Automatic Generation of Cyber Architectures Optimized for Security, Cost, and Mission Performance: A Nature-inspired Approach

Neal Wagner, Cem Ş. Şahin, Jaime Pena, and William W. Streilein
MIT Lincoln Laboratory
Lexington, MA, USA
Email: {neal.wagner, cem.sahin, jdpena, wws}@ll.mit.edu

***Abstract*—Network segmentation refers to the practice of partitioning a computer network into multiple segments and restricting communications between segments to inhibit a cyber attacker's ability to move and spread infection. While segmentation is widely recommended by cyber security experts, there is no clear guidance on what segmentation architectures are best to maximize a network's security posture. Additionally, the security gained by segmentation does not come without cost. Segmentation architectures require resources to implement and may also cause degradation of mission performance. Network administrators currently rely on judgment to construct segmentation architectures that maximize security while minimizing resource cost and mission degradation. This paper proposes an automated method for generating segmentation architectures optimized for security, cost, and mission performance. The method employs a hybrid approach that combines nature-inspired optimization with cyber risk modeling and simulation to construct candidate architectures, evaluate them, and intelligently search the space of possible architectures to hone in on effective ones. We implement the method in a prototype decision system and demonstrate the system via a case study on a representative network environment under cyber attack.**

I. Introduction

Network segmentation (NS) is a defensive mitigation technique designed to reduce the damage due to cyber attack. Its goal is to limit attacker access to and movement within a network by partitioning the network into multiple segments or enclaves and restricting communications between enclaves and between enclaves and the Internet. This partitioning is typically implemented by the use of firewalls, network egress and ingress filters, application-level filters, and/or physical (hardware) infrastructure [1]. NS is widely regarded as critical for network security [2]–[5] but is poorly understood with only vague guidance (e.g., [3], [6]) on how to apply it. For even small networks many different NS architectures are possible and the number of possibilities grows exponentially with network size.

The problem is compounded by the fact that security does not come without cost. Segmentation architectures require resources to implement and maintain and also may cause degradation of mission performance. Network administrators must select architectures that maximize security posture while minimizing resource cost and mission degradation. Currently, administrators are forced to rely on judgment to balance tradeoffs between security, cost, and mission performance. A further compounding factor is that for many enterprise networks the problem of mission mapping has not been solved [7], [8]. Mission mapping refers to the mapping of an organization's mission onto the cyber assets (e.g. devices/servers, software applications, communication protocols, etc.) used to execute it. Because many organizations do not know exactly what cyber assets are being used to support their mission and how they are being used, they cannot reasonably estimate the mission degradation that may result due to a given NS architecture.

This paper proposes an automated method for constructing ns architectures that are optimized for security, cost, and mission performance. The method employs a hybrid approach that combines nature-inspired optimization with cyber risk modeling and simulation to construct candidate architectures, evaluate them, and intelligently search the space of possible architectures to generate efficacious ones. The proposed method is implemented in a prototype decision system and demonstrated via a case study on a representative network environment under cyber attack. Our work addresses an important gap in the area of cyber security decision support (CSDS): the need for systems that leverage data-driven methods to generate optimal/near-optimal security decisions.

The field of CSDS is still quite young with only a handful of studies to date. Two recent studies seek to aggregate input from subject matter experts (SMEs) to address cyber threats: in [9] SME assessments are used to forecast threats and recommend security measures while in [10] SME rankings of cyber attacks and relevant security components are aggregated to provide security assessments of computer systems during the system design phase.

Another study details a cyber infrastructure to facilitate and secure individual-based decision making and negotiation for Internet-of-Things devices with respect to applications in health care [11]. In [12], a Bayesian Belief Network, cyber vulnerability assessment, and expected loss computation are combined to compute appropriate premiums for

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.

cyber insurance products while [13] utilizes game theory and combinatorial optimization to evaluate cyber security investment strategies. Finally, in [14] a decision support system to assist cyber defenders protecting mobile ad-hoc networks (MANETs) is developed to remediate malicious intrusions and reduce network energy costs.

In this paper, we build upon the work of [15] to develop an automated decision system to generate NS architectures optimized for multiple objectives. In [15] a semi-automated system for the NS decision problem is developed that optimizes for one objective only, namely security risk. Here, we present a fully-automated implementation of the system that includes optimization with respect to the three critical, and often conflicting, objectives: security, cost, and mission performance. The contributions of this paper are the following.

- We provide a method to automatically generate effective NS architectures and realize this method in a fully-automated decision system. The system utilizes a novel hybrid algorithm that combines nature-inspired optimization with cyber risk modeling and simulation.
- The system outputs NS architectures that are optimized for security, cost, and mission performance.
- We provide a mission performance model that allows for approximated measurement of mission degradation due to a given segmentation architecture when the mapping of cyber assets onto organizational mission is not available.

The rest of this paper is organized as follows: Section II provides an overview of the network segmentation defensive mitigation, Section III describes the decision system's overall design, Section IV details the security, cost, and mission performance models used to evaluate candidate architectures, Section V presents a case study that demonstrates the system for a representative network environment under cyber attack, and Section VI concludes.

II. Network Segmentation Defensive Mitigation

Network segmentation (NS), as described above, is a cyber defensive mitigation meant to inhibit an attacker's ability to move and spread infection throughout the network. NS accomplishes this by partitioning the network into multiple enclaves and restricting communications between enclaves and between enclaves and the Internet. Figs. 1(a) and (b) provide example NS architectures.

Fig. 1(a) gives an example architecture in which no partitioning is utilized. Here, the entire network (i.e. network devices, servers, routers, switches, etc.) is contained within a single enclave in which all network devices are allowed direct communication with all other network devices. Additionally, the enclave is connected to the Internet, that is it allows direct communication to the Internet where cyber attackers preside. Communications between the network enclave and the Internet occur via one or more software services (e.g. web browsers, email, ssh, etc.) running on the enclave. In the figure, X software services allow this communication.

In Fig. 1(a), a cyber attacker may exploit vulnerabilities present in one or more of the software services to penetrate the network enclave. Common strategies for penetration include sending phishing emails with infected attachments or links to infected websites to entice a network user to open the attachment or browse to the infected website and, thus allow the attacker to infect her device and gain a foothold in the network. Once inside, the attacker can easily move and spread infection to other network devices because the infected device is allowed to directly communicate with all other network devices.

The example given by Fig. 1(b) is a NS architecture that is partitioned into four enclaves, three that are allowed direct communication with the Internet (Enclaves #1-3 in the figure) and a fourth that is allowed direct communications to the other three enclaves but not to the Internet (Enclave #4 in the figure). Note also that Enclaves #1-3 do not have direct communications with each other. Communications from Enclaves #1-3 to the Internet occur via the same set of X software services (split up over the three enclaves) as given in Fig. 1(a). An additional software service provides communication from these Internet-facing enclaves to Enclave #4.

For the architecture of Fig. 1(b), an attacker may also be able to penetrate the network and gain access to one of Enclaves #1-3. However, upon penetration the attacker's ability to spread infection to other network devices is hampered by the communication restrictions. In order for the attacker to infect a device in Enclave #4, she must exploit an additional software service, the one that allows communication with that enclave. To spread to other Internet-facing enclaves, she must also exploit additional services. The restrictions created by the architecture serve to slow down the rate at which the attacker can spread infection and allows the defender more time to patch vulnerable software services and/or cleanse infected enclave devices.

A NS architecture selected by the defender consists of the following components.

- A set of network enclaves. An enclave is defined as a group of devices with homogeneous reachability.
- Software services that allow communications between enclaves and between enclaves and the Internet.
- The rate at which software services are patched.
- The rate at which enclaves are cleansed. Enclave cleansing is a process by which an enclave's devices are cleansed to remove infection and restored to their original state. The result of enclave cleansing is to dis-entrench an attacker who has penetrated.

III. Automatic Generation of NS Architectures: Decision System Design

Extending the work presented in [15], we have designed an automatic method for generating NS architectures that are optimized for security, cost, and mission performance. The proposed method is realized as a fully-automated decision system. The system inputs parameters that characterize the network environment and cyber threat and outputs an effective architecture for that environment.

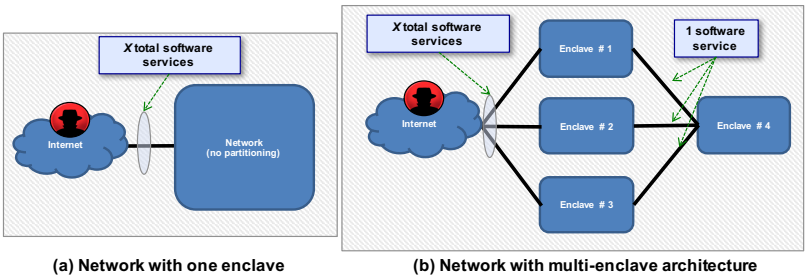

Fig. 1. Two example network segmentation architectures: (a) a network with no partitioning and (b) a network partitioned into four enclaves, three that allow direct communications to the Internet and a fourth enclave that allows direct communications to the other three network enclaves but not to the Internet.

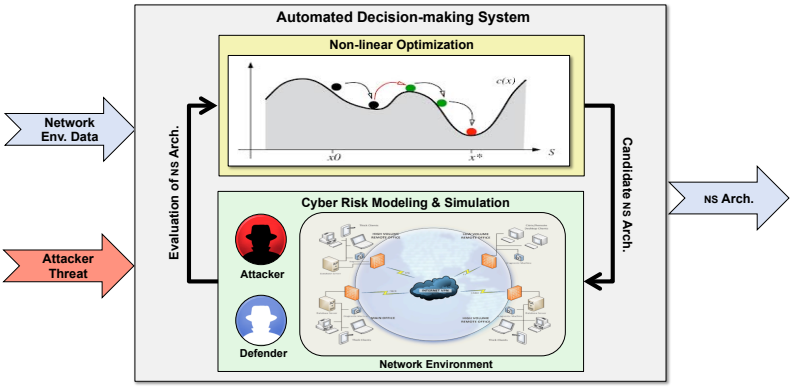

Fig. 2. High-level design of the decision system to automatically generate NS architectures optimized for security, cost, and mission performance.

Fig. 2 gives a graphical depiction of the decision system. In the figure, the system is represented by the gray box. System inputs (blue and red inflowing arrows to the left of the figure) are used to characterize the network environment including its security posture and existing cyber threat. System output (blue arrow to the right of the figure) is an optimized NS architecture for the given network environment. The decision system is comprised of two algorithmic components, non-linear optimization and cyber risk modeling and simulation (mod/sim) (depicted by the yellow and green boxes inside the gray box in the figure). The optimization component suggests candidate NS architectures while the mod/sim component evaluates these architectures. These two components run iteratively: the optimization component suggests a candidate architecture, it is evaluated via the mod/sim component, and the evaluation is then fed back to the optimization component where it is utilized to guide its search to construct newer, more promising candidate architectures. Together these components work to search the space of possible NS architectures and hone in on effective architectures.

As discussed in Section I, the number of possible NS architectures grows exponentially with network size and thus deterministic search methods are intractable for all but the smallest networks. Here, we employ soft computing algorithms inspired by nature to execute the function of the optimization component. The current version of the system utilizes a Simulated Annealing (SA) algorithm to explore the space of NS architectures.

SA is an adaptation of the Metropolis-Hastings Monte Carlo method for approximating the global optimum of a given function. It is inspired by the process of annealing in metallurgy [16]. Generally, the algorithm starts with an initial solution, generates a new solution that is a neighbor to it in the decision problem search space, evaluates both the old and new solutions against a given objective function, probabilistically accepts the new solution in place of the old solution, and then repeats these steps (using the currently accepted solution) for some number of iterations. The algorithm mimics the annealing process by initially having a relatively higher probability of accepting a new solution that is inferior to the currently accepted solution and progressively lowering that probability at each iteration. Note that if a new solution is superior to the currently accepted solution, it is always accepted.

Alg. 1 specifies the SA algorithm for the NS decision system. In the algorithm, the objective is to minimize the combined risk to security, cost, and mission performance, and thus lower values of the evaluation function $eval$ (lines 8,9 of the algorithm) are superior. Computation of the evaluation function is detailed in Section IV below.

In Alg. 1, the function Generate-Neighbor (line 7 of the algorithm) is used to generate a new candidate solution architecture that is a "neighbor" to the currently accepted solution architecture. Recall from Section II that a NS architecture includes a set of enclaves, the software services that allow communications, and the rates at which software patching and enclave cleansing occur. A new architecture is generated by

Algorithm 1 Optimize-NS-Architecture(s_0, k_{max})

{s_0: initial segmentation architecture, k_{max}: max. no. of iterations}
$s \leftarrow s_0$ {Accept s_0 as current solution}
$s_{best} \leftarrow s$ {Save the best solution found so far}
$k \leftarrow k_{max}$
repeat
  $T \leftarrow \frac{k}{k_{max}}$ {Set temperature T}
  $s_{new} \leftarrow$ Generate-Neighbor(s)
  $eval(s)$ {Evaluate s, s_{new}}
  $eval(s_{new})$
  if $eval(s_{new}) < eval(s)$ **then**
    $s \leftarrow s_{new}$ {Accept s_{new} if superior to s (lower risk)}
    if $eval(s_{new}) < eval(s_{best})$ **then**
      $s_{best} \leftarrow s_{new}$ {Save the best solution found so far}
    end if
  else
    $r \leftarrow$ random value $\in [0, 1]$ {Probabilistically accept inferior s_{new}}
    if $r \leq \frac{e^{\frac{(1-eval(s_{new})-eval(s))}{T}}}{e^{(1/T)-1}}$ **then**
      $s \leftarrow s_{new}$
    end if
  end if
  $k \leftarrow k - 1$ {Decrement k to reduce T}
until $k = 0$
return s_{best}

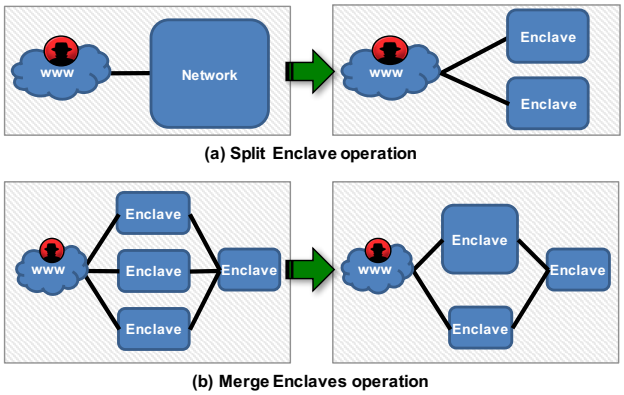

Fig. 3. Generating a new NS architecture by changing the number of enclaves: (a) the Split Enclaves operation to increase the number of enclaves and (b) the Merge Enclaves operation to decrease the number of enclaves.

altering the current architecture in one of the following ways.

- Increasing or decreasing the number of enclaves.
- Adding or removing software services allowing communication between two enclaves or altering a software service to change one of its endpoints (i.e. change its source or destination enclave).
- Changing the software patching rate or the enclave cleansing rate.

Figs. 3(a) and (b) illustrate operations that increase/decrease the number of enclaves. Fig. 3(a) depicts the Split Enclaves operation. Here, the number of enclaves is increased by splitting an existing enclave into two or more enclaves. Fig. 3(b) depicts the Merge Enclaves operation: the number of enclaves is decreased by merging two or more enclaves into a single enclave.

The operations described above are used to generate new NS architectures that are then evaluated by the mod/sim component. This combination of generation and evaluation serves to drive the decision system to explore promising

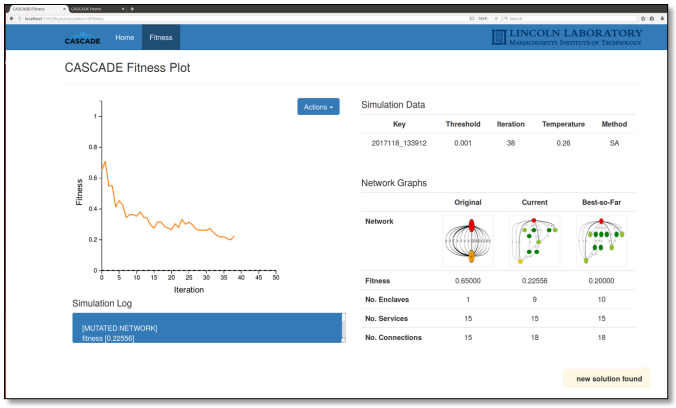

Fig. 4. Screen capture of the user interface of the prototype decision system.

areas of the search space and automatically construct effective architectures. The decision system is implemented using a combination of Scala X.X, Python Y.Y, and JavaScript Z.Z.

Fig. 4 shows a screen capture of the user interface of the prototype system. The plot to the left of the figure gives the fitness value of the current solution architecture over several iterations of the SA algorithm and the three graphs to the right of the figure give graphical representations of the starting, current, and best-so-far architectures generated during the run. Note that Fig. 4 provides an illustrative example of the prototype system interface; results shown do not correspond to experiments discussed in this paper. Section IV below details the system's fitness function while Section V describes architectures generated during experimentation and their graphical representations.

IV. NS Architecture Evaluation Model

The decision system described in the previous section requires an evaluation function to measure the effectiveness of generated solution architectures. As discussed in Section I, we wish to generate architectures that are optimized for security, cost, and mission performance. The following sections detail the security, cost, and mission performance models used for evaluation.

A. Measuring Security Risk

We utilize a continuous-time Markov chain model (CTM) developed in [17], [18] to measure the security inherent to a NS architecture. The CTM characterizes security risk for a given network environment under threat from external attackers whose goal is to penetrate and spread throughout the network. The model captures changes in network state due to the arrival of new software vulnerabilities, patches, exploits, and the communications allowed within a given NS architecture. Arrivals of these security-related events are modeled as Poisson processes, and thus transitions between states are characterized by sampling rates in which intervals between event samples are exponentially distributed with a given mean. Here, we use Poisson processes to capture attackers arriving to exploit vulnerabilities and defenders arriving to remediate

vulnerabilities and/or cleanse infected enclaves as used in [19]. We will refer to these sampling rates as simply Poisson rates.

The model consists of three entities: (i) a network environment, (ii) one or more network enclaves (i.e. groups of devices with homogeneous reachability), and (iii) one or more software services. A network environment is characterized by a set of enclaves and communication pathways that connect these enclaves. Communication pathways represent *functional information flows* (FIF) which include physical connections (i.e. by a physical line), transitive connections (e.g., a server enclave being able to communicate with the Internet even though no direct line exists), and more complex flows (e.g., an email is sent from the Internet that arrives at a DMZ enclave, is pulled by an exchange server and is finally downloaded and read by a user in a LAN enclave). A FIF is modeled only as a communication pathway from a source enclave to a destination enclave. Intermediate enclaves between the source and destination are not modeled. Note that the Internet is also modeled as a single enclave in which we assume the attacker presides and, thus is always compromised.

A FIF connecting two enclaves is enabled by one or more services (i.e. software applications) running on the destination enclave that are subject to vulnerabilities that may be exploited by an attacker and patches that remediate these vulnerabilities. Figs. 5 (a) and (b) depict the Markov process states for an individual service and enclave, respectively.

As given in Fig. 5(a), a service is characterized by states reflecting its current number of vulnerabilities (0, 1, 2, $\cdots$) and whether or not attackers have developed an exploit for one or more of these vulnerabilities. To capture transitions between states we utilize three Poisson rates: the vulnerability arrival rate Δ_{vuln}^{-1}, the exploit development rate Δ_{dev}^{-1}, and the patch rate Δ_{patch}^{-1}. A service starts out in the 0-vulnerability state, transitions to higher-vulnerability states with the arrival of vulnerabilities, transitions to the exploit developed state upon the development of 1 or more exploits for its existing vulnerabilities, and finally, transitions back to the 0-vulnerability state upon a patching event. While the service is in the exploit developed state, it can be exploited by the attacker. Exploit events are captured by the exploit deployment rate $\Delta_{exploit}^{-1}$.

Fig. 5(b) shows the Markov states for an enclave. An enclave is characterized by two states, a clean state and a compromised state. An enclave starts out in the clean state, transitions to a compromised state with the arrival of exploit events for one or more of its software services, and then transitions back to a clean state upon the arrival of an enclave cleansing event.

The enclave cleansing rate Δ_{clean}^{-1} is a directly-specified parameter while the enclave compromise rate Δ_{comp}^{-1} is specified as a function of the Markov processes governing the states of one or more services running on the enclave. Services running on an enclave are independent and Δ_{comp}^{-1} is computed as

$$\Delta_{comp}^{-1} = \sum_{i=1}^{N} \Delta_{comp}^{-1}(s_i) \tag{1}$$

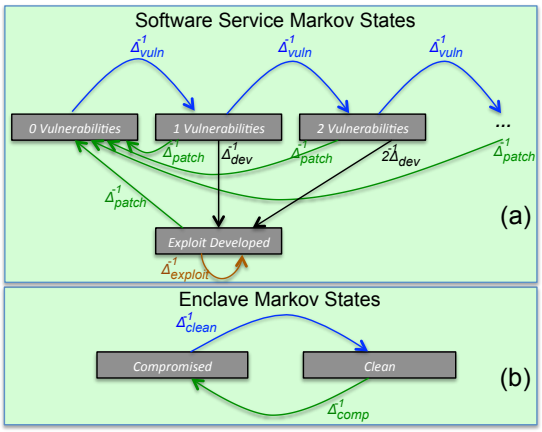

Fig. 5. (a) Markov process states for a software service represent how many vulnerabilities are present and whether an exploit for any of the vulnerabilities has been developed since it was last patched. (b) Markov states for an enclave represent whether or not the enclave has been penetrated by an attacker since it was last cleansed.

where s_i represents i^{th} service, $i \in [1, N]$, running on a given enclave E, N is the total number of available services on E and $\Delta_{comp}^{-1}(s_i)$ is the compromise rate of service s_i. Note that the compromise rate for a single service $\Delta_{comp}^{-1}(s_i)$ is captured via the Markov process model depicted in Fig. 5(a).

An event-based simulation is used to compute the security risk for a given NS architecture. The simulation computes, for each enclave in a given architecture, the expected probability that the enclave is compromised (i.e. penetrated by the attacker) at any moment. A simulation run is executed by the following steps. (1) A given NS architecture is instantiated with its enclaves, their services, and the communications topology that specifies which services allow communication between enclaves and between enclaves and the Internet. (2) Events are generated via the above-described Poisson rates. A run is terminated when it reaches a specified maximum number of simulated time units.

The security risk for a NS architecture as a whole is measured as the expected probability of enclave penetration by the attacker over all enclaves,

$$Sec(env, s) = \frac{1}{|encls(s)|} \sum_{e \in encls(s)} P_{penetrate}(e) \tag{2}$$

where env is a network environment under cyber threat, s is a segmentation architecture, $encls(s)$ is the set of all enclaves in s, e represents a single enclave and varies over all enclaves of $encls(s)$, $P_{penetrate}(e)$ is the probability of attacker penetration for enclave e, and $Sec(env, s)$ is the expected probability of enclave penetration by the attacker for environment env and segmentation architecture s. This measure is normalized to $[0, 1]$ where lower values mean lower security risk.

B. Measuring Cost

Cost is characterized as an information technology (IT) maintenance cost: greater segmentation (i.e. more enclaves)

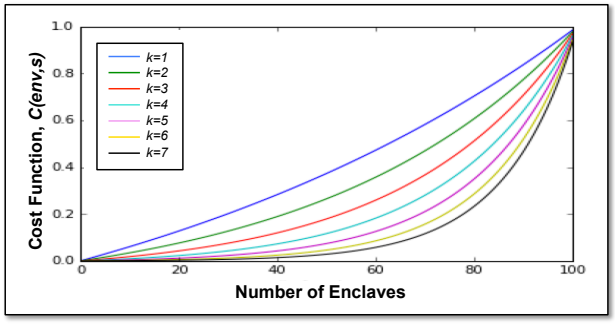

Fig. 6. IT maintenance cost function (Eq. 3) for $M = 100$ and $k = 1, \cdots, 7$.

incurs a higher IT cost to maintain. We utilize a normalized exponential function to capture the cost increase as the number of enclaves increases. The cost function is given by

$$C(env, s) = \frac{e^{(N \times k)/M} - 1}{e^k - 1} \tag{3}$$

where env is a network environment under cyber threat, s is a segmentation architecture, N is the total number of enclaves in s, M is the maximum number of enclaves that can be supported by the IT maintenance process, k is a steepness constant, and $C(env, s)$ is the IT maintenance cost for a given environment env and segmentation architecture s. This function is normalized to $[0, 1]$ where lower values mean lower cost.

Fig. 6 gives a graphical depiction of the cost function for maximum number of enclaves, $M = 100$ and several values of the steepness constant k. The cost function of eq. 3 is designed to be flexible: it can characterize a linear increase in cost with increases in the total number of enclaves, N (as shown by the blue line in the plot when $k = 1$) and it can also characterize an exponential increase in cost with increases in N (shown by the black line in the plot when $k = 7$).

C. Measuring Mission Performance

As discussed in Section I, the security gained by deploying a NS architecture may have detrimental impacts on mission performance. Because many organizations have not solved the mission mapping problem, that is they do not know exactly what cyber assets are being used to execute the mission and how they are being used, they cannot easily measure and predict the negative effects to mission performance that a given NS architecture can cause. Administrators and network architects are thus put in the difficult position of having to use subjective judgment to select a NS architecture that maximizes security posture but does not degrade mission performance to an unacceptable level.

Our goal is to construct a model to measure the *mission degradation*, the fraction of mission performance that is negatively impacted, due to a given NS architecture. The model should provide viable estimation of mission degradation for networks environments in which knowledge of mission dependencies to cyber assets is incomplete or unavailable. Towards this goal, we focus on two components of NS architecture that the defender must select: the software patching and enclave cleansing rates.

The idea is to capture mission degradation when complete knowledge of mission execution is not known. Here, we assume cleansing devices in a enclave or patching software services results in a cost to mission performance, that is the mission is degraded to some fraction of its optimum performance. When an enclave is cleansed, its devices are unavailable for some amount of time and this lack of availability negatively impacts the mission. When a software service is patched, it is unavailable for some amount of time which can also negatively impact the mission, albeit potentially to a lesser degree as patching usually takes less time than cleansing and does not necessarily require devices to be completely unavailable during the patching process. However, there exists another form of mission degradation due to software patching: that of *software disfunctionality*. New versions of software may not function exactly as older versions due to upgrades, new features, or new software bugs introduced. The altered functionality of a software application after it has been patched can cause mission operations that depend on it to execute less quickly, less accurately, and/or less completely. Regardless of what mission is being executed or whether or not that mission has been mapped to the cyber assets that support it, the base assumption is that more frequent enclave cleansing and/or software patching by the defender results in more mission degradation.

$$DA' = w_{clean} \cdot \left(\frac{\Delta_{clean}^{-1}}{\Delta_{cleanMax}^{-1}} \right) + (w_{patchTime} + w_{patchDisf}) \cdot \left(\frac{\Delta_{patch}^{-1}}{\Delta_{patchMax}^{-1}} \right) \tag{4}$$

$$DA = max \left[\left(\frac{\Delta_{clean}^{-1}}{\Delta_{cleanMax}^{-1}} \right), \left(\frac{\Delta_{patch}^{-1}}{\Delta_{patchMax}^{-1}} \right), DA' \right] \tag{5}$$

Eqs. 4 and 5 specify a mathematical model to capture the defender's activity with respect to cleansing and patching. Eq. 4 characterizes a defender's activity as a weighted average of the ratios of the rates of cleansing and patching to their maximum possible rates, respectively. In Eq. 4, Δ_{clean}^{-1} and Δ_{patch}^{-1} represent the defender rates for cleansing and patching, respectively, $\Delta_{cleanMax}^{-1}$ and $\Delta_{patchMax}^{-1}$ represent the maximum possible rates of cleansing and patching, respectively, and w_{clean}, $w_{patchTime}$, and $w_{patchDisf}$ are weights representing the relative impact that cleansing and patching have to mission degradation, respectively. Note that there are two weights associated with the impact of patching, $w_{patchTime}$, and $w_{patchDisf}$. $w_{patchTime}$ reflects the impact that patching has on mission-related cyber asset availability and $w_{patchDisf}$ represents the impact that patching has on mission-related cyber asset functionality. DA' gives the measure of cleansing and patching activity by the defender as a weighted average where $w_{clean} + w_{patchTime} + w_{patchDisf} = 1.0$ This measure is

normalized to $[0, 1]$ where higher values mean more defender activity, a value of 1 means the maximum amount of defender activity, and a value of 0 means no defender activity (i.e. the defender never cleanses or patches).

Eq. 5 computes the finalized measure of defender activity, DA, as a maximum of the individual ratios of the cleansing and patching rates to their respective maximum possible rates and DA' computed in Eq. 4. We utilize the maximum function because it is possible that one activity (either cleansing or patching) when executed at the maximum possible rate may effectively cause maximum activity and thus maximum mission degradation in and of itself. For example if the maximum possible rate of enclave cleansing is once per day and the defender chooses that rate, she may completely shut down mission operations regardless of what the patching rate is.

We utilize a normalized exponential function similar to the one given in Eq. 3 to capture the increase in mission degradation as the defender activity in cleansing and patching increases. The mission degradation function is given by

$$MD(env, s) = \frac{e^{(DA \times k)/DA_{max}} - 1}{e^k - 1} \quad (6)$$

where env is a network environment under cyber threat, s is a segmentation architecture, DA is the defender activity component of s given by Eq. 5, k is a steepness constant similar to that given in Eq. 3, and DA_{max} is the maximum possible defender activity for cleansing and patching and always has a value of 1.0. This function is normalized to $[0, 1]$ where lower values mean lower mission degradation.

D. Combined Measure of Security, Cost, and Mission Performance

The final evaluation function computes a combined measure that represents the overall risk to security, cost, and mission performance and consists of the three component measures described above. The combined risk measure is given by

$$R(env, s) = w_1 \cdot Sec(env, s) + w_2 \cdot C(env, s) + w_3 \cdot MD(env, s) \quad (7)$$

where $Sec(env, s)$ is the security risk component (Eq. 2), $C(env, s)$ is the cost component (Eq. 3), $MD(env, s)$ is the mission degradation component (Eq. 6), and w_1, w_2, and w_3 are weights in $[0, 1]$ representing the relative importance of the security, cost, mission performance component measures, respectively, subject to the constraint $w_1 + w_2 + w_3 = 1.0$. This final risk measure is normalized to $[0, 1]$ where lower values mean lower combined risk. Overall, the objective of the decision system is, for a given network environment under cyber threat env, to generate a segmentation architecture s, that minimizes $R(env, s)$ of Eq. 7.

V. Experiments

We demonstrate the system via a case study on a representative network environment under cyber attack. The aim is to use the system to improve an initial segmentation architecture to minimize $R(env, s)$ of Eq. 7 for multiple scenarios in which the relative importance of security, cost, and mission performance vary.

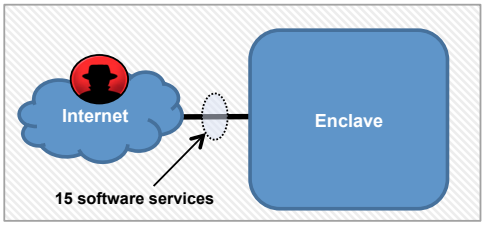

Fig. 7. Initial segmentation architecture to be improved by the decision system.

Fig. 7 shows the initial architecture, which represents an unsegmented network, that is a network with only a single enclave such that all network devices can communicate directly with all other network devices. In this single enclave network direct communications are allowed from enclave to the Internet where cyber attackers preside. Communications from the network to the Internet are made through 15 software services (applications).

The network environment is specified by parameters described in Section IV. We leverage real vulnerability, patch, and exploit data to characterize a representative software service and its associated expected rates of vulnerability arrival, patching, and exploit development using the process given in [18], [20]. Below is a summary of this process.

A. Network Environment Parameters

Within the CTM (Section IV), a software service requires specification of multiple rate parameters including (i) vulnerability arrival rate (Δ_{vuln}^{-1}), (ii) patch arrival rate (Δ_{patch}^{-1}), (iii) exploit development rate (Δ_{dev}^{-1}), and (iv) exploit occurrence rate ($\Delta_{exploit}^{-1}$). Additionally, each enclave requires specification of the enclave cleansing rate ($\Delta_{cleanse}^{-1}$). Here, the goal is to characterize a representative service.

We utilize data from the National Vulnerability Database [21] as well as results from large-scale vulnerability studies ([22]–[25]). These studies define a *vulnerability lifecycle* that captures the state of a vulnerability over time. We use this to characterize vulnerability phases including vulnerability disclosure (when the vulnerability becomes known), exploit development (when an exploit for the vulnerability is developed), exploit deployment (when the exploit is used), and patching (when a patch for the vulnerability becomes available). Figure 8 shows time dependencies between these phases. From the figure, vulnerability disclosure kicks off two processes in parallel: exploit development and patch arrival. Once an exploit has been developed for the vulnerability (and before a patch has arrived), exploits that may result in compromises can now occur for that service. Patching ends the lifecycle by rendering its exploit(s) ineffective. We use data collected with respect to these phases to compute rates for a representative service.

Fig. 8. Time dependencies between vulnerability lifecycle phases (purple arrows represent transitions between phases).

Δ_{vuln}^{-1}: To characterize the vulnerability arrival rate of a representative service, we average the most common services given in [25] as shown below:

$$\Delta_{vuln}^{-1} = \frac{\sum_{i \in N} \frac{V_i}{T}}{|N|} \tag{8}$$

where i represents the most vulnerable application for a vendor from the top vendor list derived by [25]. V_i is the weighted sum of vulnerabilities for application i over time period T where weights are given by each vulnerability's severity score [26]. N is a set containing the most vulnerable applications from top vendor list derived by [25] which collects vulnerability data over a 7-year period (2000-2007) and groups them by vendors (e.g., vulnerabilities for Microsoft products, for Adobe products, etc.). Note that Eq. 8 considers the set of all known vulnerabilities for a given software service s over a given time period, T. Using Eq. 8, we compute the expected vulnerability arrival rate, Δ_{vuln}^{-1}, as one every 65 days.

Δ_{patch}^{-1}: [23] analyzes data on nearly $15K$ vulnerabilities over a 5-year period (from 2001 to 2006) and derives vulnerability discovery dates and patch availability dates from public sources. We fit their results to a Poisson distribution and then compute a weighted average of these fitted results. Our computation yields an expected patch availability rate of one every 25 days (after vulnerability arrival). Recall from Section III that one of the choices the defender must make when specifying a segmentation architecture is the patching rate. The defender may choose to patch at a slower or faster rate and this choice will affect both the security posture and the mission performance of a given network environment. The above-computed rate represents the fastest rate at which a defender may choose to patch, that is the defender cannot apply patches to software services of a given network environment before those patches are released to the public.

Δ_{dev}^{-1}: [23] executes a similar process to derive exploit availability (exploit development) dates. This study yields exploit development rates ranging from $\approx$ 8 days before disclosure to $\approx$ 2 days after disclosure. Additionally, [25] reports that for $\approx$ 90% of vulnerabilities collected, exploits are available within 10 days of their corresponding disclosure dates. For our experiments we use the mid-point of these bounds, an expected rate of exploit development as one every 5 days (after vulnerability arrival).

$\Delta_{exploit}^{-1}$: Incident reports are generally difficult to come by because organizations do not like to share data on detected compromise events within their networks. Additionally, different kinds of exploits can be executed in varying amounts of time. Some exploits take advantage of vulnerabilities in server software (for example, the Shellshock vulnerability [27]) while others utilize phishing emails to entice users to download and open malicious file attachments. Exploits that target server vulnerabilities can be executed at any time by the attacker, but exploits that rely on phishing require the victim to trigger the exploit (e.g. by opening a malicious email attachment). We use a conservative estimation for our experiments, an expected rate of exploit deployment as one every 5 days (after exploit development).

TABLE I
NETWORK ENVIRONMENT PARAMETER SETTINGS

Parameter	Description	Setting
Δ_{vuln}^{-1}	Vulnerability Arrival Rate	1 every 65 days
$\Delta_{patchMax}^{-1}$	Max. Patch Rate	1 every 25 days
$\Delta_{patchInit}^{-1}$	Initial Patch Rate	1 every 30 days
Δ_{dev}^{-1}	Exploit Development Rate	1 every 5 days
$\Delta_{exploit}^{-1}$	Exploit Deployment Rate	1 every 5 days
$\Delta_{cleanMax}^{-1}$	Max. Enclave Cleanse Rate	1 every 7 days
$\Delta_{cleanInit}^{-1}$	Initial Enclave Cleanse Rate	1 every 365 days

The final rate to consider is the enclave cleansing rate, $\Delta_{cleanse}^{-1}$. Here, we consider the maximum rate that the defender may choose to cleanse. As discussed in Section IV, enclave cleansing usually results in greater device downtime than software patching, and thus cleansing may have greater detrimental effects on mission performance than patching does. For these experiments we assume that the maximum rate of enclave cleansing is one every 7 days. For many network environments and organizational missions, cleansing that frequently would likely make it impossible to execute mission operations.

The settings for network environment parameters are given in Table I. Parameters Δ_{vuln}^{-1}, Δ_{dev}^{-1}, and $\Delta_{exploit}^{-1}$ represent the vulnerability arrival, exploit development, and exploit deployment rates, respectively, and remain static over all experiments. Parameters $\Delta_{patchMax}^{-1}$ and $\Delta_{cleanMax}^{-1}$ represent the maximum possible rates of patching and enclave cleansing, respectively (also static over all experiments). Parameter $\Delta_{patchInit}^{-1}$ represents the initial setting for patching rate selected by the defender. Note that the patching rate Δ_{patch}^{-1} specified by a given architecture can vary over the course of a single experiment as the defender may choose slower or faster rates for patching (subject to the constraint given by $\Delta_{patchMax}^{-1}$).

Parameter $\Delta_{cleanInit}^{-1}$ in Table I represents the initial setting for cleansing rate selected by the defender. Note that like the patch rate, the cleansing rate Δ_{clean}^{-1} can vary over the course of a single experiment as the defender may choose slower or faster cleansing rates subject to the constraint given by $\Delta_{cleanMax}^{-1}$. We utilize a study that sheds light on the activities of a real cyber attacker to specify $\Delta_{cleanInit}^{-1}$. In [28] a nation-state-sponsored cyber attacker that was able to compromise hundreds of enterprise networks across many countries is documented. The study investigates the time that the attacker, code-named APT1 (Advanced Persistent Threat 1), was able to persist inside an enterprise network before

being removed/cleansed from the environment. The study reports persistence times that are up to 4 years with an average persistence time of approximately 1 year, and we use this average persistence time to set $\Delta_{cleanInit}^{-1}$.

Recall from Section IV that the mission performance component measure requires the setting of weights w_{clean}, $w_{patchTime}$, and $w_{patchDisf}$ of Eq. 4 representing the relative impact that cleansing and patching have to mission degradation, respectively. For these experiments we use $w_{clean} = 0.7$, $w_{patchTime} = 0.15$, and $w_{patchDisf} = 0.15$ to represent a network environment in which enclave cleansing causes a significant degradation of mission performance while patching causes smaller performance degradation with respect to both downtime and dis-functionality. Both the mission performance and cost component measures (Eqs. 6 and 3, respectively) also require setting the steepness constant k to capture the increase in these measures as defender cleansing/patching rates and the number of enclaves increases, respectively. For these experiments we select $k = 1.0$ for the mission performance measure, which characterizes a nearly linear rise in mission degradation when cleansing and patching rates are increased. For the cost measure we select $k = 6.0$ with maximum number of enclaves $M = 25$, which characterizes an exponential rise in cost as the number of enclaves increases to the maximum. With this setting, there are small increases in cost when the number of enclaves is less than 10 but quickly growing cost when the number of enclaves is greater.

B. Search Parameters

As mentioned above, we aim to explore how varying the relative importance of the three component objectives, security, cost, and mission performance, affect the choice of segmentation architecture by the defender. We execute several experiments in which the weights of Eq. 7, w_1, w_2, and w_3 are varied. We vary w_1, the security component measure weight, from [0,1] in increments of 0.1. For each value of w_1, we explore all possible combinations of values for w_2 (cost component measure weight) and w_3 (mission performance component weight) at increments of 0.1 subject to the constraint that $w_1 + w_2 + w_3 = 1.0$. We thus execute 66 total experiments overall with each experiment starting with the initial architecture given by Fig. 7 and consisting of 50 iterations of the decision engine.

We constrain the search to explore architectures with at least three software services allowing direct communication from a network enclave to the Internet. This constraint is used to prevent the system from generating architectures that are disconnected from the Internet as we assume that most enterprise networks require communication with external networks via the Internet. The output of each experiment is the best architecture found, that is the one with the minimum value of $R(env, s)$ from Eq. 7.

C. Results

Fig. 9 gives the results of the 66 experiments executed. Security and mission performance weightings are plotted against

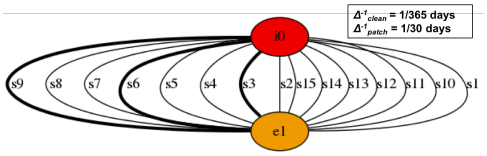

Fig. 10. Segmentation architecture generated by the system when cost weight is 1.0 and security and mission performance weights are both 0.0.

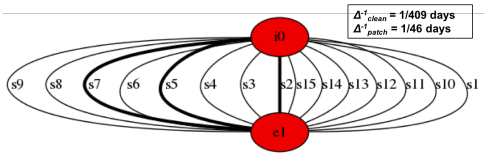

Fig. 11. Segmentation architecture generated by the system when mission performance weight is 1.0 and security and cost weights are both 0.0.

the normalized fitness of the best architecture found by the system where lower fitness values represent better solutions (i.e. less risk) with respect to the weightings used.[1] Note that the weightings for cost are also varied as described above, but these weights are not plotted in the figure for clarity, although they can easily be inferred from the weight values of the other two component measures. In the figure the surface of the result space is colored using a spectrum from red (higher risk) to green (lower risk).

From Fig. 9 we see that better fitness outputs occur when the security weight is 0.0 signifying scenarios in which all importance is placed on cost and/or mission performance and no importance is given to security. This intuitively makes sense as it is relatively easy to specify an architecture with minimal cost or minimal degradation of mission performance if security is not a concern: simply do not partition the network at all (reduces cost to its minimum value) and/or reduce rates of cleansing and patching (reduces degradation of mission performance). We also see that the worst fitness output occurs when the security weight is 1.0, meaning that all importance is placed on security and none is given to cost or mission performance. This makes sense as well: as discussed in Section I, the number of possible ways to partition a network and restrict communications between enclaves grows exponentially with network size and, thus, it is much more difficult to construct a partitioning that results in minimal security risk.

The surface of the result space in Fig. 9 consists of several peaks and valleys in the middle areas of the plot where weightings for security, cost, and mission performance are all greater than 0.0 and represent scenarios in which importance is given to all three objectives. This illustrates the inherent complexity of the tradeoffs between security, cost, and mission performance and shows that different weightings given to these objective measures can result in very different architectures.

Figs. 10-13 give graphical depictions of the best architec-

[1]Normalized fitness is computed by R/R_{worst} where R is the fitness output of the experiment (Eq. 7) and R_{worst} is the worst fitness (i.e. largest) outputted by any of the 66 experiments executed.

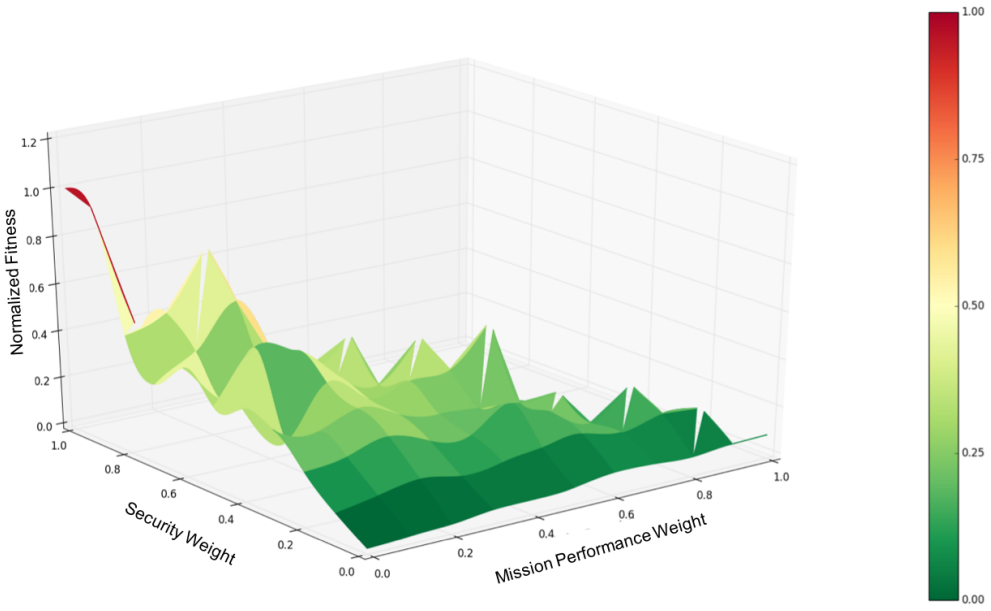

Fig. 9. Experimental results for varying weightings of security, cost, and mission performance. Security and mission performance weights are plotted against the normalized fitnesses of the best architectures found by the decision system for each experiment (cost weights are not displayed for clarity).

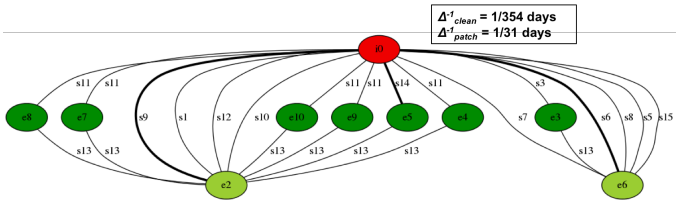

Fig. 12. Segmentation architecture generated by the system when security weight is 1.0 and cost and mission performance weights are both 0.0.

tures outputted by the system in four of the 66 experiments. In the figures, the architecture is represented as a graph in which each node represents an enclave. The topmost node (labeled $i0$) represents the Internet and the other nodes represent the enclaves of the network (labeled $e1, e2, \cdots$). Lines connecting two nodes represent software services that allow communication between enclaves or between an enclave and the Internet. The enclaves are color-coded to represent their security with colors ranging from red (very insecure) to green (secure). The Internet node is always red as that is assumed to be compromised at all times. Also shown in the figures are the cleansing and patching rates Δ_{clean}^{-1} and Δ_{patch}^{-1}, respectively, associated with the architecture.

Figs. 10 and 11 show the best architectures outputted by the system when the cost weight is 1.0 and when the mission performance weight is 1.0, respectively, and represent scenarios in which all importance is placed on either cost or mission performance and no importance is given to security. From the figures, these architectures do not use any partitioning at all, that is the system chooses a single enclave in which all network devices can communicate directly with all other devices. This follows as partitioning does not offer any benefit to either cost or mission performance.

When the cost weight is 1.0 (Fig. 10), the cleansing and patching rates are unchanged from their initial settings (1/365 days and 1/30 days, respectively). However, when the mission performance weight is 1.0 (Fig. 11), the cleansing and patching rates are reduced from their initial settings to 1/409 days and 1/46 days, respectively. This also follows as the cost measure only considers the number of enclaves present and does not consider rates of cleansing or patching while the mission performance measure does considers these rates and, thus, reduces these rates to improve mission performance. As expected the security risk component measure (Eq. 2) for both of these architectures is high: 0.670 for the architecture of Fig. 10 and 0.810 for the architecture of Fig. 11. This means that for those architectures a network enclave is compromised (i.e. penetrated by the attacker) an average of 67% and 81% of the time, respectively.

Fig. 12 shows the best architecture generated by the system when the security weight is 1.0, representing a scenario where all importance is given to security and no importance is given to either cost or mission performance. Here, the system partitions the network into 9 enclaves and increases the cleansing rate Δ_{clean}^{-1} to 1/351 days to achieve a security risk component measure of 0.165, a significant improvement in security compared to the architectures of Figs. 10 and 11. As expected the cost and mission performance component measures are not as good (i.e. higher) with cost component measure of 0.016 (compared to 0.0 for the architecture of Fig. 10) and mission performance degradation of 0.019 (compared to 0.013 for the architecture of Fig. 11).

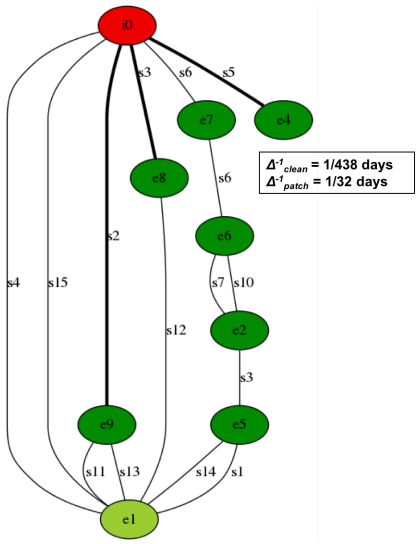

Fig. 13. Segmentation architecture generated by the system when security weight is 0.3, cost weight is 0.4, and mission performance weight is 0.3.

In Fig. 13 the best architecture generated when the security and mission performance weights are each set to 0.3 and the cost weight is set to 0.4 is shown, representing a scenario where roughly equal importance is given to all three objectives. This architecture gives an interesting and non-intuitive solution to satisfy the three objectives. It utilizes fewer enclaves than the architecture of Fig. 12 which gives an improved cost component measure of 0.004. It also uses slower cleansing and patching rates to produce an improved mission performance degradation component measure of 0.018. With fewer enclaves and slower cleansing and patching rates, one might expect the security risk component measure to be worse (i.e. higher) but this is not the case. The architecture makes use of fewer total software services and communication pathways that rely on depth to actually *improve* the security risk component measure to 0.130 compared to the architecture generated when the security weighting is 1.0 (which has a security risk component measure of 0.165).

Because of the non-zero cost and mission performance weightings, the system is forced to explore alternative solutions beyond just increasing the total number of enclaves or increasing cleansing/patching rates to improve security. Here, the system uses depth to position some enclaves multiple hops away from the Internet. This makes it more difficult for an attacker as she must wait for new vulnerabilities to appear and utilize new exploits at each hop in order to spread to internal enclaves.

These results show that the system can construct segmentation architectures that satisfy simple scenarios when only one objective is given importance as well as complex scenarios when importance is placed on three different and conflicting objectives. The system utilizes nature-inspired search to explore effective solutions that are sometimes counter-intuitive and, thus, can potentially generate architectures that are better than the ones specified by human experts. These experiments were run on a mid-range business laptop (Dell Inspiron X.X) and took approximately 3-5 minutes to execute per experiment. Because the system is automated, it can potentially be used in high performance computing environments to generate solution architectures for network environments at larger scales (e.g. $10^2 or 10^3$ enclaves).

VI. Conclusion

This paper presents a nature-inspired cyber security decision system that automatically generates NS architectures optimized for three conflicting objectives, security, cost, and mission performance. The system is implemented and demonstrated for a representative network environment under cyber attack. Results show that the system can generate architectures that satisfy both simple scenarios where only one objective is considered and complex scenarios where multiple objectives must be considered.

Future work is focused on testing the decision system in real or emulated network environments to validate the effectiveness generated architectures. Further work will consider alternative algorithms for the optimization component of the system such as genetic algorithms, grammatical evolution, and/or memetic algorithms. Finally, we plan to extend the decision system to solve new cyber security decision problems such as optimal configuration of wireless sensor networks or software defined perimeter defense [29].

References

[1] R. Gezelter, *Computer Security Handbook*. Wiley, 1995, ch. Security on the Internet.

[2] Google, Inc., "Google's Approach to IT Security," Google, Tech. Rep., 2012.

[3] National Security Agency, "Top 10 Information Assurance Mitigation Strategies," Tech. Rep., 2013.

[4] Microsoft, Inc., "Enterprise Security Best Practices," Microsoft, Tech. Rep., 2015. [Online]. Available: https://technet.microsoft.com/en-us/library/dd277328.aspx

[5] SANS Institute, "Critical Security Controls for Effective Cyber Defense Version 6.0," Tech. Rep., 2015.

[6] N. Reichenberg, "Improving Security via Proper Network Segmentation," Security Week, March 2014.

[7] A. E. Schulz *et al.*, "Cyber network mission dependencies," Massachusetts Institute of Technology Lincoln Laboratory, Tech. Rep. TR-1189, 2015.

[8] J. Guion *et al.*, "Dynamic cyber mission mapping," in *Proc. of the Industrial and Systems Engineering Conference*, 2017.

[9] M. Grimaila and A. Badiru, "A hybrid dynamic decision making methodology for defensive information technology contingency measure selection in the presence of cyber threats," *Operational Research*, vol. 13, no. 1, pp. 67–88, 2013.

[10] S. Miller *et al.*, "Modelling cyber-security experts' decision making processes using aggregation operators," *Computers & SecurityVolume 62*, vol. 62, pp. 229–245, 2016.

[11] A. Hakansson and R. Hartung, "An infrastructure for individualised and intelligent decision-making and negotiation in cyber-physical systems," *Procedia Computer Science*, vol. 35, pp. 822–831, 2014.

[12] A. Mukhopadhyay *et al.*, "Cyber-risk decision models: To insure it or not?" *Decision Support Systems*, vol. 56, pp. 11–26, 2013.

[13] A. Fielder *et al.*, "Decision support approaches for cyber security investment," *Decision Support Systems*, vol. 86, pp. 13–23, 2016.

[14] C. Huber *et al.*, "Cyber fighter associate: A decision support system for cyber agility," in *Conference on Information Science and Systems (CISS)*, 2016.

[15] N. Wagner *et al.*, "Towards automated cyber decision support: A case study on network segmentation for security," in *IEEE Symposium on Computational Intelligence for Cyber Security*, December 2016.

[16] J. Brownlee, *Clever Algorithms: Nature-Inspired Programming Recipes: Nature-inspired Programming Recipes*. Lulu.com, 2011.
[17] J. Riordan *et al.*, "A model of network porosity," MIT Lincoln Laboratory, Tech. Rep., 2016.
[18] N. Wagner *et al.*, "Capturing the security effects of network segmentation via a continous-time markov chain model," in *Proc. of the ACM Spring Simulation Multi-Conference*, April 2017.
[19] R. Lippmann *et al.*, "Continuous security metrics for prevalent network threats: Introduction and first four metrics," MIT Lincoln Laboratory, Tech. Rep., 2012.
[20] N. Wagner *et al.*, "Quantifying the mission impact of network-level cyber defensive mitigations," *Journal of Defense Modeling and Simulation: Applications, Methodology, Technology*, vol. 14, no. 3, pp. 201–216, 2017.
[21] NVD. (2016) The national vulnerability database. [Online]. Available: https://nvd.nist.gov/
[22] A. Nappa *et al.*, "The attack of the clones: A study of the impact of shared code on vulnerability patching," in *2015 IEEE Symposium on Security and Privacy*, May 2015, pp. 692–708.
[23] S. Frei *et al.*, "Large-scale vulnerability analysis," in *Proceedings of the 2006 SIGCOMM Workshop on Large-scale Attack Defense*, ser. LSAD '06. New York, NY, USA: ACM, 2006, pp. 131–138.
[24] S. Frei, "Security econometrics the dynamics of (in) security," Ph.D. dissertation, ETH ZURICH, 2009.
[25] S. Frei *et al.*, *Modeling the Security Ecosystem - The Dynamics of (In)Security*. Boston, MA: Springer US, 2010, pp. 79–106.
[26] CVSS. (2015) Common vulnerability scoring system v3.0. [Online]. Available: https://www.first.org/cvss/specification-document
[27] N. Perlroth, "Security Experts Expect 'Shellshock' Software Bug in Bash to Be Significant," New York Times, Sept. 2014.
[28] K. Mandia *et al.*, "APT1: Exposing One of China's Cyber Espionage Units," FireEye, Inc., Tech. Rep., 2013.
[29] B. Bilger *et al.*, "Software Defined Perimeter," Cloud Security Alliance, Tech. Rep., 2013.