use the competition. A comparative study shows that the way a defensive population preferentially acts, e.g. shifting to mitigating recent attack patterns, results in different evolutionary outcomes, expressed as different dominant attack patterns and mitigations.

CCS CONCEPTS

• **Computing methodologies** → *Multi-agent systems.*

KEYWORDS

Coevolution, cyber security, modeling and simulation

ACM Reference Format:
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. In *2021 Genetic and Evolutionary Computation Conference (GECCO '21), July 10–14, 2021, Lille, France.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3449639.3459351

GECCO '21, July 10–14, 2021, Lille, France

ACM ISBN 978-1-4503-8350-9/21/07.
https://doi.org/10.1145/3449639.3459351

1 INTRODUCTION

Advanced Peristent Threats (APTs) are a class of severe and sophisticated cyber threats perpetrated by known groups of actors, who have been identified by the tactics, techniques and procedures they typically use. While advanced persistent threats (APTs) continue to defeat common cyber defenses, community action is constantly improving the hand of defenders. One kind of community action occurs when victims of APTs go beyond recovering from an attack, onto deciphering who may have harmed them, how that harm was inflicted, and where it was targeted. Motivated by the value of sharing information, victims disclose their experiences publicly through a number of government and industry reporting outlets. Large repositories of potential threats, systematically compiled from attack reports preemptively warn the entire security community. Specific repositories, e.g. ATTACK™, reporting *Advanced Persistent Threats* (APTs), CAPEC, cataloguing APT *Attack Patterns*, and CVE, enumerating observed examples of *vulnerabilities* targeted by *Attack Patterns*, are being used in a variety of ways, see CAPEC use cases [15], though not yet by evolutionary computation. [12–14].

Evolution of APTs and other cyber threats is well documented by the security community [2, 3, 7, 24]. To study this phenomena through modeling, we choose to use competitive coevolutionary algorithms [21] because they are well-suited to move from observations to studies providing understanding. This phenomena is interesting to model because deciding which parts of a system to patch while maintaining network operations is a problem with a large search space that is currently solved manually by human experts. Our first challenge is to ensure competitions are effective abstract models and simulations of coevolutionary interactions that occur between cyber analogs to the algorithm's definitions of the adversaries. Therefore, we formulate competitions featuring a set of attack patterns versus a set of mitigations. To search through variations of attack patterns and mitigations the algorithms incorporate, for the first time, public APT and vulnerability resources, specifically the CAPEC catalog and the CVE dictionary. The CAPEC catalog enumerates a search space of APT attack patterns and the CVE dictionary enumerates software that is the focus of mitigations.

Our second contribution is to formulate different abstractions of generally described evolutionary APT dynamics to understand

how contrasting preferences lead to different outcomes. First, we align a standard coevolutionary algorithm, Coev, with an abstract description of attack patterns coevolving back and forth with mitigations. Next, we align a previously introduced variant of a coevolutionary algorithm named LS_Coev with an abstract description where evolving defenders react more slowly than attackers. Finally, motivated by observations about publicity around new attacks, and the urgency of these attacks, we design a variant named Recent_Coev that aligns to defenders adaptively selecting mitigations that counter the most recently popularized attack, while attackers adapt to the least vulnerable networks most recently observed. We combine the competition and algorithm variants in a system called **EvoAPT** to enable modeling.

We proceed as follows. In Section 2 we describe how we set up **EvoAPT**'s algorithmic components to model APT evolution. Section 3 contextualizes our work with closely related cybersecurity modeling and simulation approaches also using competitive coevolutionary algorithms. Section 4 describes our simulation setup. Section 5 presents the results. Section 6 concludes.

2 MODELING

In this section we describe APTs and how we set up our algorithm components: competition and competitive coevolutionary algorithm variants to model APT evolution. Our system for exploring the variants with the competition is called **EvoAPT**.

2.1 Advanced Persistent Threats (APTs)

APTs are a class of severe cyber threats perpetrated by known groups of actors, (called APTs as well), who have been identified by the tactics, techniques and procedures (TTPs) they typically use [2]. One recent and prominent example of an APT is Solarigate [4]. A cyber security firm, Fireye, discovered this global intrusion campaign, that compromised their own networks and exfiltrated their intellectual property. They identified a supply chain attack trojanizing SolarWinds Orion business software updates. The trojan in turn distributed malware called SUNBURST. Sunburst used a compromised software component to use SolarWinds' Orion to detect and in some cases attempt to disable defensive software running on targeted systems. If any of an extensive list of processes was found to be running, the component shut down completely until called again [26].

Post compromise, the attackers were able to leverage multiple techniques that helped them evade detection for over a year and obscure their malicious activity. The Solarigate campaign is widespread, affecting public and private organizations around the world.

As noted in the Solarigate example, an APT is deployed as a campaign. A campaign consists of multiple steps, taken over long, irregular time intervals, each in a stealthy way. We align a list of attack patterns with a step of a campaign. Attack patterns can be found in the CAPEC catalog which tracks, via external links to the CWE, the software weakness they target, and, from the CWE entries, the vulnerable software applications they have previously targeted. For example, using BRON [6], the affected software targeted by an attack pattern named `CAPEC-17`, entitled "Using Malicious Files" can be retrieved. An attack of this type exploits a system's configuration to allow an attacker to directly access an executable file, e.g. through shell access and has targeted web browsers. BRON follows the links from `CAPEC-17` to all of its entries in the CWE. For example, one weakness is `CWE-264`: Permissions, Privileges, and Access Controls. BRON then follows all links from each weakness to CVE entries. Each is a vulnerability, e.g. there is one named `CVE-2011-1185`, and each entry in the CVE dictionary lists software configurations, in Common Platform Enumeration (CPE) format, that have been actually exploited by `CAPEC-17`. One example is `cpe:2.3:a:google:chrome:*:*:*:*:*:*:*:*`, up to (excluding) version 10.0.648.127 of Google Chrome.

We can consider the general class of attack patterns analogous to a species, and the entries in the CAPEC catalog to be explicit variants of attack patterns. These attack pattern variants exhibit differences in terms of how many examples of vulnerabilities they have attacked (equivalent to the number of CVE entries linked to them) or how many configurations they affect.

On the defensive side, one general approach to APT mitigation is updating the software of a vulnerable application or operating system. This is intended to be preventative, but in some circumstances, is neutral or even worse. A second approach is needed when updating software is infeasible, e.g. when an operating system is running applications which don't have versions supported by the update or an application has no update. For these cases, monitoring is deployed to watch for the stealthy activity of the attack. Monitoring, also called sensing, is arguably better at preventing harm from an attack but it is often costly because it involves collecting lots of data where the signal revealing the attack is very weak. We model both these approaches to mitigations. To vary what software is mitigated prior to an attack, we reference the CVE dictionary.

2.2 Competition

The competition in the threat scenario used by our competitive coevolutionary algorithm variants is formulated as follows:

- An attack is a list of attack patterns and vulnerabilities. The search space of attack patterns is the CAPEC catalog and CVE dictionary. Entries in the catalog are linked via the CWE (Common Weakness Enumeration) catalog. Because different attack patterns can link to a common CWE entry, attack patterns overlap in terms the vulnerabilities they have been observed to attack.
- A defense is a list of two kinds of mitigations: monitor or update, targeted on a node or software. Monitoring mitigations is more costly than updates but more assured to help. A defense has a fixed budget. The search space of software to which mitigation is applied is the CVE dictionary.
- A network, described by its software applications, the nodes they run on and topology, is the competition environment.
- To simulate a competition, the algorithm "applies" defense mitigations to the "network" (applying will be explained next) and then calculates a competition-score assuming the attack occurs on the mitigated network. The competition-score represents the potential harm of the attack on the mitigated network.
- To apply a monitoring mitigation to software on a selected node, the algorithm uses the node's centrality as a proxy for the

expense of the logging and analysis. It temporarily changes the *Common Vulnerability Scoring System* [17] (CVSS) value to zero.

- To apply a software update as a mitigation, the algorithm increments the version of the operating system or the version of the software that has been selected by the defense. This may result in the updated software's vulnerability changing and, therefore, its CVSS. (The mitigation intends the software to be less vulnerable by upgrading it, but there is no guarantee, the new version is not itself a vulnerability.)
- To compute a competition-score, the algorithm checks whether the attack patterns impact any the network's software applications after all mitigations have been applied. This check is performed by a data collection and retrieval system called BRON [6]. **EvoAPT** formats the network's description in *CPE* [16] format, a Common Platform Enumeration standard to make itself interoperable with BRON. If BRON finds the software application is affected, it returns a value from the CVSS from the CVE entry. Domain experts assign CVSSs in the CVE dictionary using a free and open industry standard for assessing vulnerabilities. The CVSS depends on the accessibility, integrity, and confidentiality of the application and is in the range of 0 to 10. Note that because attack patterns are linked to common CWE weakness entries, implying overlap in affected software applications, the competition-score is not the sum of each attack pattern's CVSS, while it is the sum of the CVSSs of all affected software.
- A population member's fitness is the sum of competition-scores over all competitions in which they participate.

2.3 Coevolutionary algorithm variants

Moving to the evolutionary phenomena we want to model, we note that APTs frequently indiscriminately target large slices of networks, not necessarily because they *are* ideal targets, but because they, in general, *could be*. On the other hand, defensive mitigations will preventively repel some attacks and not others. Thus we can model two populations where the members of each compete, with our formulated competition, pairwise against each other, and the fitness of each member is the sum of the harm that is either inflicted (a maximizing objective for attacks) or incurred (a minimizing objective of the same quantity: harm, for defenses) over all a member's competitions. We note that this basic formulation of interacting circumstances and many competitions also underpin prior work such as [20, 22].

A baseline model of dynamics is well served by "standard" or "alternating" coevolutionary algorithm that evolves two populations with selection and variation using standard selection, crossover and mutation techniques. One population comprises attacks and the other defenses. In each generation, competitions are held by pairing an attack and a defense. Each attack–defense pair in the competition is assigned a score. Fitness is then calculated as the sum of competition scores. The populations are evolved in alternating steps: first, the attack population is selected, varied, updated and evaluated against the defenses, and then, the same for the defense population. We name this standard coevolutionary algorithm Coev.

Recognizing that APTs are stealthy and take time to detect, we include a previously introduced variant of a coevolutionary algorithm named 'Lockstep', that we denote as LS_Coev [9]. It algorithmically, see Algorithm 1 and Figure 1, evolves in epochs where, first the attacks evolve multiple generations against a static population of defenses, then the attacks are locked down, and defenses evolve against them, though for fewer generations. This variant is parameterized to control the number of generations each adversarial population is static or evolving.

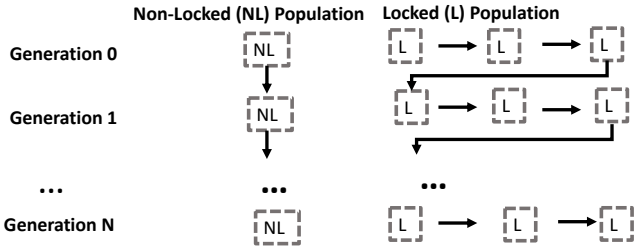

Figure 1: Lockstep coevolutionary algorithm.

Algorithm 1 LS_Coev

```
procedure PopulationStep(population, parents_size, adversaries)
    parents ← TournamentSelection(population, parents_size)
    new_individuals ← Variation(parents)
    for individual in new_individuals do
        individual.fitness ← AvgFitness(individual, adversaries)        ▷ MEU solution concept
    population ← GenerationalReplacement(new_individuals, population)
procedure LockstepCoevolution(populations, generations, locked_population_parents)
    t ← 0
    best_individuals ← ∅
    while t < generations do                                            ▷ Run for # generations
        locked_pop ← populations_locked
        nonlocked_pop ← populations_nonlocked
        t′ ← 0
        while t′ < locked_pop_generations do      ▷ locked population evolves against fixed adversary population
            PopulationStep(locked_pop, sizeof(locked_pop), nonlocked_pop)
            t′ ← t′ + 1
        PopulationStep(nonlocked_pop, sizeof(nonlocked_pop), locked_pop)
        PopulationStep(locked_pop, locked_population_parents_size, nonlocked_pop)
        best_individuals ← ExtractBest(populations)
        t ← t + 1
    return best_individuals                                             ▷ Returns best solutions found
```

Finally, Solarigate illustrates that, once an APT is found, the entire community should try to find and address it within their own enterprise. Therefore, we design a variant named Recent_Coev. Recent_Coev proceeds with two evolving populations, attacks, $\mathbf{A} = A_1, .., A_n$, and defenses, $\mathbf{D} = D_1, ..., D_n$, plus a single "recent" attack, A_R and a single "recent" defense mitigation, D_R. In a generation, the attacks are evolved then competed against D_R and the defenses are evolved and competed against A_R. Then, the algorithm assigns A_R the best performing attack A^* and D^* the best performing defense before executing the next generation. In this way, both populations are always evolved against the most recent, most effective adversary.

3 RELATED WORK

Table 1 presents **EvoAPT** and earlier projects that applied coevolutionary algorithms, including modeling, to cyber security settings. The information in the table focuses on the coevolutionary algorithm used, the cyber security environment, the cyber security

topic, attack behaviors, defense behaviors, model evaluation environment and objective.

Systems in Table 1 vary in terms of realism, though they are all models and models are by definition simplifications of reality. One basis of realism is whether a system runs actual attack code (aka malware) like a threat-actor does. On this basis, RIVALS-Deception [20] runs attack code (when it runs nmap to reconnaissance scan). In the middle of the spectrum of realism, RIVALS-DDOS [20] runs code that models a DOS attack, rather than execute DOS malware. On this basis, **EvoAPT** is at the abstract end of the spectrum. While CAPEC attack patterns include code examples that in the wild, execute as trojan or other malware, **EvoAPT** does not *execute* any form of attack while it does however, represent real-world attack patterns and draw upon a huge catalog of them.

On the basis of types of behavior being modeled, **EvoAPT**, like CANDLES [22], (row 1, Table 1), investigates an abstraction of behaviors that differ from investigating "data features", like, e.g. AIS-TCP [19], (row 6 of Table 1). **EvoAPT** traces an attack pattern's CWE links to all of its CVE entries to identify all of its real-world observed examples. It quantifies an attack pattern's harm on the competition network by consulting the CVSS. The CVSS is a real world estimate. Collectively, these features make **EvoAPT** structurally novel among similar systems, and make its approach abstract yet faithful to the meaning of an APT and attack pattern severity. **EvoAPT** also uses CPE formatting to support "plug-and-play" like integration with BRON and formats in the data. This appears to makes it unique (among entries in Table 1) in using this global standard widely adopted by the cyber security community.

There is previous work that uses CAPEC, CWE and CVE information, but not coevolutionary algorithms. For example, the SEPSES knowledge graph links information from standard data sources including CVE, CAPEC, CWE, and CVSS [10]. CALDERA Pathfinder [8] shows what vulnerability is exposed to an adversary based on threats linked via attack patterns from CAPEC. In addition, the data is also used in examples of modeling including situational awareness [11], predicting missing edges between CVE, CWE and CAPEC [29], and investigating data breaches with semantic analysis of ATT&CK [18].

4 SIMULATION SETUP

We define the adversarial behavior, objectives, network and experimental parameters in this section.

4.1 Defining the Adversarial Behavior

EvoAPT uses Grammatical Evolution (GE) which is a type of evolutionary algorithm [23]. GE has been used in several adversarial domains in cybersecurity [5, 22, 25]. It uses a Backus Naur Form (BNF) context-free grammar and an intermediate interpreter to map from the "genome" to a "phenome" that expresses an executable behavior. Like all EAs, variation occurs on the genome (in GE, an integer sequence) and fitness depends on the phenome. We refer to the genome-phenome pair as an individual. In GE the interpretation step raises locality issues, however the grammar and the rewriting assure syntactically valid offspring [27]. Experience indicates that the evolutionary dynamics of GE do not differ significantly from other forms of genetic programming. A grammar allows the representation of candidate solution behavior to be easily customized and expressed in direct domain vocabulary. Grammars also offer design flexibility: changing out a grammar and the environment of behavioral execution does not require any changes to the rest of the algorithm.

Figure 2: Attack and Defense Grammars

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

# DEFENSE GRAMMAR
<defense> ::= <mitigation>,<mitigation>,<mitigation>,<mitigation>
<mitigation>::= <update> | <monitor>
<monitor>::= <node>,<ap_id>
<update> ::= 'os',<node> | 'app',<app_id>
<ap_id> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6| 7| 8| 9 ...
<node> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6| 7| 8| 9 ...
```

We describe the search space of attacks with a BNF grammar, see Figure 2. An individual encodes 3 attack patterns and one software vulnerability. Entries in the CAPEC catalog (we used 579 values retrieved from [6] are selected by an individual using a numerical identifier lookup. The attack grammar allows these identifiers to be evolved. The attack includes a CVE entry (i.e. software vulnerability) that does not have to be associated with an attack pattern. This allows a known vulnerability to be targeted without being targeted using any attack patterns from CAPEC.

The defense grammar, see Figure 2, describes a list of network mitigations. **EvoAPT** integrates a budget constraint into defending. Conceptually, it assigns a cost to both kinds of mitigations – software updates and software monitoring. It limits the maximum number of monitoring mitigations to model their frequently high cost and allows updates that don't exceed the budget limit. At an implementation level of cost and budgeting, our limit allows a defense only 4 mitigations and at most 3 of them being monitoring. When a mitigation is beyond its budget, the mitigation is ignored.

4.2 Network

We experiment with 2 enterprise class C like networks, provided by domain experts. They are large and small in size. The large network has 787 nodes and 789 edges while the small network has 147 nodes and 147 edges. For each network, **EvoAPT** stores an adjacency graph. Nodes have a type property: e.g. server, client. Each node also contains a list of the applications that execute on it, specified in Common Platform Enumeration (CPE) format. We calculate centrality of a node, (recall, to assign a cost to a monitoring mitigation), using a function called `degree_centrality`[1].

4.3 Adversarial Objective

The defense-attack coupled objective is "minimax", i.e. the defense seeks to minimize the maximum fitness of the attack. Practically, in **EvoAPT**'s implementation level, attacks maximize fitness, i.e.

[1]https://networkx.github.io/documentation/networkx-2.2/reference/algorithms/centrality.html

Table 1: Coevolution in cyber security applications. A name to simplify identification, the coevolutionary algorithm variant, cyber security environment, competition, attack and defense behaviors, model of environment and objective are shown in the columns.

Name	Alg. Variant	Topic	Competition	Attack behavior	Defense behavior	Env Model	Objective
CANDLES [22]	Alternating	Network exploit and propagation	Prevention vs Contagion	Exploit & Reconnaissance	Detection & Mitigation	Simplified Network Simulator	Adversary profit
RIVALS-Enclave [20]	Alternating	Network Enclave Partition	Isolation vs Contagion	Malware Contagion "strength"	Tap and device placement	Graph	Mission Delay
RIVALS-DDOS [20]	Alternating, Archives, Compendium	Peer-2-peer network	Robustness vs Denial	Graph impairment	Network settings	Simplified network simulation	Mission Disruption
RIVALS-Deception [20]	Lockstep	SDN network	Deception vs Reconnaissance	NMAP scan	Deceptive SDN configuration	Mininet	Detection Speed
RIVALS-SDP[25]	Alternating	Software Defined Perimeter	Security vs Useability	Effort	Compliance and Tap	Graph	Risk and Useability
AIS-TCP[19]	Alternating	Network traffic	Detection vs Evasion	Self	Non-self	TCP dump	Detection accuracy
Coev-Malware[24]	Alternating	Android Malware and detection	Detection vs Evasion	Malware variant	Code features	Code and virus detectors	Detection accuracy
ArmsRace-1[1]	Parallel with Archive	Android Malware and detection	Detection vs Evasion	Malware variant	Code features	Code and virus detectors	Detection accuracy
ArmsRace-2[28]	Alternating	Android Malware and detection	Detection vs Evasion	Malware variant	Code features	Code and virus detectors	Detection accuracy

Table 2: Experiment parameter names and values

Parameter name	Value
Population size	20
Generations	10
Lock Step Population size	10
Lock Step Generations	2
Crossover Probability	0.9
Mutation Probability	0.1
Max Length	100
Tournament Size	2
Elite size	1
Number of runs	30

Table 3: EvoAPT Algorithm Variants and names

Variant	Name
Evolve attacks vs a non-evolving defense.	Attack
Evolve defenses vs a non-evolving attack	Defense
Coevolve both populations, alternating	Coev
Coevolve with Lock-step	LS_Coev
Coevolve with Recency prioritization	Recent_Coev
Coevolve both populations, alternating clipped search space	CLIPPED

the harm they inflict, and defenses maximize the negated fitness of the attack, minimizing harm inflicted.

4.4 Experimental Parameters

Experimental parameters are in Table 2 and our algorithm variants are named in Table 3. To identify the best attack and defense from a run, for the large network, we run an "out of sample evaluation". We start with a pool of the fittest members of the run's two populations at the last generation. We compete this pool against a small set of new adversaries that we manually select from other runs executed independently from these experiments. The fittest attack and defense of this exercise are designated best-of-run. This evaluation creates an even playing field for comparing the best-of-run's. It is necessary because fitness in a coevolutionary algorithm is relative to a population of competitors and each run's final population is assumed to be different.

The experiments are summarized in Table 3. Each experiment is run with competitions on the large and small networks. The next section describes the results from the experiments.

5 SIMULATIONS

We present the averaged results of 30 runs for each of the 5 algorithm variants, with both networks (large and small) in Table 4. The out-of-sample performance can be examined in Table 5. First, we will compare the overall results of the 5 algorithm variants and then we will look at each variant individually.

Table 4: Average Best Fitness for the 5 algorithm variants and two network sizes. The theoretically highest possible attack fitness for the small network is $109,411$ **and for the large network is** $426,814$**. The theoretically best defense fitness for the small network is** $-108,224$ **and for the large network is** $-427,690$

	Small Network		Large Network	
Variant	Attack Fitness	Defense Fitness	Attack Fitness	Defense Fitness
Attack	$109,393 \pm 0.0$	NA	$426,659 \pm 0.0$	NA
Defense	NA	$-90,566 \pm 2,260$	NA	$-416,691 \pm 3,302$
Recent_Coev	$108,367 \pm 0.0$	$-107,835 \pm 55$	$426,813 \pm 3$	$-419,668 \pm 50$
Coev	$105,386 \pm 1,505$	$-104,520 \pm 1,530$	$426,648 \pm 1,564$	$-421,485 \pm 2,445$
LS_Coev	$105,782 \pm 1,313$	$-105,031 \pm 1,450$	$426,652 \pm 1,160$	$-423,619 \pm 1,785$

Table 5: Out of Sample (OoS) Fitness

Variant	OoS Attacker	OoS Defender
Attack	NA	Mean:426, 620, Min: 426, 526
Defense	Mean:−408, 782, Min: −426, 750	NA
Recent_Coev	Mean:−408, 746, Min:−426, 714	Mean:426, 626, Min: 426, 526
Coev	Mean:−408, 623, Min:−426, 596	Mean:426, 617, Min: 426, 517
LS_Coev	Mean:−408, 591, Min:−426, 536	Mean:426, 620, Min: 426, 522

5.1 Algorithm Variant Comparison

Table 4 provides performance measurements of the 5 algorithm variants with both large and small networks. In general, we would expect that a population that is evolving against a static adversary

would have a higher fitness value compared to two populations coevolving in response to each other. Our experiments validate this assumptions, per Table 4 where both Attack and Defense outscore coevolutionary variants. The theoretically highest possible attack fitness for the small network is 109, 411 and for the large network is 426, 814. The theoretically highest lowest defense fitness for the small network is −108, 224 and for the large network is −427, 690. We see that an adaptive defense (attack), assuming a static attack (defense), can reliably approach these theoretical limits. We expect that additional fitness evaluations would allow them to reach them.

Table 5, column 2, provides the performance of the best defenses selected from a run when they are pitted against a small set of out of sample attacks (lower fitness indicates a better performance). The Defense defenses do not perform as well as the coevolved ones because they have been evolved narrowly, ie. against a static attack. Out of sample fitness performance differs among the variants of coevolution. The defenses found by LS_Coev are the fittest relative to the other approaches, Coev is in the middle and Recent_Coev is the least fit (though fitter than Defense). This reflects the different modeling purposes of the algorithms. The Coev variant simulates an interaction dynamic that is more concurrent, where defenses react much quicker than in LS_Coev. During evolution, Recent_Coev redefines its focus each new generation to the strongest attack or defense in the previous generation, so it is exposed to fewer attack variations, making it more brittle to previously-unseen attacks.

Table 5, column 3, provides the performance of the best attacks selected from a run when they are competed against a small set of out of sample defenses (higher fitness is better). The relative results are less definitive so we examined the defenses manually. The examination uncovered the existence of a small number of attack patterns that link to a large number of vulnerabilities, implying that a large group of the vulnerabilities have similar CVSSs. These "powerful' attack patterns are very influential, i.e. are found in all of the strongest attacks from each of the evolution and coevolution simulations.

We count the number of unique attack patterns found in the final populations of the 30 runs for each variant. The large and small networks are exposed, respectively, to the threats of 196 and 186 attack patterns. This is a only fraction of the total number of attack patterns in the CAPEC catalog.

5.2 Evolving Attacks, Static Defense

The goal of the Attack simulations was to examine the performance of an attack that is adapting versus a non-evolving defense that we manually designed. We examine the evolved individuals' attack patterns. Almost all (≈ 98%) of the attack patterns that posed a threat to the networks were found at least once in the best performing individual in both small and large networks runs. There were many more unique attack patterns that appeared in the best performing individual: we found a total of 517 and 515 unique attack pattern variants across all the individuals for the large and small networks. Some attack patterns occurred quite frequently compared to others, as can be seen Table 6. Many of the attack patterns that frequently occurred are related to buffer overflow. We also note that some attack patterns were more likely to occur with certain other attack patterns. This is likely explained by the fact that attack patterns overlap with the vulnerabilities they have been observed to attack because of linkage through common CWE weakness entries.

Table 6: Most frequent attack patterns that occurred in the best performing individual in the large network Attack simulation across the 30 runs.

Single attack patterns
Overflow Buffers (61)
XML Oversized Payloads (42)
Leverage Executable Code in Non-Executable Files (39)
Manipulating Web Input to File System Calls (37)
Using Malicious Files (32)
Pairs of correlated attack patterns
Manipulating Web Input to File System Calls, Overflow Buffers (33)
Leverage Executable Code in Non-Exec. Files, XML Oversized Payloads (25)
Restful Privilege Elevation, XML Oversized Payloads (21)
AJAX Fingerprinting, Overflow Buffers (20)
Using Malicious Files, XML Oversized Payloads (18)

Table 7: Software selected for mitigation in the Defense simulations. The frequency indicates how many times the highest performing individual (in a trial) referenced this node. There were 120 total nodes referenced.

Network	Node ID	Frequency	Centrality
Small Network	windows_10-ABUsr11	7	0.007
	windows_10-ABUsr2	9	0.007
	windows_server_2008-AHProxy	13	0.007
	windows_server_2008-AHDNS	15	0.007
	windows_server_2008-JF	26	0.007
Large Network	windows_server_2008-Dfile4	7	0.0012
	windows_server_2008-AHProxy	9	0.0012
	windows_server_2008-Q	13	0
	windows_10-MCN	22	0.0025
	windows_10-CEF	26	0.012

5.3 Static Attack, Evolving Defenses

The goal of the Defense simulations was to examine the performance of defenses against a non-evolved (static) attack that we manually selected as effective when examining the final population of a run. An overview of the different evolved mitigations is presented in Table 8. Generally, we see that the defense chose to mitigate potential attacks on an operating system (OS) most of the time. The OS most commonly referenced was a Windows 2008 Server. We also see that for applications, monitoring was more frequent than software updating. This makes sense since monitoring has a higher impact (CVSS is zero'd). The two networks, with different sizes and software configurations, differed in the application that was most mitigated. However, runs on both networks referenced many applications that were mitigated only once during the run. This is likely an artifact of the simulations' small population size and the number of fitness evaluations the runs executed.

We are particularly interested in the most frequently appearing nodes. These are listed in Table 7 along with their centrality. Theoretically, these would indicate to security personnel which nodes on the system they should consider most vulnerable, given the applications (and versions of those applications), they support and their operating system version.

Table 8: Details on the different mitigations that occurred in the best performing individual in the DEFENSE simulations for the large and small networks

Mitigation/ NodeType/Common	Large Network	Small Network
Monitor	0.225	0.22
OS update	0.59	0.64
Application update	0.18	0.14
Client	0.43	0.50
Server	0.57	0.50
Most Common OS	Windows Server 2008	Windows Server 2008
Most Common Application	Google Chrome	Adobe Acrobat

Table 9: Details on the different mitigations that were found in the best performing defense for the large network in the coevolution simulations. The numerical values indicate the percentage of the mitigations that contained this value.

Mitigation/NodeType/Common	RECENT_COEV	COEV	LS_COEV
Monitor _Add	0.23	0.275	0.25
OS update	0.54	0.53	0.58
Application update	0.23	0.195	0.17
Client	0.33	0.55	0.56
Server	0.667	0.45	0.44
Most Common OS Mitigated	Windows 10	Windows 10	Windows 10
Most Common App Mitigated	Internet Explorer	Adobe Acrobat	Adobe Acrobat

Table 10: Coevolved nodes selected for mitigation. The frequency indicates how many times the highest performingindividual (in a run) referenced this node. There were 120 total nodes referenced. Centrality is shown in column Centr.

	Large Network			Small Network		
Variant	Node ID	Freq.	Centr.	Node ID	Freq.	Centr.
RECENT_COEV	ABFile	8	0.0012	AGF	9	0.007
	CFTP	9	0.0012	windows_server_2008 AHDNS	12	0.007
	DFile 3	12	0.0012	windows_server_2008 AIDNS	21	0.007
	DFile 4	16	0.0012	JF	21	0.007
	windows _10 CEF	35	0.0012	AIF	25	0.007
COEV	CK	3	0.0012	windows_10-ABUsr15	3	0.007
	windows_10 CL	3	0.0012	windows_server_2008 -OFDNS	3	0.007
	windows_10 CK	4	0.0012	windows_server_2008- AIDNS	4	0.007
	windows_10 MCN	7	0.0025	windows_server_ 2008 0OWeb	4	0.007
	windows_10 CEE	7	0.0012	windows_server_ 2008 A	5	0.007
LS_COEV	windows_10 FileCont	3	0.0012	windows_server_2008 AGF	2	0.007
	windows_server_2008 DFile4	3	0.0012	windows_server_2008 AINC File	2	0.007
	windows_10 CK	4	0.0012	windows_10 ARUser14	3	0.007
	windows_10 MCN	6	0.0025	windows_server_2008 AHproxy	4	0.007
	windows_10 CEF	8	0.0012	windows_server_2008 JF	10	0.007

5.4 Coevolving Defenses and Attacks

A summary of the algorithm variants' evolved fitness is presented in Table 4. Summaries of the different mitigations found by coevolution defenses can be found in Tables 9 and Table 10. We discuss each variant one at a time, comparing each to the others.

RECENT_COEV. From the out-of-sample performance, we see that in the defense case RECENT_COEV coevolution, has better performance than the static cases but worse performance compared to COEV and LS_COEV. The algorithm was able to find 468 unique attack patterns across the 30 runs in both the large network and small network. However, the attack only found 91% of the possible attack patterns for both networks which is smaller compared to the percentage found by the attack evolving against a fixed defense. This is expected because adapting to an adapting adversary is more challenging for evolutionary search.

The defense mitigations used by RECENT_COEV were a bit different than the ones chosen in the DEFENSE simulations but were fairly consistent with the mitigations found by COEV and LS_COEV. RECENT_COEV preferred servers to clients which was different than COEV and LS_COEV. Further analysis is required to figure out why.

Table 11: Attack patterns that occurred in the best performing individual in the small and large network coevolution simulation across the 30 runs

Coevolution	Large Network	Small Network
RECENT_COEV	Overflow Buffers (20)	Overflow Buffers (19)
	Dom-Based XSS (14)	Leverage Executable Code in Non-Executable Files (13)
	XML Oversized Payloads (12)	Manipulating Web Input to File System Calls (13)
	XSS Using MIME Type Mismatch (12)	Target Programs with Elevated Privileges (12)
	Buffer Overflow in an API Call (11)	XML Oversized Payloads (12)
COEV	Using Malicious Files (11)	Target Programs with Elevated Privileges (9)
	Filter Failure through buffer overflow (11)	Restful Privilege Escalation (11)
	Restful Privilege Escalation (12)	Buffer Overflow (12)
	Overflow Buffers (12)	Using Malicious Files (12)
	XML Oversized Payloads (15)	XML Oversized Payloads (16)
LS_COEV	XML Oversized Payloads (11)	Leverage Executable Code in Non-Executable Files (10)
	Manipulating Web Input to File System Calls (12)	AJAX Fingerprinting (10)
	XML Nested Payloads (14)	XML Oversized Payloads (11)
	Overflow Buffers (15)	XML Nested Payloads (15)
	Dom-Based XSS (17)	Overflow Buffers (16)

COEV. COEV had the second best performance out the algorithm variants in the out-of-sample test. We see similar attack patterns among highly fit attacks. There were 475 and 482 unique attack patterns for the large and small networks, which accounted for about 93% of the possible attack patterns from both networks. One of the main benefits of coevolution is the ability to explore a more diverse group of both defenses and attacks which ultimately leads to being more 'prepared' for an unknown attack. Finding 93% of the possible attacks combined with the varied defenses led to the increased performance over the RECENT_COEV.

LS_COEV. LS_COEV had the highest fitness in the out-of-sample defense test. While the most frequently used attack patterns were similar, the number of unique attack patterns found and the percentage of possible attack patterns was higher for LS_COEV: 482 unique attack patterns for the large network, which account for 94% of the possible attack patterns of the large network. While the increase is not large, there are some individual attack patterns that can be quite damaging and can lead to a large increase or decrease in fitness. If we increased the number of lock step generations, we likely would get close to having at least one of each possible attack pattern in an attack solution. LS_COEV and COEV used mitigations similarly.

5.5 Search Space Clipping

To consider how the simulations may apply more directly to existing systems, we narrowed the search space for both attacks and defenses by examining the experimental runs and extracting the set of high-performing attack patterns and vulnerabilities listed in Tables 10 and 11. We then examined the generation time series

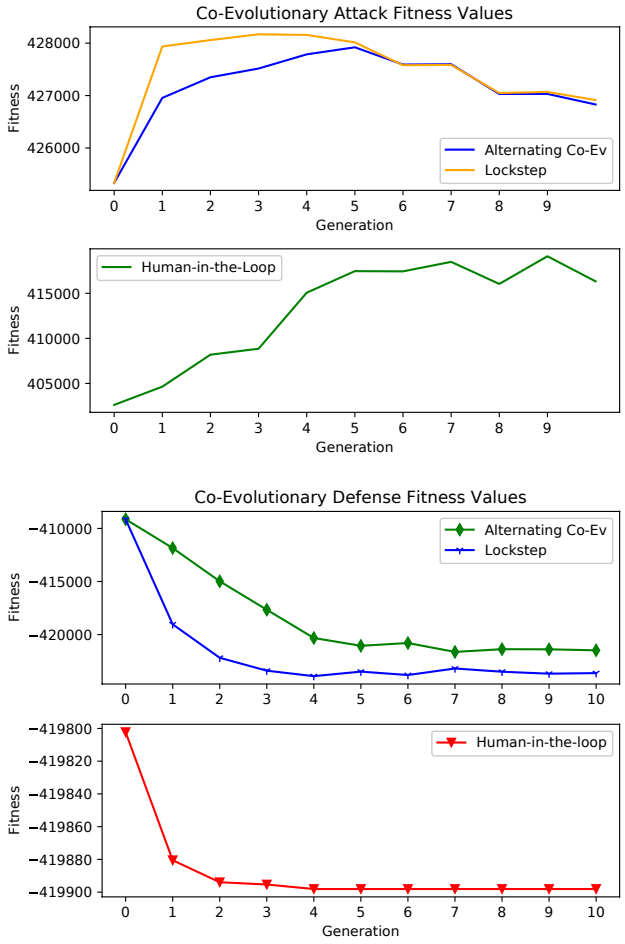

Figure 3: Fitness values over time during the coevolutionary searches.

for 100 runs with these clipped search spaces using Coev as our algorithm variant. The results, with the name CLIPPED in Figure 3, are shown alongside runs of LS_Coev and Coev with the larger search spaces. The clipped runs lie within a much narrower fitness band than the others'. Narrowing of the search space reduced the number of poorly matched attack and defense competitions. What likely led to a larger variance in fitness scores in the larger search spaces was removed by clipping the spaces and hence the flatter line.

We also examined the details of the mitigations and discovered that both LS_Coev and Coev mitigated a set of common operating systems and applications. Note that this could not be discerned from the node identifiers in the individuals because software is very commonly similar among nodes in our networks.

5.6 Discussion

To summarize at a higher level, comparisons between LS_Coev and Coev reveal differences that imply each adversarial population is sensitive to the timing between when its starts to react to its adversary and for how long this adaptation occurs. All comparisons show that the attack patterns that evolve i.e. which dominate, see Table 11, and the effectiveness of the mitigations that evolve, see Table 5, on population bases, change in distinctive ways. When the competition within the objective shifts, as with Recent_Coev, we see the outcome of a simplified experiment around attention and the consequences of where it focuses mitigation and attack: if the entire population shifts its collective "attention" to a single adversary, the cost in terms of lost out-of-sample robustness is high. This points the way forward to developing simulations with more population variance, perhaps expressed by different networks for each defense or by simulating some cooperation among defenses.

5.6.1 Limitations. We should state limitations of different facets of **EvoAPT**:

- Competition-wise, **EvoAPT** investigates only two networks that a) were acquired in an ad-hoc way and b) are the same for all defenses. This limits fidelity around mitigation and attack pattern distributions across evolved populations.
- Using the CVSS for attacks provides a limited model of what benefit attacks gain when they succeed. **EvoAPT** does not model the value of exfiltrating or enabling a future AP critical to a APT goal. In addition, a general critique of CVSS is that the actual risk to an organization might not be accurately reflected by this value.
- Algorithmically: True to evolutionary algorithm design principles, the variants use blind mutation. However this implies **EvoAPT** is not informed by the information structure of attack patterns.
- APT steps were modeled as an attack pattern list and one vulnerability in lieu of different steps actual APTs may use.

6 CONCLUSIONS & FUTURE WORK

Because APTs evolve with the security community's defenses, our goal has been to model this phenomena with coevolutionary algorithms. One challenge was to develop a competition space that abstractly, yet accurately, expresses the actions of the two adversaries: APTs and the security community. We addressed it by drawing upon public, descriptions of attack patterns and software that has needed mitigations. Another challenge was to align variants of coevolutionary algorithms with evolutionary dynamics observable in the wild. We addressed it with conventionally alternating, lockstep, and recency-driven, competitive coevolutionary algorithms. Our comparative study shows that the way a defensive population preferentially acts, e.g. with a shifting set of recent attack patterns as an objective, or reacting to all attack patterns more slowly than an attack population, results in different evolutionary outcomes, expressed as different dominant attack patterns.

There are more coevolutionary algorithm variants that we would like to explore, e.g. those using archives. We also plan future work that will use data science techniques to more explicitly and quantitatively document APT evolution. We also will consider how our findings can more directly extend the current use cases of attack patterns, see [15].

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] 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)*. IEEE, 1–8.

[2] Chia-Mei Chen, Gu-Hsin Lai, and Dan-Wei Marian Wen. 2018. Evolution of Advanced Persistent Threat (APT) Attacks and Actors. In *International Computer Symposium*. Springer, 76–81.

[3] Sen Chen, Minhui Xue, Lingling Fan, Shuang Hao, Lihua Xu, Haojin Zhu, and Bo Li. 2018. Automated poisoning attacks and defenses in malware detection systems: An adversarial machine learning approach. *computers & security* 73 (2018), 326–344.

[4] FireEye. 2020. Highly Evasive Attacker Leverages SolarWinds Supply Chain to Compromise Multiple Global Victims With SUNBURST Backdoor. https://www.fireeye.com/blog/threat-research/2020/12/evasive-attacker-leverages-solarwinds-supply-chain-compromises-with-sunburst-backdoor.html.

[5] Dennis Garcia, Anthony Erb Lugo, Erik Hemberg, and Una-May O'Reilly. 2017. Investigating Coevolutionary Archive Based Genetic Algorithms on Cyber Defense Networks. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion* (Berlin, Germany) *(GECCO '17)*. ACM, New York, NY, USA, 1455–1462. https://doi.org/10.1145/3067695.3076081

[6] Erik Hemberg, Jonathan Kelly, Michal Shlapentokh-Rothman, Bryn Reinstadler, Katherine Xu, Nick Rutar, and Una-May O'Reilly. 2020. BRON–Linking Attack Tactics, Techniques, and Patterns with Defensive Weaknesses, Vulnerabilities and Affected Platform Configurations. *arXiv preprint arXiv:2010.00533* (2020).

[7] Tasmina Islam, Ingolf Becker, Rebecca Posner, Paul Ekblom, Michael McGuire, Hervé Borrion, and Shujun Li. 2019. A socio-technical and co-evolutionary framework for reducing human-related risks in cyber security and cybercrime ecosystems. In *International Conference on Dependability in Sensor, Cloud, and Big Data Systems and Applications*. Springer, 277–293.

[8] Jon Baker. [n.d.]. CALDERA Pathfinder. https://medium.com/mitre-engenuity/caldera-pathfinder-7564e63f3082 https://medium.com/mitre-engenuity/caldera-pathfinder-7564e63f3082.

[9] Jonathan Kelly, Michael DeLaus, Erik Hemberg, and Una-May O'Reilly. 2019. Adversarially adapting deceptive views and reconnaissance scans on a software defined network. In *2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*. IEEE, 49–54.

[10] Elmar Kiesling, Andreas Ekelhart, Kabul Kurniawan, and Fajar Ekaputra. 2019. The SEPSES Knowledge Graph: An Integrated Resource for Cybersecurity. In *International Semantic Web Conference*. Springer, 198–214.

[11] Bin Liu, Xixi Zhu, Junfeng Wu, and Li Yao. 2020. Rule Reduction after Knowledge Graph Mining for Cyber Situational Awareness Analysis. *Procedia Computer Science* 176 (2020), 22–30.

[12] MITRE. [n.d.]. Common Attack Pattern Enumeration and Classification. https://capec.mitre.org/

[13] MITRE. [n.d.]. Common Vulnerabilities and Exposure. https://cve.mitre.org/

[14] MITRE. [n.d.]. Common Weakness Enumeration. https://cwe.mitre.org/

[15] MITRE. [n.d.]. Summary of CAPEC use cases. https://capec.mitre.org/about/use_cases.html

[16] NIST. [n.d.]. Common Platform Enumeration. https://nvd.nist.gov/products/cpe

[17] NIST. [n.d.]. Common Vulnerability Scoring System. https://nvd.nist.gov/vuln-metrics/cvss

[18] Umara Noor, Zahid Anwar, Asad Waqar Malik, Sharifullah Khan, and Shahzad Saleem. 2019. A machine learning framework for investigating data breaches based on semantic analysis of adversary's attack patterns in threat intelligence repositories. *Future Generation Computer Systems* 95 (2019), 467–487.

[19] Marek Ostaszewski, Franciszek Seredynski, and Pascal Bouvry. 2007. Coevolutionary-based mechanisms for network anomaly detection. *Journal of Mathematical Modelling and Algorithms* 6, 3 (2007), 411–431.

[20] 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 (2020), 219–250.

[21] Elena Popovici, Anthony Bucci, R Paul Wiegand, and Edwin D De Jong. 2012. Coevolutionary Principles.

[22] George Rush, Daniel R. Tauritz, and Alexander D. Kent. 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* (Madrid, Spain) *(GECCO Companion '15)*. ACM, New York, NY, USA, 859–866. https://doi.org/10.1145/2739482.2768429

[23] Conor Ryan, J.J. Collins, Jj Collins, and Michael O'Neill. 1998. Grammatical Evolution: Evolving Programs for an Arbitrary Language. In *Lecture Notes in Computer Science 1391, Proceedings of the First European Workshop on Genetic Programming*. Springer-Verlag, 83–95.

[24] 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.

[25] Michal Shlapentokh-Rothman, Erik Hemberg, and Una-May O'Reilly. 2020. Securing the software defined perimeter with evolutionary co-optimization. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion*. 1528–1536.

[26] Sophos. [n.d.]. How SunBurst malware does defense evasion. https://news.sophos.com/en-us/2020/12/21/how-sunburst-malware-does-defense-evasion

[27] Ann Thorhauer and Franz Rothlauf. 2014. On the locality of standard search operators in grammatical evolution. In *International Conference on Parallel Problem Solving from Nature*. Springer, 465–475.

[28] Zachary Wilkins, Ibrahim Zincir, and Nur Zincir-Heywood. 2020. Exploring an artificial arms race for malware detection. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion*. 1537–1545.

[29] Hongbo Xiao, Zhenchang Xing, Xiaohong Li, and Hao Guo. 2019. Embedding and Predicting Software Security Entity Relationships: A Knowledge Graph Based Approach. In *Neural Information Processing*, Tom Gedeon, Kok Wai Wong, and Minho Lee (Eds.). Springer International Publishing, Cham, 50–63.