Adversarial Co-evolution of Attack and Defense in a Segmented Computer Network Environment

Erik Hemberg
MIT, CSAIL
Cambridge, Massachusetts
hembergerik@csail.mit.edu

Joseph R. Zipkin
MIT Lincoln Laboratory
Lexington, Massachusetts
joseph.zipkin@ll.mit.edu

Richard W. Skowyra
MIT Lincoln Laboratory
Lexington, Massachusetts
richard.skowyra@ll.mit.edu

Neal Wagner
MIT Lincoln Laboratory
Lexington, Massachusetts
neal.wagner@ll.mit.edu

Una-May O'Reilly
MIT, CSAIL
Cambridge, Massachusetts
unamay@csail.mit.edu

ABSTRACT

In computer security, guidance is slim on how to prioritize or configure the many available defensive measures, when guidance is available at all. We show how a competitive co-evolutionary algorithm framework can identify defensive configurations that are effective against a range of attackers. We consider *network segmentation*, a widely recommended defensive strategy, deployed against the threat of serial network security attacks that delay the mission of the network's operator. We employ a simulation model to investigate the effectiveness over time of different defensive strategies against different attack strategies. For a set of four network topologies, we generate strong availability attack patterns that were not identified *a priori*. Then, by combining the simulation with a co-evolutionary algorithm to explore the adversaries' action spaces, we identify effective configurations that minimize mission delay when facing the attacks. The novel application of co-evolutionary computation to enterprise network security represents a step toward course-of-action determination that is robust to responses by intelligent adversaries.[1]

CCS CONCEPTS

• **Theory of computation** → **Evolutionary algorithms**; • **Networks** → Network reliability;

KEYWORDS

cybersecurity, co-evolution, network, evolutionary algorithms

[1]DISTRIBUTION STATEMENT A. Approved for public release. Distribution is unlimited.

This material is based upon work supported by the Assistant Secretary of Defense for Research and Engineering under Air Force Contract No. FA8721-05-C-0002 and/or FA8702-15-D-0001. Any opinions, findings, conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Assistant Secretary of Defense for Research and Engineering.

GECCO '18 Companion, July 15–19, 2018, Kyoto, Japan

ACM ISBN 978-1-4503-5764-7/18/07…$15.00
https://doi.org/10.1145/3205651.3208287

ACM Reference Format:
Erik Hemberg, Joseph R. Zipkin, Richard W. Skowyra, Neal Wagner, and Una-May O'Reilly. 2018. Adversarial Co-evolution of Attack and Defense in a Segmented Computer Network Environment. In *GECCO '18 Companion: Genetic and Evolutionary Computation Conference Companion, July 15–19, 2018, Kyoto, Japan.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3205651.3208287

1 INTRODUCTION

Network security is a serious and growing problem [1]. Frequently, attackers introduce malware into networks to achieve any number of malicious purposes, including stealing data or turning the network's operation to their own ends. Once an attacker has compromised a device on a network, they can often move laterally to connected devices, akin to contagion. Depending on the attacker's intent, the effects can be devastating. In 2012, an attacker penetrated the network of Saudi Aramco, the national oil company of Saudi Arabia. The attacker moved laterally to spread malware to as many as 30,000 network devices, which were rendered not only inoperable but unrecoverable [3].

In this work, we assume a network supports an enterprise in carrying out its business or *mission*, and an adversary employs *availability attacks* against the network to disrupt this mission. Specifically, the attacker starts by using an exploit to compromise a vulnerable device on the network. This inflicts a mission delay when a mission critical device (mission device) is infected. Then, the attacker moves laterally to compromise additional devices and maximally delay the mission; see Figure 1.

We examine a defensive measure called *network segmentation*, which divides the network topologically into *enclaves* that serve as isolation units to deter inter-enclave contagion. A network enclave is a section that is partitioned from the rest of the network. Enclaves can vary in size, position, and even vulnerability to attack due to the software services they host. Network segmentation design is a tradeoff space: a more segmented network provides less mission efficiency because of increased overhead in inter-enclave communication. However, smaller enclaves contain compromise by limiting the spread rate, and their cleansing incurs fewer mission delays. Network operators can also use monitoring capabilities and network cleansing policies to detect and disentrench attackers, but more rigorous applications of these measures also come with increased costs. Network monitoring is often performed using taps,

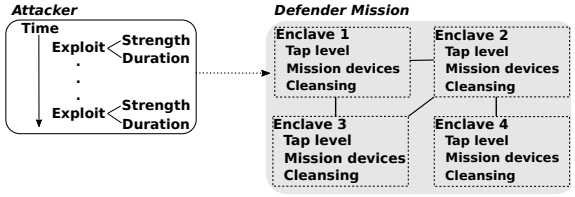


Figure 1: The attacker uses a series of propagating exploits with varying strength and step-by-step duration to delay the defender's mission. The defender network is partitioned into enclaves with varying vulnerability levels. Each enclave has some share of mission devices, a monitoring tap and the ability to be cleansed of malware.

a physically dedicated hardware device, with varying sensitivity depending on configuration and monitoring organization.

Our high-level goal is a methodology to identify the most effective defender parameters (enclave topology, device-to-enclave placement and monitoring strategy) against serial propagating malware attacks. Our system, named **AVAIL**, has two components: a simulation evaluated in a segmented network environment, and a competitive co-evolutionary framework with fitness evaluated by the simulation. The simulation component, see Alg. 1, evaluates the parameters for the defender and gradual contagion spread rates for the attacker. The co-evolutionary component seeks strategies for the defender and attacker that aim to, respectively, minimize and maximize mission delay against all adversary strategies.

Course-of-action recommendation in computer security is often hampered by failure to account for the adversary's intelligence and freedom of action [14]. **AVAIL** uses co-evolution to simulate an attacker adapting to the defender's decisions. Decision makers can consult its results to guide their selection of a robust topology and accompanying enclave configuration. Depending on the parameters with which **AVAIL**'s co-evolutionary algorithm has been run, they can compare results in terms of mean expected utility, best-worst-case utility, or even a Pareto front trading off utility with different attacks. Here, **AVAIL** returns mean expected mission delay.

The major contributions of this work are as follows: (1) We show how to model the dynamics of network segmentation and availability attacks with a competitive co-evolutionary algorithm and simulation model. **AVAIL**'s co-evolutionary algorithm a) represents serial propagating attacks that are proceeding in steps and are characterized by the contagion strength and duration, b) represents enclave configurations by tap sensitivity and mission device shares, and c) optimizes a minimax mission delay fitness objective. (2) We conduct an empirical study to demonstrate how to generate *a priori* unknown attacks and assess the performance of an enclave defense against them. (3) Our study optimizes the best configuration defense under the metric of expected utility, but our experiments could be repeated with other metrics, e.g. best-worst-case delay. (4) Our application of co-evolutionary algorithms to enterprise network security decision-making is novel.

In Sec. 2 we present related work. Sec. 3 introduces the network simulator and the co-evolutionary algorithm. Sec. 4 details the experiments and analyzes them. Sec. 5 concludes.

2 RELATED WORK

We describe network segmentation in Sec. 2.1, network security modeling and simulation using evolutionary algorithms in Sec. 2.2, and evolutionary algorithms for network security in Sec. 2.3.

2.1 Network segmentation

Network defense measures vary by what network layer they work on, their presumed threat model and what strategy they realize [9, 11]. One such category of defenses is *network segmentation*, the practice of introducing heterogeneity into network topology and connectivity. Two related network-segmentation strategies suggested in [11] are *segregating networks and functions* (SNF) and *limiting workstation-to-workstation communication* (LWC). Prior work has found both these strategies effective in containing network attacks [20]. SNF partitions a network into enclaves based on the constituent resources' functional requirements and attendant security risks. Under this strategy, for example, devices hosting private personnel data would be placed in a separate, more protected enclave than devices hosting public-facing Web services, which are vulnerable to attack due to ease of access to their interfaces. Implementation tools include firewalls, network egress and ingress filters, application-level filters, and hardware [6]. LWC regulates network communications more granularly than SNF, essentially placing each device into its own enclave, toward enforcing the *principle of least privilege* [20], a computer-security heuristic stating that "every program and every user of [a] system should operate using the least set of privileges necessary to complete the job" [16]. Between SNF and LWC are a range of possible variations on network segmentation at various levels of granularity.

2.2 Security Modeling and Simulation

Modeling and simulation comprise a powerful approach, "*mod-sim*", to investigating general security scenarios [17] and computer security in particular [8, 18, 22]. Mod-sim is often necessary because search and outcome spaces are too complex for analytical solution, but testbeds can have long experimental cycle times and require wasted attention to irrelevant detail. Mod-sim systems range in complexity, level of abstraction and resolution.

The mod-sim system for security risk assessment of network segmentation architectures (NSM) of Wagner *et al.* [20] is the most closely related to **AVAIL**, though more complex and controlled by multiple realistically chosen parameters. It models low-level network functionality and investigates abstract functions of a network being segregated according to the principle of least privilege. We distill **AVAIL**'s enclave vulnerability parameter values from the Wagner *et al.* NSM study [19], which models changes in network security levels with a continuous-time Markov chain with parameters set by actual captured network data. In contrast to NSM, **AVAIL** has a more abstract and simpler network model and mission simulation and a different threat model; see Sec. 3.1. We refer to the network segmentation security assessment model of Wagner *et al.* as NSM.

2.3 Co-evolutionary Search Algorithms

Co-evolutionary algorithms explore domains in which the quality of a candidate *solution* is determined by its ability to successfully pass some set of *tests*. Reciprocally, a *test*'s quality is determined by

its ability to force errors from some set of *solutions*. For example, in **AVAIL**, candidate enclave configurations, i.e. *solutions*, are successful to the extent that they minimize mission delay against availability attack strategies, i.e. tests. Attack strategies are successful if they maximize mission delay. In competitive co-evolution the search is an arms race with adversarial interaction between *test* and *solution*, both evolving with opposite objectives [13].

Selection pressure in an co-evolutionary algorithm is different from that in an evolutionary algorithm because, in the former, a solution's performance is contextual, i.e. depends on what tests it has faced. This implies that co-evolutionary algorithms may not only face the problem of local optima but also encounter problematic dynamics where *tests* are unable to put pressure on *solutions* to improve, or drive toward a solution that is the *a priori* intended goal. This issue has been addressed in co-evolutionary literature, and there are some suggested *remedies* to specific pathologies [2, 4, 13]. They embody a general approach that includes maintaining diversity to generate a search gradient and using more explicit memory, e.g. an *Hall of Fame* an archive to monitor progress [10].

AVAIL uses Grammatical Evolution (GE), an algorithm that has been paired with competitive co-evolution, e.g., in [7]. **AVAIL** bears similarity to CANDLES [15], which models high-level attack and defense strategies using high-level representations of their behaviors. It supports a qualitative analysis. **AVAIL** is also similar to RIVALS [5], which uses competitive co-evolution and adapts abstract denial-of-service attack strategies against a mission executing on a peer-to-peer network. Both RIVALS and **AVAIL** incorporate an availability measurement into both adversaries' objectives. In contrast to RIVALS and CANDLES, **AVAIL** (1) focuses on network segmentation as a defensive measure with enclave cleansing as local mitigation, (2) assumes a serial, propagating availability attack threat model, and (3) executes with a defensive enclave vulnerability parameter that inherits a degree of validation from the actual captured network data upon which it is based.

3 METHODS

Table 1 presents the notation we use in describing our methods. In Sec. 3.1 we present **AVAIL**'s threat model. In Sec. 3.2 we present its enclave network simulation. In Sec. 3.3 we describe its competitive co-evolutionary framework.

3.1 Threat Model: Availability Attacks

Our threat model assumes attacks that diminish the availability of mission devices. The attacker starts by infiltrating each enclave and compromising a single device. The attack then spreads after some duration and attempts to compromise other devices within the enclave. Compromised mission devices contribute to mission delay. All events (actions and their outcomes) are probabilistic.

3.1.1 Defense Model. We use the following:

- The defender's objective is to minimize mission delay.
- Only some devices, the *mission devices*, contribute to the mission. The defender partitions a quantity of mission devices among enclaves at outset of the mission.
- The defender deploys in each enclave a *tap*, a monitoring tool to detect device compromise. This tap alerts upon successful detection depending on the spread of compromise within the enclave and its *sensitivity level*. At high sensitivity it can possibly alert even though no mission devices are compromised (a *false positive*).
- A tap alert, whether a true positive or a false positive, triggers an enclave *cleansing*. Cleansing takes down and reboots all devices in the enclave simultaneously, removing the compromise but increasing mission delay.

Table 1: Notation for AVAIL network enclave simulator.

Symbol	Description
N	Number devices, including mission devices
M	Number of *mission* devices; partitioned over enclaves by defender
T	Simulation time, $T \in \mathbb{T}$
m	Mission delay, $m \in \mathbb{N}$, $m \leq T$
$\mathbf{E}$	The list of enclaves, $\mathbf{E} = \{e_0, \ldots, e_k\}$, $k \in \mathbb{N}$
e	An enclave; it is the tuple $e = \langle \mathbf{D}, v, \beta, s, n \rangle$
$\mathbf{D}$.all	A list of all devices, $\mathbf{D} = \{d^0, \ldots, d^{n-1}\}$
$\mathbf{D}$.compromised	compromised devices list
$\mathbf{D}$.uncompromised	uncompromised devices list
v	Vulnerability of enclave, $v \in [0 \ldots 1]$
β	Contagion rate, $\beta \in \mathbb{R}$, $\beta \geq 0$
B	Attacker contagion rate budget $B \geq \beta$
s	Tap sensitivity of enclave; $s \in \mathbb{R}$, $s > 0$
$I(t)$	Number of infected devices at time t, $\mathrm{I} \in \mathbb{N}$, $I \geq 0$
n	Number of devices in enclave, $n \in \mathbb{N}$

3.1.2 Attack Model. We use the following:

- The attacker's objective is to maximize mission delay.
- The attacker trades off the goal to compromise more devices with the need to evade detection, which becomes more likely as more devices are compromised. Thus, after compromise, an attack may wait before spreading.
- An attack expends effort, i.e. requires strength, to overpower a device's defensive measures. Attack strength is a fixed budget resource that is divided evenly between each step of the attack and across enclaves.
- Successful compromise depends on attack strength and enclave vulnerability.
- The attack spreads without cost but does not know where mission devices are and cannot specifically target them.
- Attack spread follows an epidemic model [20] based on a "dumb" worm [23]; see Eq. (1).

3.2 Enclave Network Simulation

We provide all simulation software under the MIT License[2]. It should be consulted for precise design and implementation details.

The key simulation inputs are a network topology of enclaves, the vulnerability levels of each enclave, and the attack and defense genomes passed by the co-evolutionary framework. Alg. 1 INITIALIZE decodes the defense genome to set up tap sensitivity and partition mission devices across the enclaves and the attack genome to schedule when attacks spread and the strength of each attack step.

[2]The link to the code repository will be made available upon acceptance.

Alg. 2 SIMULATE is called after initialization. Each time step, every enclave is updated for additional compromise, checked for compromise detection, and cleansed and reset if there is a detection. Then, mission delay is tallied. Function DETECT-COMPROMISE refers to tap sensitivity and the count of compromised devices to probabilistically determine whether an alert occurs. Function SPREAD-MALWARE increments the compromised devices. Spread occurs at rate β following the logistic function in Eq. (1). The initial stage of growth of compromised devices is approximately exponential. As saturation sets in, the growth slows and it progresses towards maturity. The number of infected devices, $I : \mathbb{N} \rightarrow \mathbb{R}, I(t)$, is modeled by

$$I(t) = \frac{nI_0 e^{\beta t}}{n + I_0(e^{\beta t} - 1)}. \quad (1)$$

Here, $t \in \mathbb{N}$ is time, $I(0)$ is number of infected devices at time 0, $n \in \mathbb{N}$ is number of devices and $\beta > 0$ is growth rate, i.e. an indicator of attack strength. We use $I(0) = 1$.

Algorithm 1 Network and Enclave initialization

Require: **E**, set of enclaves; v, vulnerability of each enclave; attack plan and defense config
function INITIALIZE(v)
  $m \leftarrow 0$; mission delay
  for $e \in$ **E do**
    $e.D.all \leftarrow$ UniformlyInsert(non-mission devices)
    $e.v \leftarrow$ vulnerability ▷ From simulation initialization
    $e.D.all \leftarrow$ Insert(mission-devices) ▷ From defense configuration
    $e.s \leftarrow$ tap sensitivity ▷ From defense configuration
    RESET-ENCLAVE(D)
  SCHEDULE-CONTAGION(β, $duration$) ▷ From attack plan
  $m \leftarrow$ SIMULATE(**E**, T)

function RESET-ENCLAVE(**D**)
  shuffle(**D**.all)
  D.compromised $\leftarrow \emptyset$
  D.uncompromised $\leftarrow$ **D**.all

Algorithm 2 Enclave network simulation

Require: **E**, enclaves; T, Simulation time, v, Vulnerability, s, Tap sensitivity
function SIMULATE(**E**, T)
  for $t \ldots T$ **do**
    for $e \in$ **E do** ▷ For each enclave
      SPREAD-MALWARE(v, $e.D$) ▷ infection spread
      needs-cleansing = DETECT-COMPROMISE(e.**D**.compromised, s)
      if needs-cleansing **then**
        CLEANSE-ENCLAVE($e.D.all$,m)
        RESET-ENCLAVE(e.**D**)
      else
        $m \leftarrow$ UPDATE-MISSION-DELAY(e.**D**.compromised)

function DETECT-COMPROMISE(compromised, s)
  threshold $\leftarrow$ |compromised| $* s$ ▷ Sensitivity of tap and compromised devices
  return random(0.0 . . . n) < threshold ▷ Is tap triggered

function CLEANSE-ENCLAVE(D, m)
  for $d \in D$ **do**
    if $d \in$ mission-device-set **then**
      $m \leftarrow$ UPDATE-MISSION-DELAY()
  cleanse(D) ▷ Cleanse the devices

function SPREAD-MALWARE(v, **D**, β, t)
  if (random() < v) **then**
    $I \leftarrow \left\lfloor \frac{ne^{\beta t}}{n+(e^{\beta t}-1)} \right\rfloor$ ▷ Total compromised devices
    Δ I $\leftarrow I -$ |**D**.compromised| ▷ Newly compromised devices
    Insert(**D**.compromised,Remove(**D**.Uncompromised,ΔI))

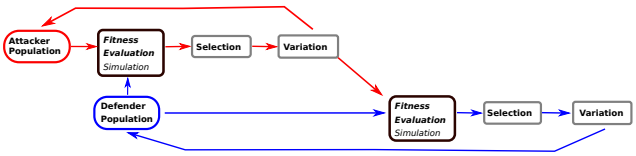


Figure 2: Alternating co-evolutionary algorithm.

3.3 Co-evolutionary Framework

The co-evolutionary framework, see Figure 2, consists of (1) attack and defensive configuration representations, (2) a competitive co-evolutionary algorithm, and (3) an interface to the simulation.

AVAIL uses grammars to represent attacks and defensive configurations; see Figure 3 and 4. Grammars facilitate the expression and exploration of complex attack sequences and defender strategies. It also eases the incorporation of domain knowledge into future work. Specifically, **AVAIL** uses Grammatical Evolution (GE) to define its genomes method. GE uses a variable-length integer representation [12] with low locality operators [21].

AVAIL evolves two populations with tournament selection and standard crossover and mutation. One population comprises attacks and the other defenses. Each generation, competitions are formed by pairing attack and defense. The populations are evolved in alternating steps: first the attacker population performs selection, variation, and evaluation against the defenders and replacement, and then the defender population performs selection, variation, and evaluation against the attacker population and replacement. Each attacker–defender pair is dispatched to the simulation, which returns the mission delay resulting from the attacker's attack on the defender's network configuration. Mission delay is then converted directly into attack fitness and negated for defender fitness. Each attacker is competed against all defenders one at a time and each defender against all attackers one at a time.

4 EXPERIMENTAL SETUP

4.1 Co-evolutionary Framework Setup

The experiments are designed to create a baseline and investigate the performance of evolutionary search and co-evolutionary search on a network segmentation problem. For this study comparing evolutionary algorithms is out of scope, and is deferred to future work. Table 2 presents the co-evolutionary algorithm parameters. The GE grammars used here are foremost a proof-of-concept and provide am agile and flexible platform. In addition, the grammars are simple to avoid the locality issues related to GE [21]. Moreover, the GE implementation does not use any wrapping and implements caching to reduce the number of redundant fitness evaluations. The fitness is calculated as the average of the mission delay, m, returned by the simulator, see Alg. 2. The attacker objective is to maximize the mission delay, and the defender objective is to minimize it.

AVAIL's attack grammar enables an attacker to explore many short and weak attacks, fewer strong attacks, or a mixture. Strength and duration of an attack are subject to the attacker's overall budget; see Figure 3. The grammar uses start symbol `<attack_plan>`. Note, `<attack_plan>` rule is recursive so it can vary in length.

Table 2: Co-evolutionary GA parameters

Parameter	Value
Population size	30
Generations	30
Crossover Probability	0.8
Mutation Probability	0.1
Max Length	100
Tournament Size	2
Elite size	1
Attack competitions	30
Defender competitions	30
Attacker objective	maximize mission delay
Defender objective	minimize mission delay

Table 3: AVAIL simulation parameters

Parameter	Value
Number of non-mission devices (N)	250
Number of mission devices (M)	10
Attacker Exploit Strength budgets (B)	0.5, 1.25, 2.0
Attacker Time budget	100
Defender tap level range	$[0, \ldots, 1.5]$
Defender enclaves	1, 2, 4, 16 (see Table 4)
Experiment	**Setup**
GA-DEFENSE	Fixed Attacker, Evolved Defender
GA-ATTACK	Fixed Defender, Evolved Attacker
COEV	Co-evolving Defender and Attacker

Figure 3: Attacker grammar

```
<attack_plan>  ::= <attack> | <attack> <attack_plan>
<attack>  ::= (<strength_share>, <time_share>)
<strength_share> ::= 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10
<time_share>     ::= 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10
```

Figure 4: Defender for grammar for E2

```
<config> ::= <tap_sensitivity>, <num_mission_devices>;
             <tap_sensitivity>, <num_mission_devices>;
<tap_sensitivity> ::= 0.00 | 0.01 |...| 1.50
<num_mission_devices> ::= 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10
```

Table 4: Experimental enclave vulnerabilities. Each element in the list is the vulnerability level of an enclave.

Name	|E|	Vulnerability levels v
E1	1	0.954
E2	2	[0.954, 0.44]
E4	4	[0.48, 0.762, 0.827, 0.762]
E16	16	[0.761, 0.762, 0.828, 0.387, 0.393, 0.387, 0.397, 0.416, 0.419, 0.241, 0.239, 0.235, 0.241, 0.243, 0.245, 0.321]

A decoded <attack_plan> consists of one or more attacks each of duration τ and strength β. The <time_share> genes are decoded with reference to each other and the total simulation time. Shares are summed and the duration of each attack is its share over the sum times the total time. The <strength_share> genes are decoded with reference to each other and the total attack budget B. Shares are summed and the strength of each attack is its share over the sum times the budget. For example, given attacker time budget $\mathsf{T} = 100$, attack budget $B = 2.0$, an attacker with genome $(1, 3), (1, 2)$ decodes to $\beta = 1.0$ for $t = [0, \ldots, 60]$ and $\beta = 1.0$ for $t = [41, \ldots, 100]$.

The defender grammar enables the defender to partition the mission devices in quantity across the network's enclaves and set the sensitivity s of the taps; see Figure 4. The defender does not need an explicit budget since the mission delay is influenced by how much cleansing is done; this depends in turn on the <tap_sensitivity>. The grammar start symbol is <config>. It sets up a fixed-length genome with two configuration properties for each enclave. An example of a two-enclave configuration is shown in Figure 4. At decoding, mission devices are grouped in enclaves in enclave index order. For example, if enclave 1 of 4 receives 10 devices, there are no mission devices in the other enclaves. If there are mission devices unassigned after decoding, they are distributed uniformly over the enclaves. Non-mission devices are assigned to enclaves uniformly.

4.2 Experimental Procedure

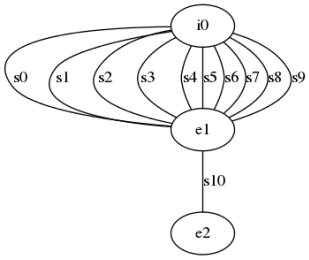
Our scenarios, see Table 3, are based on a network with $N = 250$ devices including $M = 10$ mission devices. An experimental scenario is {topology, vulnerability levels, attack duration budget, attack strength budget}. We study 4 possible enclave topologies: Figure 5, $|e| \in \{1, 2, 4, 16\}$. We run the NSM [19] framework over multiple trials with varying input and use the trials' average vulnerability for each enclave to set up scenario enclave vulnerability levels, see Table 4. The parameters for NSM are derived from historical security data [19] to strength the realism of the mod-sim. The attack duration budget is always $T = 100$ and the strength budget, B is a value from: $\{0.5, 1.25, 2.0\}$. For brevity we refer to the topologies as E1, E2, E4, E16, refer to the vulnerability levels as non-uniform (see Table 4), omit T and only include B when contextually necessary. We do 30 independent runs for each experiment. In each fitness evaluation in each experiment, we run 10 independent trials of the network simulator to average the simulation values over. The simulation runs for $T = 100$ time steps.

We conduct three different experiments and report results that are the average of 30 runs, except for **COEV** where we show a randomly selected run that is consistent with the others:

GA-DEFENSE: We use GE to evolve defenders against each of 5 different non-evolving attacks, see Table 5 with $B = 2.0$. We compare the 4 enclave topologies each with enclave vulnerabilities, see Table 4, thus the scenarios are {E1|E2|E4|E16, 100, 2.0}

GA-ATTACK:We use GE to evolve attacks against a non-evolving defense with uniform tap levels and device sharing across enclaves. We compare the 4 enclave topologies with enclave vulnerabilities, thus the scenarios are {E1|E2|E4|E16, 100, 0.5|1.25|2.0}

COEV: Co-evolved Attacks and Defenses with GE. We use **AVAIL** to co-evolve the attacks and defenses in response to each other's populations with $B = 2.0$. We compare the 4 enclave topologies with enclave vulnerabilities, see Table 4, thus the scenarios are {E1|E2|E4|E16, 100, 2.0}



(a) 2 enclaves **(b) 4 enclaves** **(c) 16 enclaves $e_1, \ldots, e_{16}$**

Figure 5: Experimental enclave topologies, i_0 is an Internet connected enclave, s* are services (connections) between enclaves.

Table 5: Static attack plan, budget $T = 100$, $B = 2.0$. Attack <strength_share> (decoded to β) and <time_share> per attack

Name	<strength_share>	<time_share>
A5-I	[1, 2, 3, 4, 5]	[1, 1, 1, 1, 1]
A1	[1]	[1]
A2-I	[1, 3]	[1, 2]
A10	[1, 1, 1, 1, 1, 1, 1, 1, 1, 1]	[1, 1, 1, 1, 1, 1, 1, 1, 1, 1]

4.3 Experiments and Results

4.3.1 Evolving Attackers. We start our investigation with the **GA-ATTACK** experiment regarding size of enclave. Here our research question is the relative merit of each enclave size. We find the greatest mission delay each enclave size would suffer by using the GA to optimize and identify the strongest attack. Recall that the attacker's budget is evenly divided per enclave before the per enclave budget is then divided across an attack according its <strength_share>.

Based on the threat model and the known efficacy of network segmentation [20], we would expect E1, the single enclave, to perform poorly and suffer from the most mission delay while only requiring one attack. Per Figure 6 and Table 6, this is confirmed. We would also expect the benefit of additional enclaves to increase to some critical size after which they no longer are helpful because devices are so few per enclave that the first step of any attack compromises them. Finding this size is challenging due to additional factors such as the relative number of mission devices to non-mission ones and the attack strength. We observe that mission delay decreases with enclave size growing from E1 to E16 with the exception that, for attack strength budgets $B = 0.5$ and $B = 0.125$, the mission delay E4 suffers is less than that of the mission delay E16 suffers – E2 (46.8 to 81.8) for attack budgets (0.5, 1.25, 2.0), E4 (6.5, 14.3, 26.9) and E16 (16.8, 16.7, 16.9). This highlights a critical threshold in sizing. If a network expects attacks of low strength, i.e. is able to bound its expectation of the attacker's resources below the amount equivalent to $B = 0.125$, 4 enclaves are better. But when attacks of high strength are anticipated, i.e. the attacker's resources are more than $B = 1.25$, 16 enclaves are better. With 4 enclaves, mission devices can be hidden well enough among non-mission devices so that, for weak attacks, they resist compromise very well. We experimented with E=5,...,16 and confirmed the critical threshold

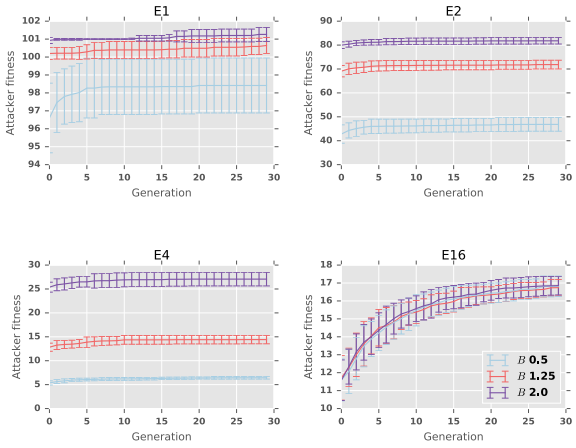


Figure 6: Evolving attacks against different network architectures. Best solution is shown. X-axis shows generation and Y-axis shows fitness value. Each line shows a different attacker strength budget (B)

at E=4. These findings are consistent with the acknowledgment in computer security that the design of defensive measures must be dictated by the expected strengths of attackers: a nation–state threat actor requires different approaches than "script kiddies" out for monetary gain or enjoyment.

With 16 enclaves, both the 240 non-mission and 10 mission devices are sparsely distributed across the enclaves. In these sparse enclaves they are more susceptible to attack, regardless of strength. This is rational: there is only 1 or 0 mission devices per enclave, so propagating an attack is not as effective at delaying the mission (when each attack is 5 time steps) as simply re-infecting the enclave with greater odds of infecting a mission device immediately. With 16 enclaves, an effective attack appears to have a plan of approximately 17 attacks.

In Table 6 we see a difference in the number of enclaves that the best attacker selects for attack plans of more attacks, both based on the strength budget and on the number of enclaves. Against E1 a higher budget gives more attacks. Conversely, against E4 a lower

Table 6: GA-ATTACK: Evolved attacker solutions (best per run).

Experiment	Fitness (Mean±Std)	Number of attacks (Mean±Std)	<time_share> (Mean±Std)
E1 B=0.5	98.451 ± 1.470	1.000 ± 0.000	5.161 ± 2.411
E1 B=1.25	100.582 ± 0.459	1.677 ± 0.467	5.404 ± 3.398
E1 B=2.0	101.254 ± 0.386	2.355 ± 0.478	5.452 ± 3.184
E2 B=0.5	46.825 ± 2.849	1.000 ± 0.000	5.226 ± 2.837
E2 B=1.2	71.840 ± 1.821	1.000 ± 0.000	5.871 ± 2.970
E2 B=2.0	81.782 ± 1.336	1.000 ± 0.000	5.097 ± 2.798
E4 B=0.5	6.485 ± 0.293	5.710 ± 2.035	6.107 ± 2.598
E4 B=1.25	14.312 ± 0.955	1.000 ± 0.000	6.065 ± 2.782
E4 B=2.0	26.909 ± 1.418	1.000 ± 0.000	6.419 ± 2.916
E16 B=0.5	16.816 ± 0.571	17.742 ± 3.162	5.744 ± 2.724
E16 B=1.25	16.742 ± 0.460	17.484 ± 3.732	5.631 ± 2.695
E16 B=2.0	16.856 ± 0.519	17.258 ± 2.615	5.536 ± 2.654

budget gives more attacks. There is no noticeable difference in the average duration of any attack step.

4.3.2 Evolving Defenders. We proceed to the **GA-DEFENSE** experiment. Here our research question is still the relative merit of each enclave size, but we use the GA to optimize enclave configurations (mission device placement and tap sensitivity) under attacks that vary in how many are in a plan. Recall that the attacker's budget is evenly divided per enclave before the per enclave budget is then divided across an attack according its genome's <strength_share> genes. All attackers have the same budget regardless of how many attacks they have in plan, so those with longer plan launch weaker attacks. Mission delay results are shown in Figure 7. Table 7 shows the same data in detail numerically.

As we expect, a single-enclave configuration is highly susceptible to mission delay. A single attack, A1, delays the mission significantly more (mission delay = 86.2) than attacks in a plan with A10, due to attack weakness and the large number of devices in the enclave, hardly inflicting any delay at all (mission delay = 1.5). Scanning Table 7 column-wise, the same trend is seen for 2 and 4 enclaves. However for 16 enclaves, there is an inflection at A5 (5 attacks in a plan). With 10 attacks in a plan, the inflicted mission delay is higher than with 5.

As the number of enclaves increases, the effectiveness of a single attack, A1, drops (Mission delays are E1 = 86.2, E2 = 9.3, E4 = 4.7, E16 = 2.7). This trend is the same for two attacks in a plan, A2. In contrast, both A5 and A10 have an inflection point at two enclaves. They drop in effectiveness from one to two enclaves but increase in effectiveness when defended by E4 and E16. One can see this by scanning Table 7 row by row.

We examined the best defensive configurations across enclave sizes and attack sizes. They are easiest to decipher at E2, E4 with E16 being generally too complex. With E2 and E4, we generally see that the defenders evolve to place mission devices in enclaves of low vulnerability and they cleanse these enclaves more frequently (i.e. they have a greater tap sensitivity in enclaves of low vulnerability) or at least the same frequency as the other enclaves. This is rational because of the need to protect the mission devices from compromise. The results are consistent with sound security guidance to practice active management of enterprise network resources.

Table 7: GA-DEFENSE: Evolved defender fitness (-Mission Delay) best per run.

A/D	E1 (Mean±Std)	E2 (Mean±Std)	E4 (Mean±Std)	E16 (Mean±Std)
A1	−86.2 ± 1.764	−9.3 ± 0.611	−4.7 ± 1.163	−2.7 ± 0.128
A2	−69.6 ± 1.748	−6.0 ± 0.591	−3.2 ± 0.407	−4.1 ± 0.226
A5	−22.4 ± 1.098	−1.1 ± 0.168	−1.9 ± 0.240	−6.5 ± 0.238
A10	−1.5 ± 0.206	−0.4 ± 0.069	−1.6 ± 0.207	−7.1 ± 0.317

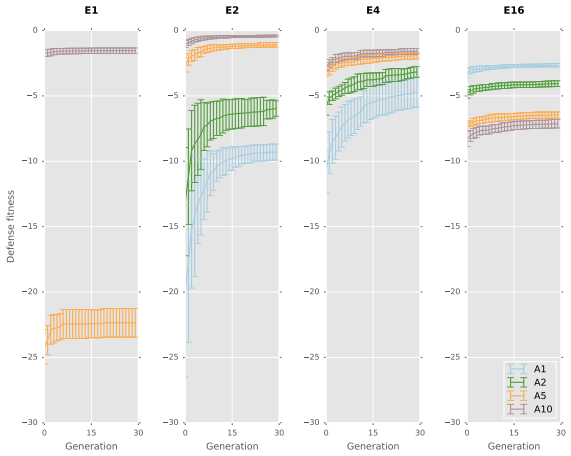
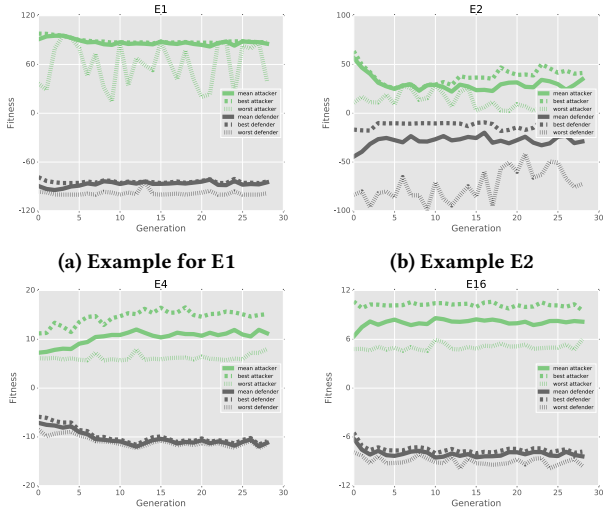


Figure 7: Evolving defense against different network architectures. Best fitness of defender over generations strength $B = 2.0$. Each line shows a different attacker. Defender fitness is cut off at −30, see Table 7 for all defender fitness values at last generation.

We can also frame a best-worst-case question: What is the best enclave size given the worst-case mission delay of any enclave? In this case, E1, E2, E4 all are most hurt by a single attack and E16 by a plan of 10 attacks, and E4 has the lowest mission delay. Therefore, against all attacks in this set, a good choice would be 4 enclaves. This confirms the intuition that some but not too many enclaves are advantageous. This question can be posed also from the attacker's perspective: what is the ideal number of attacks to launch in a plan given the least mission delay of each attack plan? The data indicates that, if the attacker didn't know enclave sizes, it should launch a plan of 2 attacks because the best-worst-case mission delay is 3.2.

4.3.3 Co-evolving attackers and Defenders. To end, we investigate **COEV** where, repeatedly, first the defenders test and evolve then the attackers respond to the adapted defenders and evolve themselves. This is a more realistic view of the adversarial arms race. Results for best, population mean and worst fitness from an example run are shown in Figure 8.

For one enclave we see that the best attacker and best defender fitness track the population mean fitnesses. The mission delay oscillates at 90 ± 5. This is higher than when defenders, **GA-DEFENSE**, can evolve to minimize a non-evolving attack (for A1, A2, A5 and



(a) Example for E1 **(b) Example E2**

(c) Example for E4 **(d) Example E16**

Figure 8: Example from co-evolution showing best, worst and mean fitness of defender and attacker solution over generations strength $B = 2.0$. Green denotes attacker and black denotes defender. Solid line shows mean fitness, wide dashes shows best fitness and narrow dashes shows worst fitness.

A10), reflecting adaptability of attackers in response to defenders. It is lower than when attackers, **GA-ATTACK** can evolve against a non-evolving defender, this time reflecting the adaptability of defenders in response to attackers. This observation extends to enclave sizes 2, 4, and 16. The results are intuitive. An evolving population performs better against a static adversary than against an adversary that can itself evolve in reaction. In addition, the population aspect of evolutionary and co-evolutionary algorithms impacts the general robustness of adversaries' strategies because one strategy is "competed" against many of its adversary's strategies. In summary, the expectations from the co-evolutionary experiments and understanding the upper-bound, lower-bound, and best-worst-case scenarios of the experiments are all helpful in supporting deployment decisions and estimations of risk.

5 CONCLUSIONS AND FUTURE WORK

We have focused on one particular network security measure, network segmentation. We combined mod-sim and competitive co-evolutionary algorithms into a system that allows different enclave topologies to be compared, each optimized to minimize mission delay by the evolutionary adaptation of its defensive configuration against attack strategies optimized to maximize mission delay. We also examined evolution of attackers and defenders separately when their adversary did not evolve. We were able to evaluate mission delay under realistic enclave vulnerability levels that we obtained from the high fidelity mod-sim NSM of [19]. This novel application of co-evolutionary computation to enterprise network security represents a step toward course-of-action determination that is robust to responses by intelligent adversaries. Future work on the competitive co-evolutionary framework will extend it with run-result archiving, Hall of Fame analysis and visualizations to improve user decision support. Future security-oriented work will investigate alternative threat models with increased adversarial complexity, as well as larger networks. Future work could also include validation via testbed emulation or live experiment with users to support our results.

REFERENCES

[1] Akamai Technologies. 2017. State of the Internet quarterly security reports. (2017). https://www.akamai.com/us/en/about/our-thinking/state-of-the-internet-report/global-state-of-the-internet-security-ddos-attack-reports.jsp

[2] Josh C Bongard and Hod Lipson. 2005. Nonlinear system identification using coevolution of models and tests. *IEEE Transactions on Evolutionary Computation* 9, 4 (2005), 361–384.

[3] Christopher Bronk and Eneken Tikk-Ringas. 2013. The cyber attack on Saudi Aramco. *Survival* 55, 2 (2013), 81–96.

[4] Sevan Gregory Ficici. 2004. *Solution concepts in coevolutionary algorithms*. Ph.D. Dissertation. Citeseer.

[5] D. Garcia, A. Erb Lugo, E. Hemberg, and U. O'Reilly. 2017. Investigating Coevolutionary Archive Based Genetic Algorithms on Cyber Defense Networks. In *Proceedings of the 19th Annual Conference on Genetic and Evolutionary Computation (GECCO '17)*. ACM, 8. https://doi.org/10.475/1234

[6] Robert Gezelter. 2015. E-commerce and Web server safeguards. In *Computer Security Handbook* (6th ed.), Seymour Bosworth, Michel E. Kalbay, and Eric Whyne (Eds.). Wiley.

[7] Robin Harper. 2014. Evolving robocode tanks for Evo robocode. *Genetic Programming and Evolvable Machines* 15, 4 (2014), 403–431.

[8] Mona Lange, Alexander Kott, Noam Ben-Asher, Wim Mees, Nazife Baykal, Cristian-Mihai Vidu, Matteo Merialdo, Marek Malowidzki, and Bhopinder Madahar. 2017. Recommendations for Model-Driven Paradigms for Integrated Approaches to Cyber Defense. *arXiv preprint arXiv:1703.03306* (2017).

[9] Stuart McClure, Joel Scambray, and George Kurtz. 2009. Hacking exposed: network security secrets and solutions. (2009).

[10] Thomas Miconi. 2009. Why coevolution doesnâĂŹt "work": superiority and progress in coevolution. In *European Conference on Genetic Programming*. Springer Berlin Heidelberg, 49–60.

[11] National Security Agency Information Assurance Directorate. 2013. IAD's top 10 information assurance mitigation strategies. (2013).

[12] Michael O'Neill and Conor Ryan. 2003. *Grammatical evolution: evolutionary automatic programming in an arbitrary language*. Vol. 4. Springer.

[13] Elena Popovici, Anthony Bucci, R Paul Wiegand, and Edwin D De Jong. 2012. Coevolutionary principles. In *Handbook of Natural Computing*. Springer, 987–1033.

[14] Antonio Roque. 2018. Validating computer security models. *arXiv preprint arXiv:1710.01367* (2018).

[15] 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 on Genetic and Evolutionary Computation Conference*. ACM, 859–866.

[16] Jerome H. Saltzer and Michael D. Schroeder. 1975. The protection of information in computer systems. *Proc. IEEE* 63, 9 (1975), 1278–1308.

[17] Milind Tambe (Ed.). 2012. *Security and Game Theory: Algorithms, Deployed Systems, Lessons Learned*. Cambridge University Press.

[18] Brian Thompson, James Morris-King, and Hasan Cam. 2016. Controlling risk of data exfiltration in cyber networks due to stealthy propagating malware. In *Military Communications Conference, MILCOM 2016-2016 IEEE*. IEEE, 479–484.

[19] Neal Wagner, Cem Şafak Şahin, Jaime Pena, James Riordan, and Sebastian Neumayer. 2017. Capturing the security effects of network segmentation via a continuous-time Markov chain model. In *Proceedings of the 50th Annual Simulation Symposium*. ACM.

[20] Neal Wagner, Cem Ş Şahin, Michael Winterrose, James Riordan, Diana Hanson, Jaime Peña, and William W Streilein. 2016. Quantifying the mission impact of network-level cyber defensive mitigations. *The Journal of Defense Modeling and Simulation: Applications, Methodology, Technology* (2016).

[21] Peter A Whigham, Grant Dick, James Maclaurin, and Caitlin A Owen. 2015. Examining the Best of Both Worlds of Grammatical Evolution. In *Proceedings of the 2015 on Genetic and Evolutionary Computation Conference*. ACM, 1111–1118.

[22] Michael L Winterrose and Kevin M Carter. 2014. Strategic evolution of adversaries against temporal platform diversity active cyber defenses. In *Proceedings of the 2014 Symposium on Agent Directed Simulation*. Society for Computer Simulation International, 9.

[23] Shui Yu, Guofei Gu, Ahmed Barnawi, Song Guo, and Ivan Stojmenovic. 2015. Malware propagation in large-scale networks. *IEEE Transactions on Knowledge and Data Engineering* 27, 1 (2015), 170–179.