Noname manuscript No.
(will be inserted by the editor)

Optimal Security Hardening on Attack Tree Models of Networks: A Cost-Benefit Analysis

Rinku Dewri · Indrajit Ray · Nayot Poolsappasit · Darrell Whitley

the date of receipt and acceptance should be inserted later

Abstract Researchers have previously looked into the problem of determining if a given set of security hardening measures can effectively make a networked system secure. However, system administrators are often faced with a more challenging problem since they have to work within a fixed budget which may be less than the minimum cost of system hardening. An attacker, on the other hand, explores alternative attack scenarios to inflict the maximum damage possible when the security controls are in place, very often rendering the optimality of the controls invalid. In this work, we develop a systematic approach to perform a cost-benefit analysis on the problem of optimal security hardening under such conditions. Using evolutionary paradigms such as multi-objective optimization and competitive co-evolution, we model the attacker-defender interaction as an "arms race", and explore how security controls can be placed in a network to induce a maximum return on investment.

Keywords Security management, Attack trees, Multi-objective optimization, Competitive co-evolution

Rinku Dewri
Department of Computer Science, University of Denver, Denver, CO 80208, USA
E-mail: rdewri@cs.du.edu

Nayot Poolsappasit
Department of Computer Science, Missouri University of Science & Technology, Rolla, MO 65409, USA
E-mail: nayot@mst.edu

Indrajit Ray · Darrell Whitley
Department of Computer Science, Colorado State University, Fort Collins, CO 80523, USA
E-mail: {indrajit,whitley}@cs.colostate.edu

1 Introduction

Network-based computer systems form an integral part of any information technology infrastructure today. The different levels of connectivity between these systems directly facilitate the circulation of information within an organization, thereby reducing invaluable wait time and increasing the overall throughput. As an organization's operational capacity becomes more and more dependent on networked computing systems, the need to maintain accessibility to the resources associated with such systems has become a necessity. Any weakness or vulnerability that could result in the breakdown of the network has direct consequence on the amount of yield manageable by the organization. This in turn requires the organization to not only consider the advantages of utilizing a networked system, but also consider the costs associated with managing the system.

Researchers have proposed building security models for networked systems using paradigms like *attack graphs* [1–5] and *attack trees* [6–9], and then finding attack paths in these models to determine scenarios that could lead to damage. However, determining possible attack paths, although useful, does not help the system administrators much. They are more interested in determining the best possible way of defending their network in terms of an enumerated set of hardening options [10]. Moreover, the system administrator has to work within a given set of budget constraints which may preclude her from implementing all possible hardening measures or even measures that cover all the weak spots. Thus, the system administrator needs to find a trade-off between the cost of implementing a subset of security hardening measures and the damage that can potentially happen to the system if certain weak spots are left unpatched. We performed a Pareto analysis of

this multi-objective problem in one of our earlier works [11]. The method is driven by an attack tree model that captures the cause-consequence relationships between different network states. These relationships are then used to determine a set of security controls that exhibit non-dominance characteristics with respect to the control cost and residual risk.

While such trade-off analysis provide valuable insights into the problem of optimal security hardening, the approach provides only a static perspective to the problem. The assumption here is that the end goal is to identify a set of security controls that can prevent a particular security breach from occurring. An attacker, meanwhile, continues to explore alternative attack scenarios to inflict maximum damage possible to a system, despite the security controls that are in place. Many a times, the attacker's goal may be just to cause some damage and not necessarily cause the specific security breach that the defender is trying to protect against. The attacker may be aided by several factors in this quest. Defenses may have unknown vulnerabilities that can be exploited as a system evolves with time. Misconfiguration of defenses can render them susceptible to attacks. If the attacker has insider knowledge about system configuration, weak spots and defenses (or lack thereof), such knowledge can be used to increase the probability of a defense failing. Such a situation may not be acceptable to the higher ups in the organization. Their perspective may be to not only prevent a specific security breach but also accept minimal collateral damage. This may require a continual updating of the defense strategy based on attacker activities. Thus, one important objective of security hardening is to make life as difficult for the attacker as possible by adjusting security controls. It seems worth investigating if such an arms race between the attacker and the defender will be perpetual or there exists a state involving security controls in which the defender is guaranteed that unexpected damages will never be inflicted no matter how the attacker changes his strategies.

In this paper, we begin with the formal definition of attack trees first suggested in our earlier work [11]. Using a model that quantifies the potential damage in a system and the security control cost incurred to implement a set of security hardening measures, we show how a cost-benefit analysis can be performed on an attack tree to aid the decision maker. As the primary contribution of this work, we extend the trade-off analysis to explore the optimal security hardening problem keeping in view the attacker's perspective, namely, defenses can be broken. Our goal is to identify how security controls can be decided to maximize the return on investment for a defender, under the scenario that an attacker is actively engaged in maximizing its return on attacks. We explore the optimal security control placement problem as a dynamic engagement between the defender and the attacker, and model the problem as an "*arms race*". Solutions to the optimization problem are obtained using the *competitive co-evolution* paradigm and show how the constant engagement between a defender and an attacker drives the solution towards a state of equilibrium.

Using the outcomes of this analysis, we highlight the inadequacy of current research in addressing the security hardening problem. We argue that research in this frontier has mostly adopted a perspective of optimality that becomes questionable under the light of changing network and attacker dynamics. Hence, towards the end, we present a few insights on what alternative perspective is required.

The rest of the paper is organized as follows. Section 2 explores some of the major works in optimal security hardening. Section 3 gives some background information on multi-objective optimization and competitive co-evolution. In section 4 we describe a simple network that we use to illustrate our problem formulation and solution. The attack tree model is presented in section 5. In section 6 we discuss a motivating example to show how the attacker's perspective may bring about changes in the choice of a security control. In section 7 we define the cost models used in this study. This is followed by the formalization of the optimization problems in section 8. Specifics on the solution methods are presented in section 9. Empirical results and discussions are presented in section 10. Finally, we conclude in section 11 with references to future work.

2 Related Work

Network vulnerability management has been previously addressed in a variety of ways. Noel et al. use *exploit dependency graphs* [10] to compute minimum cost-hardening measures. Given a set of initial conditions in the graph, they compute boolean assignments to these conditions, enforced by some hardening measure, so as to minimize the total cost of those measures. As pointed out in their work, these initial conditions are the only type of network security conditions under our strict control. Hardening measures applied to internal nodes can potentially be bypassed by an attacker by adopting a different attack path. Jha et al. [2] on the other hand do not consider any cost for the hardening measures. Rather, their approach involves finding the minimal set of atomic attacks critical for reaching the goal and then finding the minimal set of security measures that cover this set of atomic attacks.

Such analysis is meant for providing solutions that guarantee complete network safety. However, the hardening measures provided may still not be feasible within the financial or other business constraints of an organization. Under such circumstances, a decision maker must perform a cost-benefit analysis to understand the trade-off between hardening costs and network safety. Furthermore, a minimum cost hardening measure set only means that the root goal is safe, and some residual damage may still remain in the network. Owing to these real-world concerns, network vulnerability management should not always be considered as a single-objective optimization problem.

A multi-objective formulation of the problem is presented by Gupta et al. [12]. They consider a generic set of security policies capable of covering one or more generic vulnerabilities. A security policy can also introduce possible vulnerabilities, thereby resulting in some residual vulnerabilities even after the application of security policies. The multi-objective problem considered is the minimization of the cost of implementing the security policies, as well as the residual vulnerabilities introduced by the policies themselves. However, the authors finally scalarize the two objectives into a single objective using relative weights.

Bistarelli et al. propose *defense trees* as an extension to attack trees to analyze the economic profitability of security measures and their deterrent effects on attackers. A game theoretic perspective of the problem is introduced in an attempt to discover *Nash equilibrium* points between the security provider and the attacker [13]. Early indications of using game theory for network security is provided by Syverson [14]. The work considers the example of a network divided into "good" and "evil" nodes and reasons how game playing can be used to achieve secure computing. Lye and Wing model the interactions between an attacker and an administrator as a two-player stochastic game and use non-linear programming techniques to compute the Nash equilibria or best-response strategies for the players [15]. They propose that attacker and administrator actions probabilistically change the state of a network, resulting in gains and losses for the two players involved. Sallhammar et al. propose the use of stochastic game theory to compute probabilities to attacker actions [16,17]. They also share the view that attacks can be modeled as transitions between system states, and show how the attacker's behavior is influenced by parameters of the game on-going with the defender. Liu and Wang propose a systematic incentive-based framework to model attacker intent, objectives and strategies (AIOS) [18]. Their motivation is aimed towards separating attacker actions and attack effects since the same attack may be the source of different intentions on part of the attacker. The discussion in their work brings out an important characteristic of game theoretic modelling – termed the *dual property* – the best attack (defense) strategy is dependent on the defense (attack) strategies taken. Buldas et al. propose a risk-analysis method based on attack trees to estimate the cost and risk probability of attacks [19]. Their argument is based on the fact that attacks where the cost surpasses the benefit are unlikely in a system and hence such factors should be considered while making a rational decision on security measures. Zhang et al. develop a partially observable Markov decision process to measure the attacker's and defender's behavior in terms of intent and cost factors [20]. Their model aims at revealing the significant aspects of a system that are more likely to be exploited by an attacker and thus aid a defender in detecting on-going attacks based on atomic actions of the attacker. Jiang et al. propose an optimal active defense strategy decision (OADSD) algorithm to compute defense strategies with minimum cost from an iterative attacker-defender game [21].

An implicit assumption in these works is the existence of a payoff matrix that can be used by a software tool to deduce the points of equilibrium. However, as we explain later, the payoff matrix can be too large to be computed for a given problem. Our solution methodology differs here in the adaptation of *payoff functions* defined on attack models, which can then be used by an evolutionary algorithm to find the equilibrium points. Further, the solution methodology adopted by us implicitly models the game undergoing between the attacker and the defender, revealing not only the equilibrium solutions (if any) but also the evolutionary path traversed by the game while reaching the solution. This information provides a decision maker the added knowledge to understand the dynamics of the attacker-defender interactions.

3 Background on Solution Methods

3.1 Multi-objective Optimization

Multi-objective optimization differs from single-objective ones in the cardinality of the optimal set of solutions. Single-objective optimization techniques are aimed towards finding the global optima. There is no such concept of a single optimum solution in case of multi-objective optimization. This is due to the fact that a solution that optimizes one of the objectives may not have the desired effect on the others. As a result, it is not always possible to determine an optimum that corresponds in the same way to all the objectives under con-

sideration. Decision making under such situations thus requires some domain expertise to choose from multiple trade-off solutions depending on the feasibility of implementation.

Due to the conflicting nature of the objective functions, a simple objective value comparison cannot be performed to compare two feasible solutions of a multi-objective problem. Most multi-objective algorithms thus use the concept of dominance.

Definition 1 DOMINANCE AND PARETO-OPTIMAL SET

In a minimization problem with M objective functions $f_1, \ldots, f_M$, a feasible solution vector $\mathbf{x}$ is said to dominate another feasible solution vector $\mathbf{y}$ if

1. $\forall i \in \{1, 2, \ldots, M\} \quad f_i(\mathbf{x}) \leq f_i(\mathbf{y})$ *and*
2. $\exists j \in \{1, 2, \ldots, M\} \quad f_j(\mathbf{x}) < f_j(\mathbf{y})$

$\mathbf{y}$ is then said be dominated by $\mathbf{x}$. If the two conditions do not hold, $\mathbf{x}$ and $\mathbf{y}$ are said to be non-dominated w.r.t. each other. Further, the set of all non-dominated solutions obtained over the entire feasible region constitutes the Pareto-optimal set.

In other words, a Pareto-optimal solution is as good as other solutions in the Pareto-optimal set, and not worse than other feasible solutions outside the set. The surface generated by these solutions in the objective space is called the *Pareto-front* or *Pareto-surface*.

The classical way to solve a multi-objective optimization problem is to follow the preference-based approach. A relative weight vector for the objectives can help reduce the problem to a single-objective instance, or impose orderings over the preference given to different objectives. However, such methods fail to provide a global picture of the choices available to the decision maker. In fact, the decision of preference has to be made before starting the optimization process. Relatively newer methods have been proposed to make the decision process more interactive.

Evolutionary algorithms for multi-objective optimization (EMO) have been extensively studied and applied to a wide spectrum of real-world problems. One of the major advantages of using evolutionary algorithms for optimization is their ability to scan through the global search space simultaneously, instead of restricting to localized regions of gradient shifts. An EMO works with a population of trial solutions, trying to converge on to the Pareto-optimal set by filtering out the infeasible or dominated ones. Having multiple solutions from a single run of an EMO is not only an efficient approach, but also helps a decision maker obtain an intuitive understanding of the different trade-off options available at hand. The effectiveness of an EMO is thus characterized by its ability to converge to the true Pareto-front and maintain a good distribution of solutions on the front.

A number of algorithms have been proposed in this context [22,23]. We employ the Non-dominated Sorting Genetic Algorithm (NSGA-II) [24] for the multi-objective optimization in this study. NSGA-II has gained a wide popularity in the multi-objective optimization community, partly because of its efficiency in terms of the convergence and diversity of solutions obtained, and partly due to its extensive application to solve real-world problems.

3.2 Competitive Co-evolution

Competitive co-evolution refers to the concurrent evolution of two distinct species in which the fitness of an individual in one species is based on its competitive abilities against the individuals of the other species. Fitness evaluation with such reciprocal actions are hypothesized to occur in nature. Game theory based models of such interactions are first presented in Axelrod's *Prisoners' Dilemma* [25]. The evolution of species in a competitive habitat usually leads to an *evolutionary stable strategy* [26] which cannot be invaded by the process of natural selection. In other words, the species reverts back to the stable strategy over time.

Competitive co-evolution has been successfully applied to the evolution of strategies for games such an Tic-Tac-Toe and Nim [27]. The range of potential opponent strategies is typically very large in such games, thereby making it difficult to determine an exogenous fitness evaluation function. Other domains such as software reliability testing faces a similar problem. The solution using competitive co-evolution involves using two populations, one representing the software solutions and the other representing the test cases, each taking turns in testing and being tested against the other [28]. A survey of other real world applications is available in [29].

Success of competitive co-evolution is attributable to the emergence of an evolutionary arms race [30]. Consider two populations of defense strategies and attack strategies. To begin with, both populations are likely to have strategies of poor quality. Most of the host strategies will have low payoffs brought forth by one or two good strategies existing in the opponent population. However, since defense strategies are evolving based on their competitive abilities against attack strategies, the success of the defender implies the failure of the attacker. When the attacker finds strategies to improve its payoff by overcoming the failure, it helps the defender identify gaps previously unthought of in its

strategies. The same idea drives the attacker's strategies. New opponent strategies drive hosts towards better counter strategies, improving host performance by forcing it to respond to a wider range of more challenging test cases.

The next question that comes to mind is whether a good host strategy of the current generation can prove its competence against opponent strategies that are lost in the evolution of the opponent population. This is referred to as the *memory property* in co-evolution. To handle such situations, co-evolutionary algorithms employ a "*hall of fame*" [31] sub-population which keeps track of the best opponent solutions found from earlier generations. Success of a competition for a host strategy is measured not only relative to the current opponent strategies but is also dependent on its performance against the opponent's hall of fame. Other similar methods are elaborated in [32,33].

4 A Simple Network Model

We consider the hypothetical network shown in Figure 1 to illustrate our methodology. The setup consists of four hosts. A firewall is installed with a preset policy to ensure that only the *FTP* and *SMTP servers* are allowed to connect to the external network. In addition, FTP and SSH are the only two services an external user can use to communicate with these servers. We assume that an external user wants to compromise the *Data Server* which is located inside the firewall. The firewall has a strong set of policies setup to protect access to the internal hosts. There are six different attack scenarios possible to achieve the ultimate goal from a given set of initial vulnerabilities and network topology as listed in Table 1 and 2.

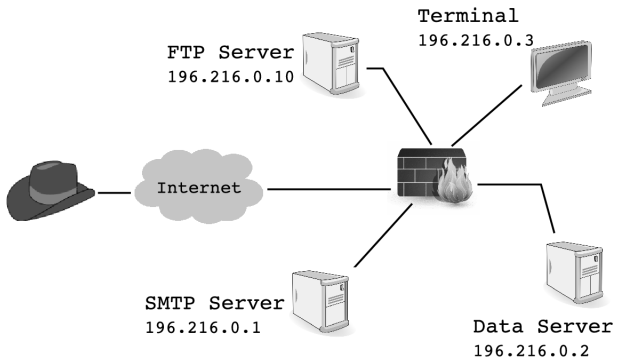


Fig. 1 Example network model.

To compromise the Data Server, an attacker can exploit the FTP and SMTP Servers using the *ftp/.rhost attack*. Both servers are running ftp server versions that are vulnerable to these exploits. In addition, their *rhost directories* are not properly write-protected. As a consequence of the ftp/.rhost exploit, an attacker establishes a trust relation between the host and attacker machines, and introduces an *authentication bypassing vulnerability* in the victim. An attacker can then log in to these servers with user access privilege. From this point the attacker can use the connection to the Data Server to compromise it. The attacker may also compromise the SMTP Server, or choose to compromise the *Terminal* machine in order to delay an attack. The Terminal machine can be compromised via the chain of *LICQ remote to user attack* and the *local buffer overflow attack* on the *"at" daemon*. Finally, the attacker from either the FTP server, SMTP server, or the Terminal machine can use the connectivity to the Data Server to compromise it through the chain of *LICQ exploit* and *"suid" local buffer overflow attack*. Such attack scenarios can be succinctly represented using an *attack tree*, discussed in details in the next section.

Table 1 Initial vulnerability per host in example network.

Host	Vulnerability	CVE#
FTP Server	Ftp .rhost attack	1999-0547
196.216.0.10	Ftp Buffer overflow	2001-0755
	Ssh Buffer overflow	2006-2421
SMTP Server	Ftp .rhost attack	1999-0547
196.216.0.1		
Terminal	LICQ remote-2-user	2001-0439
196.216.0.3	"at" heap corruption	2002-0004
Data Server	LICQ remote-2-user	2001-0439
196.216.0.2	suid Buffer overflow	2001-1180

Table 2 Connectivity in example network.

Host	Host	Port
..*.*	196.216.0.1	21,25
..*.*	196.216.0.10	21,22
196.216.0.1	196.216.0.2	ANY
196.216.0.1	196.216.0.3	ANY
196.216.0.3	196.216.0.2	ANY
196.216.0.10	196.216.0.2	ANY

5 Attack Tree Model

Materializing a threat usually requires the combination of multiple attacks using different vulnerabilities. Representing different scenarios under which an asset can be damaged thus becomes important for preventive analysis. Such representations not only provide a picture of the possible ways to compromise a system, but can also help determine a minimal set of preventive actions. Given the normal operational state of a network, including the vulnerabilities present, an attack can possibly open up avenues to launch another attack,

thereby taking the attacker a step closer to its goal. A certain state of the network in terms of access privileges or machine connectivity can be a prerequisite to be able to exploit a vulnerability. Once the vulnerability is exploited, the state of the network can change enabling the attacker to launch the next attack in the sequence. Such a pre-thought sequence of attacks gives rise to an *attack scenario*.

It is worth noting that such a notion of a progressive attack induces a transitive relationship between the vulnerabilities present in the network and can be exploited while deciding on the security measures. Attack graph [1,2,4,10] and attack tree [8,9] representations have been proposed in network vulnerability management to demonstrate such cause-consequence relationships. The nodes in these data structures usually represent a certain network state of interest to an attacker, with edges connecting them to indicate the cause-consequence relationship. Although different attack scenarios are easily perceived in attack graphs, they can potentially suffer from a state space explosion problem. Ammann et al. [1] identified this problem and propose an alternative formulation with the assumption of *monotonicity*. The monotonicity property states that the consequence of an attack is always preserved once achieved. Such an assumption can greatly reduce the number of nodes in the attack graph, although at the expense of further analysis required to determine the viable attack scenarios. An *exploit-dependency graph* can be extracted from their representation to indicate the various conjunctive and disjunctive relationships between different nodes. For the purpose of this study, we adopt the attack tree representation since it presents a much clearer picture of the different hierarchies present between attacker sub-goals. An attack tree uses explicit conjunctive and disjunctive branch decomposition to reduce the visualization complexity of a sequence of operations.

Different properties of the network effectuate different ways for an attacker to compromise a system. We first define an *attribute-template* that lets us generically categorize these network properties for further analysis.

Definition 2 Attribute-Template

An attribute-template is a generic property of the hardware or software configuration of a network which includes, but not limited to, the following:

- *system vulnerabilities (which are often reported in vulnerability databases such as BugTraq, CERT/CC, or NetCat).*
- *network configuration such as open port, unsafe firewall configuration, etc.*
- *system configuration such as data accessibility, unsafe default configuration, or read-write permission in file structures.*
- *access privilege such as user account, guest account, or root account.*
- *connectivity.*

An attribute-template lets us categorize most of the atomic properties of the network that might be of some use to an attacker. For example, *"running SSH1 v1.2.23 on FTP Server"* can be considered as an instance of the system vulnerabilities template. Similarly, *"user access on Terminal"* is an instance of the access privilege template. Such templates also let us specify the properties in propositional logic. We define an *attribute* with such a concept in mind.

Definition 3 Attribute

An attribute is a propositional instance of an attribute-template. It can take either a true *or* false *value.*

The success or failure of an attacker reaching its goal depends mostly on what truth values the attributes in a network take. Its also lays the foundations for a security manager to analyze the effects of falsifying some of the attributes using some security policies. We formally define an attack tree model based on such attributes. Since we consider an attribute as an atomic property of a network, taking either a *true* or *false* value, most of the definitions are written in propositional logic involving these attributes.

Definition 4 Attack

Let S be a set of attributes. We define Att to be a mapping $Att : S \times S \rightarrow \{true, false\}$ and $Att(s_c, s_p) =$ truth value of s_p.

$a = Att(s_c, s_p)$ is an attack if $s_c \neq s_p \wedge a \equiv s_c \leftrightarrow s_p$. s_c and s_p are then respectively called a precondition and postcondition of the attack. The set of all preconditions and postconditions of a are denoted by pre(a) *and* post(a) *respectively.*

$Att(s_c, s_p)$ is a ϕ–attack if $\exists$non-empty $S' \subset S | [s_c \neq s_p \wedge Att(s_c, s_p) \equiv (\bigwedge_i s_i \wedge s_c) \leftrightarrow s_p]$ where $s_i \in S'$.

An attack relates the truth values of two different attributes so as to embed a cause-consequence relationship between the two. For example, for the attributes $s_c =$*"vulnerable to sshd BOF on machine A"* and $s_p =$*"root access privilege on machine A"*, $Att(s_c, s_p)$ is an attack – the sshd buffer overflow attack. We would like to clarify here that the bi-conditional logical connective "$\leftrightarrow$" between s_c and s_p does not imply that s_p can be set to *true* only by using $Att(s_c, s_p)$; rather it means that given the sshd BOF attack, the only

way to make s_p *true* is by having s_c *true*. In fact, *Att("vulnerable to local BOF on setuid daemon on machine A",*s_p*)* is also a potential attack. The ϕ–attack is included to account for attributes whose truth values do not have any direct relationship. However, an indirect relationship can be established collectively. For example, the attributes s_{c_1} = "*running SSH1 v1.2.25 on machine A*" and s_{c_2} = "*connectivity(machine B, machine A)*" cannot individually influence the truth value of s_c, but can collectively make s_c *true*, given they are individually *true*. In such a case, $Att(s_{c_1}, s_c)$ and $Att(s_{c_2}, s_c)$ are ϕ–attacks.

Definition 5 Attack Tree

Let A be the set of attacks, including the ϕ–attacks. An attack tree is a tuple $AT = (s_{root}, S, \tau, \varepsilon)$, where

1. s_{root} is an attribute which the attacker wants to become true.

2. $S = N_{internal} \cup N_{external} \cup \{s_{root}\}$ is a multiset of attributes. $N_{external}$ denotes the multiset of attributes s_i for which $\nexists a \in A | s_i \in post(a)$. $N_{internal}$ denotes the multiset of attributes s_j for which $\exists a_1, a_2 \in A | [s_j \in pre(a_1) \wedge s_j \in post(a_2)]$.

3. $\tau \subseteq S \times S$. An ordered pair $(s_{pre}, s_{post}) \in \tau$ if $\exists a \in A | [s_{pre} \in pre(a) \wedge s_{post} \in post(a)]$. Further, if $s_i \in S$ and has multiplicity n, then $\exists s_1, s_2, \ldots, s_n \in S | (s_i, s_1), (s_i, s_2), \ldots, (s_i, s_n) \in \tau$, and

4. ε is a set of decomposition tuples of the form $\langle s_j, d_j \rangle$ defined for all $s_j \in N_{internal} \cup \{s_{root}\}$ and $d_j \in \{AND, OR\}$. d_j is AND when $\bigwedge_i [s_i \wedge (s_i, s_j) \in \tau] \leftrightarrow s_j$ is true*, and OR when $\bigvee_i [s_i \wedge (s_i, s_j) \in \tau] \leftrightarrow s_j$ is* true*.*

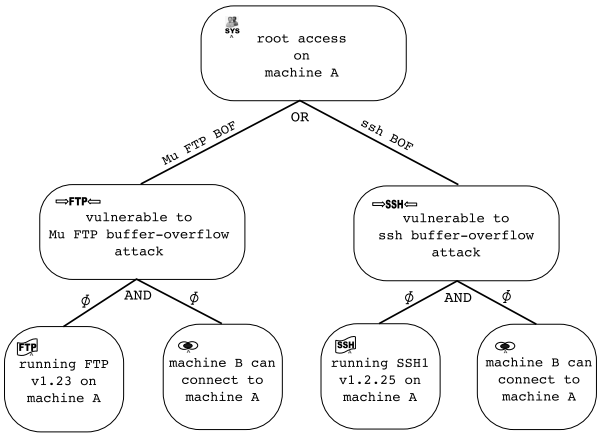


Fig. 2 Example attack tree.

Figure 2 shows an example attack tree, with the attribute "*root access on machine A*" as s_{root}. The multiset S forms the nodes of the tree. The multiset $N_{external}$ specify the leaf nodes of the tree. These nodes reflect the initial vulnerabilities present in a network and are prone to exploits. Since an attribute can be a precondition for more than one attack, it might have to be duplicated, hence forming a multiset. The attribute "*machine B can connect to machine A*" in the example is one such attribute. The set of ordered pairs τ reflects the edges in the tree. The existence of an edge between two nodes implies that there is a direct or indirect relationship between their truth values, signified by the decomposition at each node. The *AND* decomposition at a node requires all child nodes to have a truth value of *true* for it to be *true.* The *OR* decomposition at a node requires only one child node to have a truth value of *true* for it to be *true.* Using these decompositions, the truth value of an attribute $s_j \in N_{internal} \cup \{s_{root}\}$ can be evaluated after assigning a set of truth values to the attributes $s_i \in N_{external}$. Figure 3 shows the attack tree for our example network model. It depicts a clear picture of the different attack scenarios possible, as outlined in the previous section.

Attack trees for large networks can get complex. We have the search space bound to the number of attributes that specify what vulnerabilities are present in which machines. The size of the attribute instances can be as large as $A \times M$, where A is the number of attributes and M is the number of machines in the system. However, note that the generation of the attack tree is a one-time cost and is not done in real time. Our in-house tool takes as input an initial vulnerability table, generated by a vulnerability scanner, and the network topology. Using a sequence of SQL queries on a vulnerability exposure database, the tool creates consequence attributes for the tree until no further implications can be derived. Commercial tools (e.g. CAULDRON: http://proinfomd.com) are also available that explore the topological and security dependencies in a large-scale real-world network.

A defender installs defenses on the network (makes some or all leaf nodes *false*) so as to prevent the root node from becoming *true*. The defender's choice of defenses may be determined by factors such as the installation cost and the potential damage residual after making the choice. From an attacker's perspective, the attack tree is a model showing the different ways it can compromise the root node. However, we do not restrict our focus to the root node alone. An attacker's strategy might as well be directed towards inflicting the most damage in the presence of defenses, rather than just compromising the root node. The choice of such a strategy is also influenced by the difficulty that the attacker has to overcome in order to bypass any installed defenses. In the next section we give a formal outline of a defense and an attack strategy.

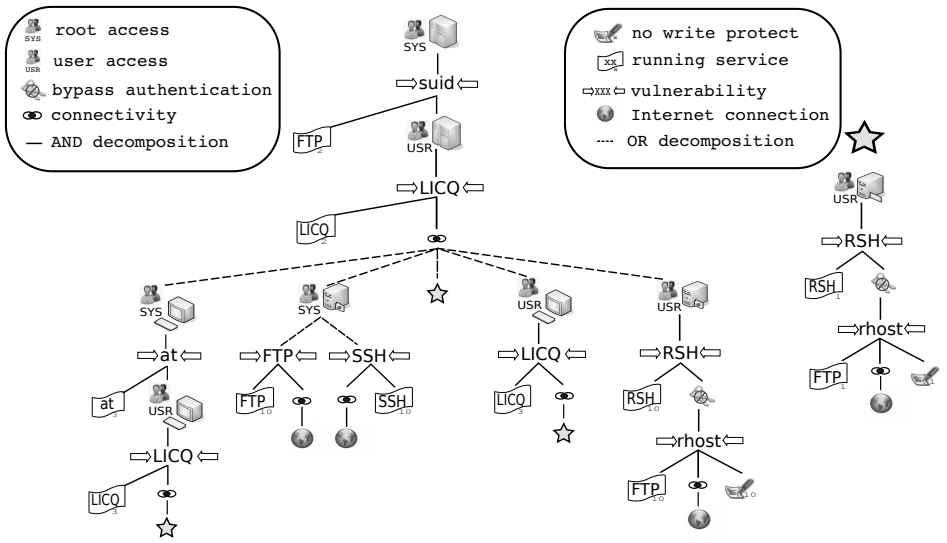


Fig. 3 Attack tree of example network model.

6 Defense and Attack Strategy

Incorporating the attacker's perspective in the optimal security hardening problem is not easy. We consider a hypothetical example to illustrate how a defender's decision to employ a particular strategy is influenced when the attacker's gains are kept in consideration. Consider the payoff matrix shown in Figure 4. The example assumes that the defender has two possible defense strategies d_1 and d_2, and the attacker has two different attack strategies a_1 and a_2 to try out. The objective of the defender is to decide on one defense strategy to adopt. The first value in a cell (i, j) is a measure of some payoff that the defender derives by adopting strategy d_i under the situation when the attacker uses strategy a_j. Given that the defender is only interested in its payoff value, it uses an average case analysis and finds that strategy d_1 can maintain a higher average payoff than d_2 – 7.5 compared to 5.5. The defender will arrive at the same strategy even with a best case analysis. The defender therefore installs the defense d_1. However, the decision on d_1 can reveal itself to be flawed when the attacker's payoffs are introduced.

The second value in a cell (i, j) is a measure of the payoff that the attacker derives by adopting attack a_j when defense d_i is in place. With d_1 in place, the attacker sees that its payoff is more (6 compared to 2) by adopting strategy a_1. Hence, it will always employ a_1, in which case the defender will always derive a payoff of 5. This value is not only less than the average payoff of d_1, but is also less than the average payoff of d_2. The value is not even better than the individual payoffs possible with d_2, i.e. 6 when a_1 occurs and 5 when a_2 occurs. Further, if we consider the situation where the attacker does not know which defense is in place and wants to choose a strategy using an average or best case analysis, strategy a_1 is the favorable choice. This is because strategy a_1 always provides a higher payoff than a_2 no matter which defense is in place. In the light of this analysis, the defender should thus be choosing strategy d_2. Since the attacker's choice is inclined towards using a_1, the defender now derives a payoff of 6, compared to 5 when choosing d_1.

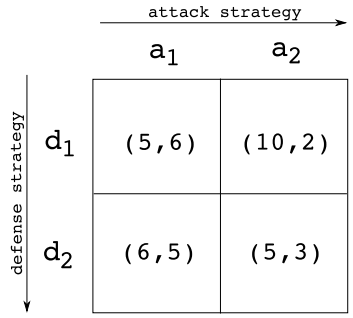


Fig. 4 A hypothetical payoff matrix showing defender and attacker payoffs.

Another interesting facet of d_2 is the equilibrium it maintains with a_1. Let us assume that the defender does a best case analysis, as in the case when the attacker's payoffs are not known, and chooses d_1. The attacker then employs a_1 to maximize its payoff. The defender notices that its payoff is not optimal when a_1 occurred and so switches to d_2 to increase it. Hereafter, although the defense strategy has changed, the attacker's best strategy is to stick to a_1. In other words, the defender and the attacker enters a state of equilibrium where none benefits any further from changes in strategy. Hence, even though d_1 appears to be the op-

timal strategy at first glance, over time the defender changes policies to finally settle down with d_2 – the *equilibrium strategy.*

One may ask what is the equilibrium solution's relation to the notion of optimal security hardening. Consider the scenario where a defender installs defenses based on some optimality criteria on a system. Over time an attacker finds the best possible way to exploit the system under the defensive configuration. The defender notices the attacker's exploitation mechanism and modifies its policies keeping in consideration the optimality criteria. The attacker adapts to the changes and the process continues. When the defense policies corresponding to the equilibrium condition are instantiated and the attacker adapts to it, the defender is already running the optimal set of policies possible for the attacker's adaption and does not need to change it. Thus, in the long run, the notion of optimal security hardening converges towards security in equilibrium with attacks.

Performing an analysis of the nature shown in the simple example is relatively more difficult on a larger scale. First, the payoff matrix can be very large in a real scenario. For d defense controls and a attack nodes, this matrix can be as large as $2^d \times 2^a$. Filling the matrix can thus involve an immense number of evaluations. Second, even if the matrix can be computed, performing the analysis to decide on the best strategy can be impractical. Note that a best (or equilibrium) defense strategy as depicted in the example may not exist at all. For example, if the values at cell $(2, 1)$ are replaced by $(4, 10)$, then the best strategy for the attacker varies depending on the defense. Nonetheless, we can argue that d_1 is a better defense strategy in this case since the payoff is better than from d_2 – 5 with a_1 as the strategy of choice for the attacker compared to 4 with a_2 as the attacker's choice.

Ideally it would be sufficient to decide on a defense strategy by comparing it against others under the light of attack strategies resulting in higher payoffs for the attacker. One may visualize the attack strategies as test cases to measure the competence of a defense strategy. Better test cases are those which are more difficult to solve, or in other words, result in inferior performance of the defense strategy. Similarly, an attack strategy should only be analyzed against defense strategies that result in higher payoffs for the defender. The presence of such cyclic dependencies in the evaluation process makes the analysis hard to conduct. Moreover, the optimal defense strategy will most likely have to be changed over time to maintain maximum payoff depending on what strategy is chosen by the attacker. Hence we believe it is worth investigating if an equilibrium strategy exists for the security hardening problem.

First we define the notion of a security control (or defense) in the context of the attack tree definition.

Definition 6 Security Control (Defense)

Given an attack tree $(s_{root}, S, \tau, \varepsilon)$*, the mapping* $SC : N_{external} \rightarrow \{true, false\}$ *is a security control if* $\exists s_i \in N_{external} | SC(s_i) = false$*.*

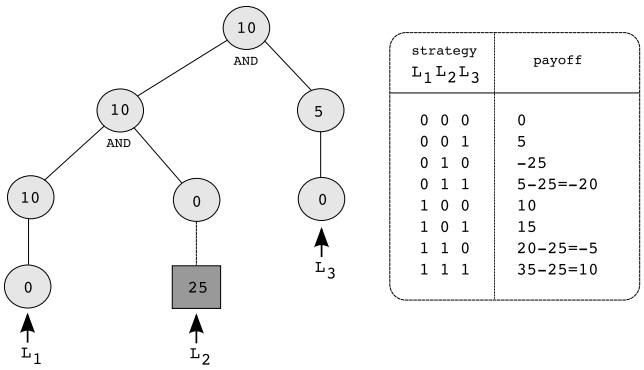
In other words, a security control is a preventive measure to falsify one or more attributes in the attack tree, so as to stop an attacker from reaching its goal. Further, in the presence of multiple security controls SC_k, the truth value of an attribute $s_i \in N_{external}$ is taken as $\bigwedge_k SC_k(s_i)$. Given a security control SC, the set of all $s_i \in N_{external} | SC(s_i) = false$ is called the *coverage* of SC. Hence, for a given set of security controls we can define the *coverage matrix* specifying the coverage of each control. For a given set of d security controls, we use the boolean vector $\mathbf{S} = (S_1, S_2, \ldots, S_d)$ to indicate if a security control is chosen by a security manager.

In order to defend against the attacks possible, the defender can choose to implement a variety of safeguard technologies. Each choice of action can have a different cost involved. Besides, some measures can have multiple coverages but with higher costs. The defender has to make a decision and choose to implement a subset of these policies in order to maximize the resource utilization.

Definition 7 Defense Strategy

For a given set of d defenses, the defense strategy $\mathbf{S_D} = (S_{D_1}, S_{D_2}, \ldots, S_{D_d})$ *is a boolean vector indicating which defenses are chosen by the defender.* $S_{D_i} = 1$ *if defense* D_i *is chosen, zero otherwise.*

The choice of this vector indirectly specifies which leaf nodes in the attack tree would be *false* to begin with. An attacker typically exploits leaf nodes that are not covered by any defense in order to progressively climb up the tree, inflicting some amount of damage to the network at every step. However, it is not always correct to assume that an attacker can no longer exploit some parts of the attack tree because of the installed defenses. With the appropriate tools and knowledge, an attacker may have the potential to bypass a defense as well. In other words, leaf nodes which were made *false* by a defense can be reverted back to being *true.* We thus assume an attacker with the requisite knowledge to breach a defense. However, in order to do so the attacker will have to incur some cost, often related to the number of defenses in place and the difficulty to



strategy $L_1 L_2 L_3$	payoff
0 0 0	0
0 0 1	5
0 1 0	-25
0 1 1	5-25=-20
1 0 0	10
1 0 1	15
1 1 0	20-25=-5
1 1 1	35-25=10

Fig. 5 Payoff for different attack strategies in hypothetical attack tree. Circles denote nodes of the attack tree and the rectangle denote a defense. Value within a circle signifies a payoff value that the attacker receives if it succeeds in reaching the node. Value within the rectangle is the cost that the attacker incurs to bypass the defense.

breach them. If an attacker's gains are less than the cost incurred, then its effort to breach the defense is not worth the time and value. This primarily motivates the defender to still install defenses despite there being a chance of breach.

Given that the attacker can bypass an installed defense (after incurring a cost), it can start its exploits from any leaf node on the attack tree. The attacker's progress towards the root is then decided by the leaf nodes it choose. Note that choosing all leaf nodes that can collectively make an intermediate node *true* need not always be the best approach for the attacker. For instance, given that defenses will be in place at different levels of the tree and the attacker will have to incur a cost to bypass them, it is possible that the attacker derives more payoff by inflicting damages at different parts of the attack tree rather than continuing along a single scenario all the way up to the root. An example of this situation is depicted in Figure 5. With the given values and the defense in place, the strategy 101 generates a higher payoff than trying to reach the root node with strategy 111. This happens because the cost to breach the installed defense nullifies any gains derived from breaching it.

An attack strategy is thus defined as follows.

Definition 8 Attack Strategy

Let n denote the number of unique leaf nodes in an attack tree. An attack strategy $\mathbf{S_A} = (S_{A_1}, S_{A_2}, \ldots, S_{A_n})$ is a boolean vector indicating which leaf nodes in the tree are chosen by the attacker for exploit. $S_{A_i} = 1$ if node $A_i \in N_{external}$ is chosen, zero otherwise.

Thus, an attack strategy specifies the path(s) that the attacker pursues to an intermediate or the top level of the attack tree. The success of the strategy depends on the defense strategy adopted by the defender, as well as the number of levels it can move up on the tree. Another way to visualize an attack strategy is the set of leaf nodes that the attacker assumes to be *true,* or will make *true* by breaching the defenses protecting them.

7 Cost Model

Security planing begins with risk assessment which determines threats, loss expectancy, potential safeguards and installation costs. Many researchers have studied risk assessment schemes, including the National Institute of Standards and Technology (NIST) [34]. For simplicity, the security manager can choose to evaluate the risks by considering a relative magnitude of loss and hardening costs [34–36]. However, relative-cost approaches do not provide sufficient information to prioritize security measures especially when the organization faces resource constraints. We adapt Butler's multi-attribute risk assessment framework [37,38] to develop quantitative risk assessments for our security optimization. Butler's framework enables an aggregated representation of the various factors dominating the business model of an organization.

7.1 Evaluating Potential Damage

The potential damage P_j represents a unit-less damage value that an organization may have to incur in the event that an attribute s_j becomes *true.* Based on Butler's framework, we propose four steps to calculate the potential damage for an attribute s_j.

Step 1 Identify potential consequences of having a *true* value for the attribute. In our case, we have identified five outcomes – lost revenue (monetary), non-productive downtime (time), damage recovery (monetary), public embarrassment (severity) and law penalty (severity) – denoted by x_{1j}, x_{2j}, x_{3j}, x_{4j} and x_{5j}.

Step 2 Estimate the expected number of attack occurrence, $Freq_j$, resulting in the consequences. A security manager can estimate the expected number of attack from the organization-based historical data or public historical data[1].

Step 3 Assess a single value function, $V_{ij}(x_{ij})$, for each possible consequence. The purpose of this function is to normalize different unit measures so that the

[1] Also known as an incident report published annually in many sites such as CERT/CC or SANS.ORG.

values can be summed together under a single standard scale.

$$V_{ij}(x_{ij}) = \frac{x_{ij}}{\underset{j}{Max}\ x_{ij}} \times 100 \qquad , 1 \leq i \leq 5 \tag{1}$$

Step 4 Assign a preference weight factor, W_i, to each possible consequence. A security manager can rank each outcome on a scale of 1 to 100. The outcome with the most concern would receive 100 points. The manager ranks the other attributes relative to the first. Finally, the ranks are normalized and set as W_i.

The potential damage for the attribute can then be calculated from the following equation.

$$P_j = Freq_j \times \sum_{i=1}^{5} W_i V_{ij}(x_{ij}) \tag{2}$$

When using an attack tree, a better quantitative representation of the cost is obtained by considering the residual damage once a set of security controls are implemented. Hence, we augment each node in the attack tree with a value signifying the amount of potential damage residing in the subtree rooted at the node and the node itself.

Definition 9 Augmented-Attack Tree

Let $AT = (s_{root}, S, \tau, \varepsilon)$ be an attack tree. An augmented-attack tree $AT_{aug} = AT|\langle I, V\rangle$ is obtained by associating a tuple $\langle I_i, V_i\rangle$ to each $s_i \in S$, where

1. I_i is an indicator variable for the attribute s_i, where

$$I_i = \begin{cases} 0 & , if\ s_i\ is\ false \\ 1 & , if\ s_i\ is\ true \end{cases}$$

2. V_i is a value associated with the attribute s_i.

In this work, all attributes $s_i \in N_{external}$ are given a zero value. The value associated with $s_j \in N_{internal} \cup \{s_{root}\}$ is then computed recursively as follows.

$$V_j = \begin{cases} \sum\limits_{k|(s_k,s_j)\in\tau} V_k & + I_j P_j \quad , if\ d_j\ is\ AND \\ \max\limits_{k|(s_k,s_j)\in\tau} V_k & + I_j P_j \quad , if\ d_j\ is\ OR \end{cases} \tag{3}$$

Ideally, P_j is same for all identical attributes in the multiset. We took a *"panic approach"* in calculating the value at each node, meaning that given multiple subtrees are rooted at an attribute with an OR decomposition, we choose the maximum value. We do so because an attacker's capabilities and preferences cannot be known in advance. A similar worst case modeling is also adopted later when computing attacker and defender payoffs. The residual damage of the augmented tree is defined as follows.

Definition 10 Residual Damage

Given an augmented-attack tree $(s_{root}, S, \tau, \varepsilon)|\langle I, V\rangle$ and a defense strategy $\mathbf{S_D}$, the residual damage is defined as the value associated with s_{root}, i.e.,

$$RD(\mathbf{S_D}) = V_{root}.$$

7.2 Evaluating Security Cost

Similar to the potential damage, the security manager first lists possible security costs for the implementation of a security control, assigns the weight factor on them, and computes the normalized value. The only difference is that there is no expected number of occurrence needed in the evaluation of security cost. In this study, we have identified five different costs of implementing a security control – installation cost (monetary), operation cost (monetary), system downtime (time), incompatibility cost (scale), and training cost (monetary). The overall cost C_j, for the security control SC_j, is then computed in a similar manner as for potential damage, with an expected frequency of 1. The total security cost for a set of security controls implemented is then defined as follows.

Definition 11 Total Security Control Cost

Given a set of d security controls, each having a cost $C_i; 1 \leq i \leq d$, and a defense strategy $\mathbf{S_D}$, the total security control cost is defined as

$$SCC(\mathbf{S_D}) = \sum_{i=1}^{d} (S_{D_i} C_i).$$

7.3 Estimating Attacker and Defender Payoffs

In order to compute the attacker and defender payoffs, the value associated with each node of the attack tree, V_j as given by (3), is modified as follows. All attributes $s_i \in N_{external}$ are given a zero value. The value associated with $s_j \in N_{internal} \cup \{s_{root}\}$ is then computed recursively as

$$V_j = \sum_{k|(s_k,s_j)\epsilon\tau} V_k \quad + I_j P_j. \tag{4}$$

Under this formulation, the value associated with a node signifies the sum of the total potential damage present in the child subtree(s) and the potential damage of the node itself. If no defense is installed, i.e. all leaf nodes are *true*, then V_{root} gives the maximum damage possible on the attack tree. When a defender decides on a defense strategy, it essentially sets the truth values of the covered leaf nodes to *false*. Uncovered leaf nodes

are set to *true*. An attacker reverts any falsified leaf node to *true* if the node is chosen as part of the attack strategy. With this configuration, we can then find out the damage inflicted on the attack tree as a result of an attack strategy.

Definition 12 Damage Inflicted

For a given defense strategy $\mathbf{S_D}$ *and an attack strategy* $\mathbf{S_A}$ *on an augmented-attack tree* AT_{aug}*, the damage inflicted* DI *is given by the value of the root node of the tree, i.e.*

$$DI(\mathbf{S_D}, \mathbf{S_A}) = V_{root}.$$

The payoff for a defender and an attacker is an estimate of the gain they receive by adopting a particular strategy and after incurring the corresponding costs associated with the implementation of the strategy. For a defender, the cost of implementation relates to factors such as operations cost, training cost, system downtime, incompatibility cost and installation cost, and given by Definition 11. For an attacker, the cost of realizing an attack strategy is related to the effort it has to put forward in overcoming any defenses on its way. We model this cost under a simplistic assumption that stronger defenses are likely to have a higher cost of implementation. Under this assumption, we measure the relative difficulty to breach a defense – a value in $[0, 1]$ – and assign the cost to breach it, $BC(\cdot)$, as a fraction (given by the difficulty value) of the cost of implementation of the defense, i.e.

$$BC(D_i) = \frac{C_i}{\underset{i}{Max}\, C_i} \times C_i. \tag{5}$$

Definition 13 Attack Strategy Cost

Given a set of d *defenses, a defense strategy* $\mathbf{S_D}$ *and an attack strategy* $\mathbf{S_A}$ *on an attack tree* AT*, the attack strategy cost* ASC *is defined as*

$$ASC(\mathbf{S_D}, \mathbf{S_A}) = \sum_{i=1}^{d} \sum_{j|D_i(A_j)=false} [BC(D_i) S_{D_i} S_{A_j}].$$

The expression above iterates through the leaf nodes covered by a particular defense. Thereafter, the cost to breach the defense is added to the attack strategy cost if the defense is part of the defense strategy and the leaf node is part of the attack strategy. When a breach occurs, the cost paid by the defender to install it (C_i) is a loss, called the breach loss $BL(\cdot)$ and expressed in a manner similar to the above equation.

$$BL(\mathbf{S_D}, \mathbf{S_A}) = \sum_{i=1}^{d} \sum_{j|D_i(A_j)=false} [C_i S_{D_i} S_{A_j}] \tag{6}$$

We then define the defender and attacker payoffs as follows.

Definition 14 Payoff for Defender and Attacker

For a given defense strategy $\mathbf{S_D}$ *and an attack strategy* $\mathbf{S_A}$ *on an augmented-attack tree* AT_{aug}*, the defender's payoff* POD *is given as,*

$$POD(\mathbf{S_D}, \mathbf{S_A}) = DI(\mathbf{0}, \mathbf{1}) + SCC(\mathbf{S_D}) - DI(\mathbf{S_D}, \mathbf{S_A}) - BL(\mathbf{S_D}, \mathbf{S_A})$$

and the attacker's payoff POA *is given as,*

$$POA(\mathbf{S_D}, \mathbf{S_A}) = DI(\mathbf{S_D}, \mathbf{S_A}) - ASC(\mathbf{S_D}, \mathbf{S_A}).$$

Here, $DI(\mathbf{0}, \mathbf{1})$ signifies the maximum damage possible on the attack tree, which happens when there are no defenses installed and the attacker exploits all leaf nodes. $\mathbf{0}$ represents the all zero vector and $\mathbf{1}$ is the all one vector. Note that both payoff functions employ the same DI value derived from the attack tree. One can argue that the attacker's knowledge on the damages sustained by the defender when compromising a node is rather limited, and thus cannot be the same as that of the defender. Further, the attacker need not have the complete knowledge about the cost of implementing a defense and hence will not know the exact value of ASC. We understand that both are rational arguments. Our justification to them is based on the fact that the POA function need not be an exact estimate of the actual payoff derived by the attacker. The optimization process only needs to compare payoff values to determine the relative effectiveness of two attack strategies, in which case it suffices to have a value proportional to the actual payoff. The POA function satisfies this requirement since the attacker's actual payoff is likely to be proportional to the damage it inflicts on the tree. Moreover, the cost paid by the attacker to overcome a defense will likely be proportional to the sustainability of the defense.

Cost factors play a crucial role in any form of network hardening. At the same time, factors such as potential damage and control costs are subjective to an organization. As highlighted in [34], cost assessment must be a part of any enterprise level system hardening program, overlapping significantly with the capital planning within the organization. An organization must evaluate its assets before and during the enforcement of a mitigation plan. For our simulation, we choose numbers to maintain relative levels of importance between nodes, depending on what information is contained in the node. These numbers do not have to represent actual monetary values to demonstrate the significance of the cost-benefit analysis.

Table 3 Security controls for example network model.

Security Control	Action
SC_1/SC_2	Disable/Patch suid @ 196.216.0.2
SC_3/SC_4	Disable/Patch LICQ @ 196.216.0.2
SC_5	Disable "at" @ 196.216.0.3
SC_6/SC_7	Disable/Patch LICQ @ 196.216.0.3
SC_8	Disable Rsh @ 196.216.0.1
SC_9	Disable Ftp @ 196.216.0.1
SC_{10}	Disconnect Internet @ 196.216.0.1
SC_{11}	Chmod home directory @ 196.216.0.1
SC_{12}/SC_{13}	Disable/Patch Ftp @ 196.216.0.10
SC_{14}/SC_{15}	Disable/Patch SSH @ 196.216.0.10
SC_{16}	Disconnect Internet @ 196.216.0.10
SC_{17}	Disable Rsh @ 196.216.0.10
SC_{18}	Patch FTP/.rhost @ 196.216.0.10
SC_{19}	Chmod home directory @ 196.216.0.10

8 Problem Formulation

The two objectives considered in the multi-objective formulations are the total security control cost and the residual damage in the attack tree of our example network model. For the attack tree corresponding to the example network model, we identified 19 different security controls possible by patching or disabling of different services, as well as by changing file access permissions. With about half a million choices available (2^{19}), an enumerated search would not be an efficient approach to find the optima. The security controls are listed in Table 3. We maintain some relative order of importance between the different services, as in a real-world scenario, when computing the potential damage and security control costs.

Problem 1 *The Single-objective Optimization Problem*

Given an augmented-attack tree $(s_{root}, S, \tau, \varepsilon)|\langle I, V\rangle$ and d security controls, find a vector $\mathbf{T^*} = (T_i^*)$, $T_i^* \in \{0,1\}; 1 \leq i \leq d$, which minimizes the function

$$\alpha RD(\mathbf{T}) + \beta SCC(\mathbf{T})$$

where, α and β are preference weights for the residual damage and the total cost of security control respectively, $0 \leq \alpha, \beta \leq 1$ and $\alpha + \beta = 1$.

The single-objective problem is the most likely approach to be taken by a decision maker. Given only two objectives, a preference based approach might seem to provide a solution in accordance with general intuition. However, as we find in the case of our example network model, the quality of the solution obtained can be quite sensitive to the assignment of the weights. To demonstrate this affect, we run multiple instances of the problem using different combination of values for α and β. α is varied in the range of $[0, 1]$ in steps of 0.05. β is always set to $1 - \alpha$.

Problem 2 *The Multi-objective Optimization Problem*

Given an augmented-attack tree $(s_{root}, S, \tau, \varepsilon)|\langle I, V\rangle$ and d security controls, find a vector $\mathbf{T^*} = (T_i^*)$, $T_i^* \in \{0,1\}; 1 \leq i \leq d$, which minimizes the total security control cost and the residual damage.

The next level of sophistication is added by formulating the minimization as a multi-objective optimization problem. The multi-objective approach alleviates the requirement to specify any weight parameters and hence a better global picture of the solutions can be obtained.

Problem 3 *The Multi-objective Robust Optimization Problem*

Let $\mathbf{T} = (T_i)$ be a boolean vector. A perturbed assignment of radius r, $\mathbf{T_r}$, is obtained by inverting the value of at most r elements of the vector $\mathbf{T}$. The robust optimization problem can then be defined as follows.

Given an augmented-attack tree $(s_{root}, S, \tau, \varepsilon)|\langle I, V\rangle$ and d security controls, find a vector $\mathbf{T^*} = (T_i^*)$, $T_i^* \in \{0,1\}; 1 \leq i \leq d$, which minimizes the total security control cost and the residual damage, satisfying the constraint

$$\max_{\mathbf{T_r}} RD(\mathbf{T_r}) - RD(\mathbf{T}) \leq \mathcal{D}$$

where $\mathcal{D}$ is the maximum perturbation allowed in the residual damage.

The third problem is formulated to further strengthen the decision process by determining robust solutions to the problem. Robust solutions are less sensitive to failures in security controls and hence subside any repeated requirements to re-evaluate solutions in the event of a security control failure. The hardening problem explored here assumes known attacks, the pre- and post-conditions of which are available from vulnerability databases. Unknown attacks can be modeled by introducing likelihoods in the edges of the attack tree. Leaf nodes can be assigned a probability of being *true* (modeling an unknown attack) and Bayesian inference techniques can be used to propagate the likelihoods to the root node. It is worthwhile to note that unknown attacks play a crucial role in the optimality of security policies. Our motivation for robust hardening is grounded on the fact that such attacks can invalidate a security mechanism; however, the potential risk to organizational assets should be bounded in the event of such breaches.

The fourth problem incorporates an estimation of attacker payoffs. In this case, our attempt is to find solutions that are possibly points of equilibrium in the arms race between the attacker and the defender. To do so, we first normalize the POD and POA functions

in order to account for differences arising in the magnitude of the values. The POA function is in the range of $[-ASC(\mathbf{S_D}, \mathbf{S_A}), DI(\mathbf{0}, \mathbf{1})]$ which is remapped to $[0, ASC(\mathbf{S_D}, \mathbf{S_A}) + DI(\mathbf{0}, \mathbf{1})]$ by adding $ASC(\mathbf{S}_D, \mathbf{S_A})$ to the value. POD function is in the non-negative range $[0, SCC(\mathbf{S_D}) + DI(\mathbf{0}, \mathbf{1})]$. The normalized functions for POD and POA – in the range of $[0, 1]$ – is then given as,

$$POD_{norm}(\mathbf{S_D}, \mathbf{S_A}) = \frac{POD(\mathbf{S_D}, \mathbf{S_A})}{SCC(\mathbf{S_D}) + DI(\mathbf{0}, \mathbf{1})} \tag{7}$$

$$POA_{norm}(\mathbf{S_D}, \mathbf{S_A}) = \frac{POA(\mathbf{S_D}, \mathbf{S_A}) + ASC(\mathbf{S_D}, \mathbf{S_A})}{ASC(\mathbf{S_D}, \mathbf{S_A}) + DI(\mathbf{0}, \mathbf{1})} \tag{8}$$

The normalized versions are more intuitive in understanding what the payoff functions model. The defender has an investment worth $SCC(\mathbf{S_D}) + DI(\mathbf{0}, \mathbf{1})$ on the attack tree. POD_{norm} gives the fraction of this investment protected by the defender's strategy for a particular attack strategy. In other words, POD_{norm} gives the fractional return on investment for the defender. From an attacker's perspective, the best it can do is to gather the payoff from maximum damage and also retain the cost incurred while doing so to itself. $DI(\mathbf{S_D}, \mathbf{S_A})$ is the amount that it actually derives. POA_{norm} is thus the fractional return on attack to the attacker.

The defender's optimization problem is to find a defense strategy $\mathbf{S_D}$ that gives maximum POD_{norm} under all possible attack strategies. The attacker's optimization problem is to find an attack strategy $\mathbf{S_A}$ that gives maximum POA_{norm} under all possible defense strategies. However, such a strategy may not exist. Besides, as argued earlier, evaluating a host strategy with all opponent strategies is often impractical. We introduce here the terms *host* and *opponent* to refer to the party whose strategy is being tested and the party against whom it is being tested respectively. In order to compare two host strategies, it is sufficient to evaluate them against their respective best opponent strategy (one generating the highest payoff for the opponent with the host strategy in place). Hence, a more suitable statement of the optimization problem is as follows.

Problem 4 *The Attacker-Defender Arms Race Problem*

Defender's Optimization Problem: Given an augmented attack tree AT_{aug} and d defenses, find the defense strategy $\mathbf{S_D}^*$ that maximizes $POD_{norm}(\mathbf{S_D}, \mathbf{S_A}^*)$, where $\mathbf{S_A}^*$ satisfies the relation $POA_{norm}(\mathbf{S_D}, \mathbf{S_A}^*) \geq POA_{norm}(\mathbf{S_D}, \mathbf{S_A})$ for any attack strategy $\mathbf{S_A}$.

Attacker's Optimization Problem: Given an augmented attack tree AT_{aug} and d defenses, find the attack strategy $\mathbf{S_A}^*$ that maximizes $POA_{norm}(\mathbf{S_D}^*, \mathbf{S_A})$, where $\mathbf{S_D}^*$ satisfies the relation $POD_{norm}(\mathbf{S_D}^*, \mathbf{S_A}) \geq POD_{norm}(\mathbf{S_D}, \mathbf{S_A})$ for any defense strategy $\mathbf{S_D}$.

The brute force method to solve each problem is to first generate the payoff matrix and then mark the cell, for every host strategy, with the highest opponent payoff. The solution is the host strategy which has the highest payoff in the marked cells. If, given the host strategy in the solution, the opponent's payoff is also the highest, and vice versa, then the solution admits a *Nash equilibrium* [39]. We want to emphasize here that solving just one problem is not sufficient. For example, assume that the defender has found the optimal solution to its problem. The POD_{norm} reported by the solution implicitly assumes that the attacker will launch the strategy $\mathbf{S_A}^*$ that gives the highest attacker payoff – established in the optimization problem by the constraint. If the attacker also solves its own optimization problem, there is no guarantee that the best strategy found by it is the same $\mathbf{S_A}^*$ as found in solving the defender's optimization problem. The outcome in this case could be that both the attacker and the defender get sub-optimal payoffs. This instantiates the requirement to solve both problems simultaneously, the desired solution being the aforesaid equilibrium. The equilibrium defense and attack strategy pair $\mathbf{S_D}^*$ and $\mathbf{S_A}^*$ satisfy the conditions

1. $POD_{norm}(\mathbf{S_D}^*, \mathbf{S_A}^*) > POD_{norm}(\mathbf{S_D}, \mathbf{S_A}^*)$ and
2. $POA_{norm}(\mathbf{S_D}^*, \mathbf{S_A}^*) > POA_{norm}(\mathbf{S_D}^*, \mathbf{S_A})$

for any given defense strategy $\mathbf{S_D}(\neq \mathbf{S_D}^*)$ and attack strategy $\mathbf{S_A}(\neq \mathbf{S_A}^*)$.

9 Specifics of Solution Methods

We use a simple genetic algorithm (SGA) [40] to solve Problem 1. The Non-dominated Sorting Genetic Algorithm-II (NSGA-II) [24] is used to solve Problem 2 and 3. The method of competitive co-evolution is used to find solutions to Problem 4.

9.1 NSGA-II

NSGA-II starts with a population P_0 of N randomly generated security control vectors $\mathbf{T}$. For each trial solution, the total security control cost is calculated using Definition 11. To compute the residual damage, the attributes covered by a security control vector in the attack tree are decided using Table 3 and set to *false*. The truth values for the remaining attributes in $N_{external}$ are set to *true*. A post-order traversal of the tree is then used to determine the truth values of the internal

nodes using the decomposition at each node. This enables us to compute the value V_{root} for the root node (the residual damage) using (3).

A generation index $t = 0, 1, \ldots, Gen_{MAX}$ keeps track of the number of iterations of NSGA-II. Each generation of the algorithm then proceeds as follows. An offspring population Q_t is first created from the parent population P_t by applying the usual genetic operations of selection, crossover and mutation [40]. The residual damage and total security control cost corresponding to each solution in the child population are also computed.

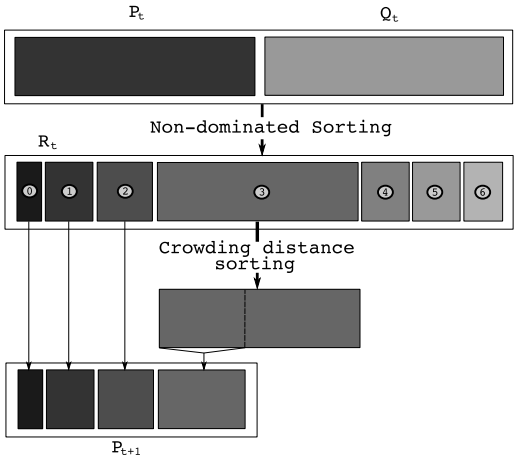


Fig. 6 One generation of NSGA-II.

The parent and offspring populations are combined to form a population $R_t = P_t \cup Q_t$ of size $2N$. A non-dominated sorting is applied to R_t to rank each solution based on the number of solutions that dominate it. A rank $k(> 0)$ solution is dominated by solutions of rank lower than k. For Problem 3, the solutions which violate the robustness constraint, i.e. an infeasible solution, are given unique ranks higher than the highest feasible solution rank. The ranking starts in ascending order from the infeasible solution with least constraint violation.

The population P_{t+1} is generated by selecting N solutions from R_t. The preference of a solution is decided based on its rank: lower the rank, higher the preference. However, since not all solutions from R_t can be accommodated in P_{t+1}, a choice is likely to be made when the number of solutions of the currently considered rank is more than the remaining positions in P_{t+1}. Instead of making an arbitrary choice, NSGA-II uses an explicit diversity-preservation mechanism. The mechanism, based on a *crowding distance metric* [24], gives more preference to a solution with a lesser density of solutions surrounding it, thereby enforcing diversity in the population. The NSGA-II crowding distance metric for a solution is the sum of the average side-lengths of the cuboid generated by its neighboring solutions in objective space. Figure 6 depicts a single generation of the algorithm.

The algorithm parameters are set as follows: population size = 100, number of generations = 250, crossover probability = 0.9, and mutation probability = 0.1. We ran each instance of the algorithm five times to check for any sensitivity of the solutions obtained from different initial populations. Since the solutions always converged to the same optima, we dismiss the presence of such sensitivity.

9.2 Competitive Co-evolution

We begin with two randomly generated populations Pop_A and Pop_D of size N_A and N_D respectively. Pop_A refers to the population of attack strategies $\{\mathbf{S_A}^1, \ldots, \mathbf{S_A}^{N_A}\}$ and Pop_D refers to that of defense strategies $\{\mathbf{S_D}^1, \ldots, \mathbf{S_D}^{N_D}\}$. In every generation, every strategy in a population is evaluated with the best opponent strategy (one with highest fitness as described later) of the previous generation to find POA_{norm} and POD_{norm}. The notations $\mathbf{S_D}^{prevbest}$ and $\mathbf{S_A}^{prevbest}$ is used to denote the best defense and attack strategy from the previous generation respectively. For the first generation, the best strategies are chosen randomly from the populations.

Next, each strategy in the populations is assigned an *age count,* $Age(\cdot)$, signifying the number of iterations for which it has survived the evolutionary process. Each strategy begins with an age count of zero which is incremented every time it manages to enter the next population. The age is reset to zero if the strategy no longer exists in the next population. With this, the fitness of a defense strategy $\mathbf{S_D}^i$ in generation (iteration) t is assigned as

$$F(\mathbf{S_D}^i, t) = \frac{F(\mathbf{S_D}^i, t-1) \times Age(\mathbf{S_D}^i) + POD_{norm}(\mathbf{S_D}^i, \mathbf{S_A}^{prevbest})}{[Age(\mathbf{S_D}^i) + 1]} \tag{9}$$

and that of an attack strategy $\mathbf{S_A}^j$ in generation t is assigned as

$$F(\mathbf{S_A}^j, t) = \frac{F(\mathbf{S_A}^j, t-1) \times Age(\mathbf{S_A}^j) + POA_{norm}(\mathbf{S_D}^{prevbest}, \mathbf{S_A}^j)}{[Age(\mathbf{S_A}^j) + 1]} \tag{10}$$

The fitness is an average measurement of the payoff of a strategy throughout the evolutionary process. With this fitness assignment, each population then independently undergoes the usual process of evolution as in a

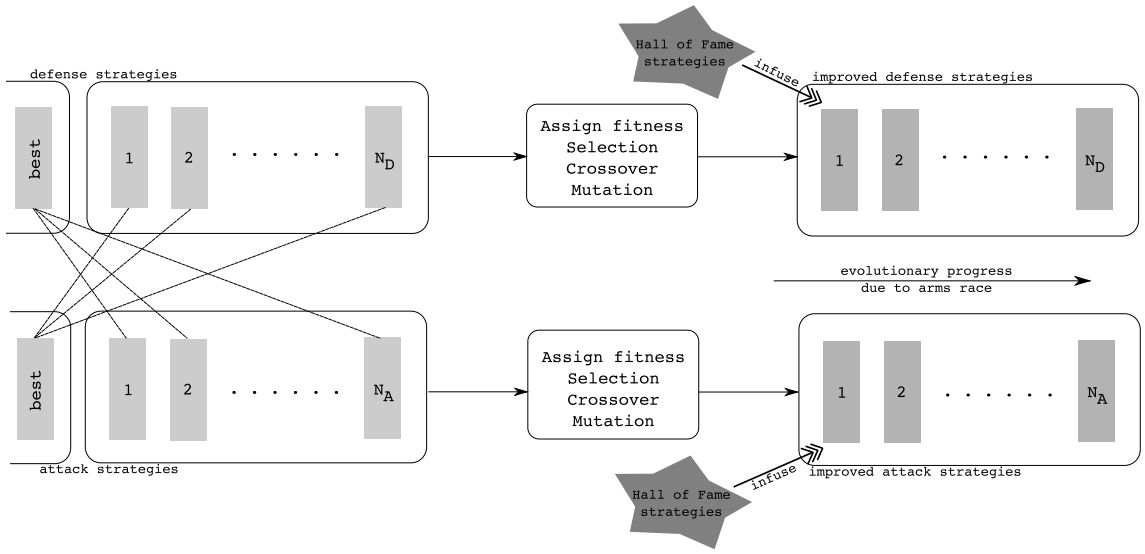


Fig. 7 Schematic of competitive co-evolution of attack and defense strategies.

genetic algorithm (GA) – selection, crossover and mutation [40] – and creates a new population of strategies. The best strategies of the past H generations replace H randomly selected strategies in the respective populations. The process is repeated until a set number of generations. Figure 7 depicts the algorithm. The parameters of the algorithm are set as follows: $N_A = 100$, $N_D = 100$, $H = 10$, single point crossover with probability 0.5, probability of mutation = 0.01, 2-tournament selection and 1000 generations. In the experiments we use the 19 defenses shown in Table 3 and the attack tree has 13 unique leaf nodes. The defender thus has 2^{19} defense strategies and the attacker has 2^{13} attack strategies to choose from.

10 Empirical Results

We first present the sensitivity results of NSGA-II and SGA to their parameters in the multi-objective problem solution methods. Increasing the population size from 100 to 500 gives us a faster convergence rate, although the solutions reported still remains the same. The effect of changing the crossover probability in the range of 0.7 to 0.9 does not lead to any significant change of the solutions obtained. Similar results were observed when changing the mutation probability from 0.1 to 0.01. The solutions also do not change when the number of generations is changed from 250 to 500. Since we did not observe any significant change in the solutions by varying the algorithm parameters, the following results are presented as obtained by setting the parameters as chosen in the previous section. For competitive co-evolution, some parameters involved in the GA affect the dynamics of the arms race undergoing between the two populations. Using a higher probability of crossover or mutation affects the age count of a solution. A high probability decreases the chances of a strategy surviving for long across iterations, thereby interrupting its chances of competing against a wider variety of opponent strategies. Increasing the population size gives a faster convergence rate, although the solution remains unaffected. We also increased the number of iterations from 1000 to 5000 to see if a dormant strategy becomes prominent over time. However, no such outcome is observed.

10.1 Problem 1: Single-objective optimization

It is usually suggested that the preference based approach should normalize the functions before combining them into a single function. However, we did not see any change in the solutions of the normalized version of Problem 1. Figure 8 shows the solutions obtained from various runs of SGA in Problem 1 with varying α. A decision maker, in general, may want to assign equal weights to both the objective functions, i.e. set $\alpha = 0.5$. It is clear from the figure that such an assignment do not necessarily provide the desired balance between the residual damage and the total security control cost. Furthermore, such balance is also not obtainable by assigning weight values in the neighborhood of 0.5. The solutions obtained are quite sensitive to the weights, and in this case, much higher preference must be given to the total security control cost to find other possible solutions. Since the weights do not always influence the objectives in the desired manner, understanding their

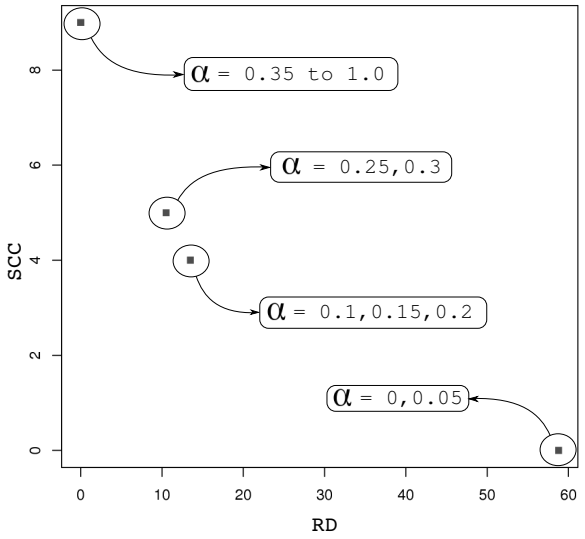


Fig. 8 SGA solutions to Problem 1 with α varied from 0 to 1 in steps of 0.05.

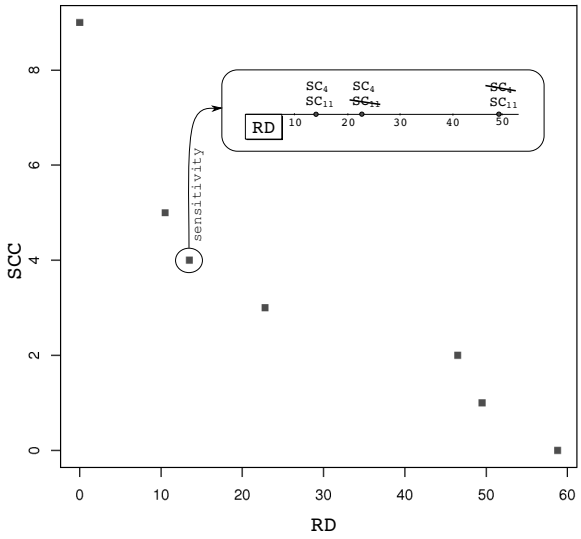


Fig. 9 NSGA-II solutions to Problem 2 and sensitivity of a solution to optimum settings.

effect is not a trivial task for a decision maker. It is also not possible to always do an exhaustive analysis of the affect of the weights on the objectives. Given such situations, the decision maker should consider obtaining a global picture of the trade-offs possible. With such a requirement in mind, we next consider Problem 2.

10.2 Problem 2: Multi-objective optimization

The two solutions corresponding to $\alpha = 0.25$ and 0.1 in Figure 8, including any other solution in the vicinity, are likely candidates for a decision maker's choice. Unlike the single-objective approach, where determining such vicinal solutions could be difficult, the multi-objective optimization approach clearly revealed the existence of at least one such solution. Figure 9 shows the solutions obtained from a single run of NSGA-II on Problem 2. NSGA-II reported all the solutions obtained from multiple runs of SGA, as well as three more solutions. Interestingly, there exists no solution in the intermediate range of $[25, 45]$ for residual damage. This inclination of solutions towards the extremities of the residual damage could be indicative of the non-existence of much variety in the security controls under consideration. The number of attack scenarios possible is also a deciding factor. Most of the security controls for the example network involve either the disabling or patching of a service, resulting in a sparse coverage matrix. For a more "continuous" Pareto-front, it is required to have security controls of comparative costs and capable of covering multiple services. A larger, more complex real-world problem would likely have more attack scenarios and a good mixture of both local and global security controls, in which case, such gaps in the Pareto-front will be unlikely.

10.3 Problem 3: Robust optimization

Once the decision maker has a better perspective of the solutions possible, further analysis of the solutions may be carried out in terms of their sensitivity to security control failures. Such sensitivity analysis is helpful in not only reducing valuable decision making time, but also to guarantee some level of fault tolerance in the network. Figure 9 shows the sensitivity of one of the solutions to a failure in one of the security controls corresponding to the solution. This solution, with security controls SC_4 and SC_{11}, will incur a high residual damage in the event of a failure of SC_4. Thus, a decision maker may choose to perform a sensitivity analysis on each of the solutions and incorporate the results thereof in making the final choice. However, the decision maker then has no control on how much of additional residual damage would be incurred in the event of failure. Problem 3 serves the requirements of this decision stage by allowing the decision maker to specify the maximum allowed perturbation in the residual damage. It is also possible to specify the scope of failure – the radius r – within which the decision maker is interested in analyzing the robustness of the solutions. For this study,

Table 4 Fully robust solutions obtained by NSGA-II with $r = 1$.

	Robust-optimum security controls	RD	SCC
R1	SC_9, SC_{11}, SC_{13}, SC_{15}, SC_{16}, SC_{19}	0.0	26.0
R2	SC_3, SC_4, SC_9, SC_{11}, SC_{18}, SC_{19}	10.5	21.0
R3	SC_3, SC_4, SC_7, SC_{11}	13.5	12.0
R4	SC_3, SC_4	22.8	8.0
R5	SC_7, SC_{11}	49.5	4.0
R6	*null*	58.8	0.0

we are mostly interested in obtaining solutions that are fully robust, meaning the residual damage should not increase, and hence set $\mathcal{D}$ to zero. Also, because of the sparse nature of the coverage matrix, we set the perturbation radius r to 1. Figure 10 shows the solutions obtained for this problem.

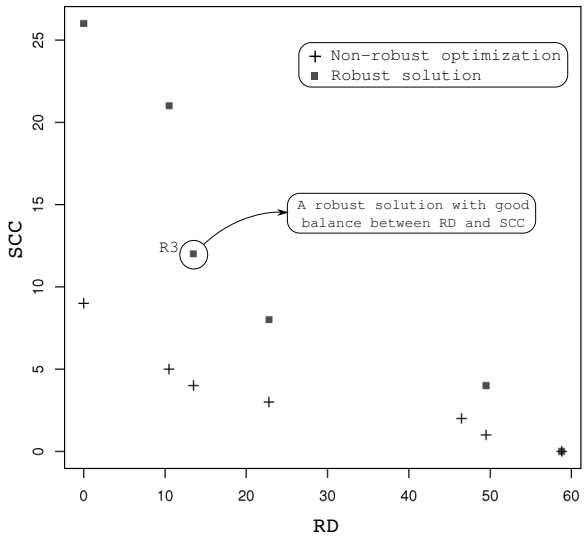


Fig. 10 NSGA-II solutions to Problem 3 with $\mathcal{D} = 0$ and $r = 1$. Problem 2 solutions are also shown for comparison.

The solutions to Problem 3 reveal that none of the optimum solutions previously obtained, except the trivial zero SCC solution, is fully robust even for a single security control failure. Such insight could be of much value for a decision maker when making a final choice. Table 4 shows the security controls corresponding to the robust solutions. With the final goal of obtaining a solution with a good balance between the residual damage and the total security control cost, the decision maker's choice at this point can be justifiably biased towards the selection of solution R3.

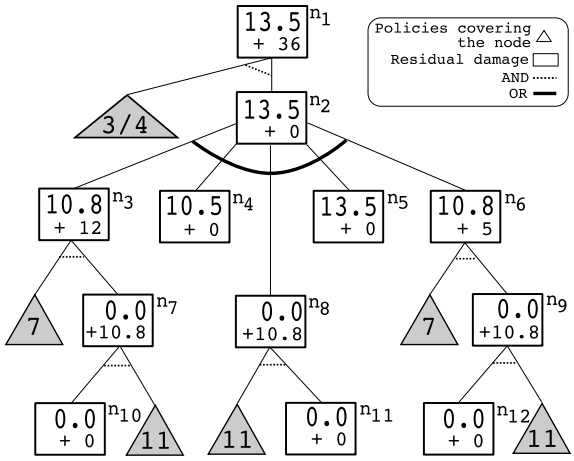


Fig. 11 Compressed attack tree showing residual damage computation with R3 as security control set.

10.3.1 Inside the Robust Solution R3

We present certain interesting properties exploited by solution R3 from the attack tree. To point out the salient features, we compress the attack tree for our example network model as shown in Figure 11. The compressed tree is obtained by collapsing all subtrees to a single node until a node covered by a security control from R3 contributes to the calculation of the residual damage. All such nodes, represented by rectangles in the figure, are labeled with the maximum residual damage that can propagate to them from the child subtree and (+) the damage value that can occur at the node itself. A triangular node represents the security controls that can disable that node. The individual damage value is accrued to the residual damage from the child node only if the attached security control, if any, fails.

The solution R3 clearly identifies the existence of the subtrees $ST_1 = \{\{n_7, n_{10}\}, \{n_8, n_{11}\}, \{n_9, n_{12}\}\}$ and $ST_2 = \{\{\ n_3, n_7, n_{10}\}, \{n_6, n_9, n_{12}\}\}$. In the event of a failure of SC_{11}, n_7 would collect a value of 10.8. Since n_3 has an AND decomposition with SC_7, it will be disabled, thereby not contributing its individual damage value of 12 to the residual damage at that node (10.8). On the other hand, if SC_7 fails, SC_{11} will disable n_7 which in turn will disable n_3. In fact, in this case the residual damage at n_3 would be zero. Similarly, n_6 and n_8 also never propagate a residual damage of more than 10.8 to its parent node. Consequently, n_2 never propagates a value more than 13.5. The individual cost of 36 at n_1 is never added to this residual damage value of 13.5 from n_2 since, owing to the AND decomposition, n_1 is always falsified by security controls SC_3 and SC_4, only one of which is assumed to fail at a time. The solution wisely applies security controls covering mul-

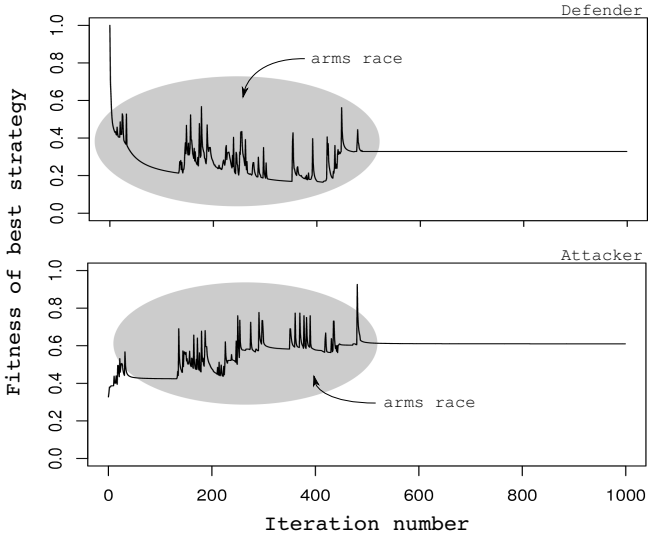


Fig. 12 Fitness of best defense and attack strategies in every iteration of the co-evolutionary process.

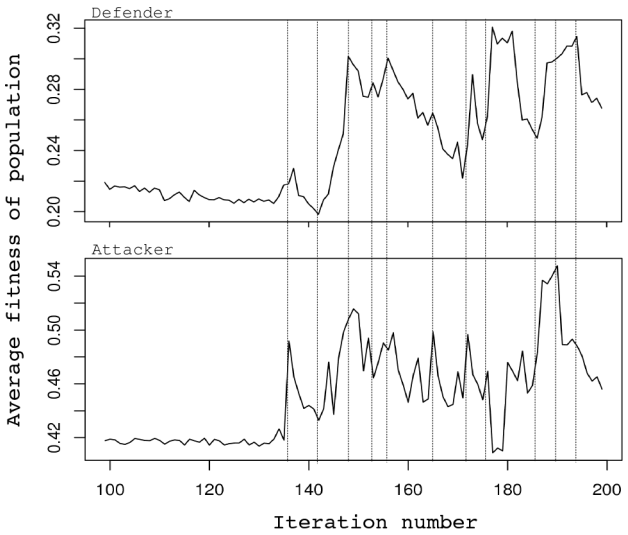


Fig. 13 Average fitness of defender and attacker strategies showing "arms race" population dynamics between the 100^{th} and the 200^{th} generations.

tiple attack scenarios, and at multiple points in those scenarios to keep the damage to a minimum.

10.4 Problem 4: Arms race

Figure 12 shows how the fitness of the best strategy in the defender and attacker populations change across generations. The random initialization of the two populations usually starts off the competition with comparatively higher fractional payoff for the defender. However, the attacker immediately finds strategies to improve its payoff, and reciprocally decreases the payoff for the defender, as can be seen on the steep decline of the defender's payoff. There is even a phase between the 50^{th} to 150^{th} generations when the attacker continued to evolve strategies with similar payoff, but ones that continued to decrease the payoff for the defender. The arms race becomes prominent after this phase. The arms race is indicative of the period when the defender and the attacker continuously change their strategies to cease the decline in their payoffs brought forth by an improved opponent strategy. In a way, this arms race depicts the change in policies that the defender has to sporadically keep enforcing in order to subdue the affects of an evolving attacker.

10.4.1 Dynamics of the Arms Race

Figure 13 depict the dynamics of the two populations during the 100^{th} to the 200^{th} generations. The average fitness of each population is plotted to show the interactions happening between them. The arms race is distinctly visible after the 130^{th} generation – one population reacts to changes in the other. Rising up to a peak indicates the phase of steady improvement in host strategies against those of the opponent's. Falling down to a pit signifies the reverse. As depicted by the vertical lines, a rising period in one population results in a falling period in the other, and vice versa. Note that the rise in one population and the fall in the other are not correlated in terms of the payoff values. An attacker's marginal improvement in payoff can result in a significant drop in the defender's payoff. More interestingly, there is no fixed duration within which the two populations alternate between rise and fall. In other words, the dynamics of finding a strategy to tackle the currently dominating opponent strategy is not known. We stress on this phenomena since any defense strategy not in equilibrium with that of the attacker eventually results in a decline in the payoff. Ideally, the better the strategy, the slower will be the decline; emphasizing that the attacker faces more difficulty in finding a counter strategy to improve its payoff.

However, with the static attack tree in place, the process of arms race do not continue forever. Both the attack and the defense strategies stabilize at around the 500^{th} generation. No host at this point manages to find a strategy to improve its payoff given the best strategy the opponent has at the point. However, this stability in the strategies is not sufficient to conclude that the attacker and defender are now in an equilibrium. This follows from the fact that there may exist an undiscovered opponent strategy that can reduce the payoff generated from the stable host strategy.

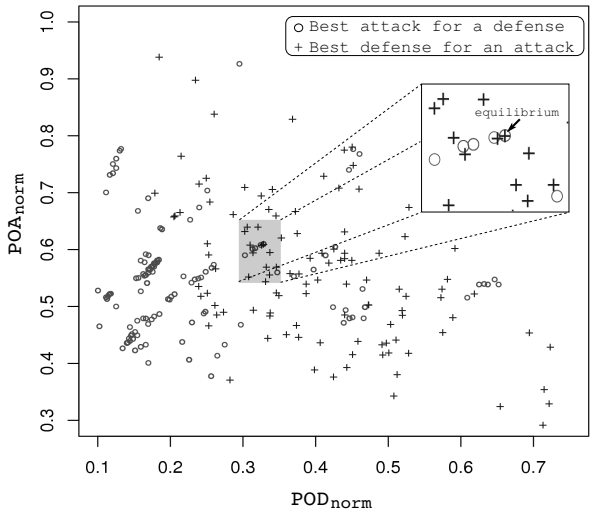


Fig. 14 Payoffs when an independent GA is run for each best host strategy from every generation to find the best opponent strategy for it. **o**/+ represent the payoffs when the defender/attacker is the host.

10.4.2 Verifying an Equilibrium Point

In order to demonstrate the effectiveness of competitive co-evolution in generating an equilibrium strategy pair, we perform the following supplementary analysis. The defender's best strategy $\mathbf{S_D}^{best_t}$ in every generation t ($1 \leq t \leq 1000$) of the process is noted. For each such strategy we run a simple genetic algorithm to generate the attack strategy $\mathbf{S_A}^{best_t}$ with the highest attacker payoff. Figure 14 shows the defender and attacker payoffs (in circles) for the pairs $\mathbf{S_D}^{best_t}$ and $\mathbf{S_A}^{best_t}$. A similar process is done taking the attacker's best strategy of every generation. The plus signs in the plot depict the payoffs for the pairs obtained from the process. We find that the only circle and plus coinciding corresponds to the stable strategy of the defender and the attacker as returned by the co-evolutionary optimization. If the defender chooses the stable defense strategy, the attacker's payoff is maximum when it uses the stable attack strategy. If the attacker uses the stable attack strategy, the defender's payoff is maximum when it uses the stable defense strategy. In other words, the stable defense and attack strategy pair is indeed an equilibrium point.

10.5 Revisiting the Optimality Criterion

One of the forthcoming questions resulting from this analysis is whether an organization's investments be directed towards a static minimal cost security policy or proactively be channeled towards an equilibrium policy. We have argued in this work that the conventional notion of an optimal security policy (a minimum cost policy) ignores the possibility of constraints in terms of available resources to implement the security controls. Pareto analysis of the nature performed here can be instead used to identify a minimal policy depending on the resource constraints of the organization. In this case, the optimality criterion is represented by the non-dominance characteristic of the Pareto solutions. Robust hardening limits the extent of unforeseen damage that can be inflicted on the system. However, we are still working under the assumption of fixed attacker capabilities.

The minimal policy resulting from a one-time evaluation may incur a lower cost with respect to a short time window, but under an evolving attacker model, this cost must be supplemented by further investments over time. We emphasize that the evolution of attack strategies need not relate to a single active attacker. The attack strategies at different evolution points might as well be executed by different attackers. The evolving strategies are a platform to demonstrate how the optimality of an organization's security policies is invalidated over time (again and again) due to the constant engagement of the attacking entities. Under such grounds, the optimality of the chosen policy can no longer be guaranteed. This can have a serious impact on business dynamics, since business models are often driven by investment returns. We have demonstrated using a game-theoretic analysis of the security hardening problem that an optimal security policy converges with the idea of equilibrium points in the long run. This optimality criterion ensures that an organization's resources are not spent in intermediate policies that are likely to undergo changes as attacker capabilities evolve over time.

While multiple attempts to define the optimality criterion have been made (including the work here), we believe these definitions cannot be complete without the inclusion of an evolving defender. It is important to note that security hardening is an on-going process. Therefore, the optimality of a policy should not only be dependent on the amount of risk it can eliminate (or how much it costs), but also on how much it deviates from existing policies. In other words, reusability of already invested resources needs to be stressed. A minimal policy (in the Pareto sense) should be minimally dissimilar from the current policy; a factor that can be easily incorporated into the non-dominance based definition of optimality. An equilibrium policy, by definition, needs no revision. However, this is only true when the network characteristics do not change over a long period of time. Any change in the network can invalidate the equilibria of a policy since the equilibrium conditions hold only for a particular snapshot of the network. Other defini-

tions of optimality also suffer from this drawback. We believe this necessitates an approach where an evolving defense model and an evolving network model is integral to the definition of an optimal security policy. Ideally, such an approach should target a trade-off between the short-term gains achievable by evaluating a snapshot of the network, and the long-term gains achievable by considering the evolving attacker capabilities (as in an equilibrium policy).

11 Conclusion and Future Work

Incorporating strong defenses against malicious attackers is challenging. Simply installing the best available defenses does not work for several reasons. The security administrator has to work within fixed budgetary constraints and has to explain the return on investment of security controls to the management. However, any convincing argument explaining the return on investment must take the attacker's benefits into consideration.

In this paper, we addressed the system administrator's dilemma, namely, how to select a subset of security hardening measures from a given set so that the total cost of implementing these measures is not only minimized but also within budget and, at the same time, the cost of residual damage is also minimized. One important contribution of our approach is the use of an attack tree model of the network to drive the solution. By using an attack tree in the problem we were able to better guide the optimization process by providing the knowledge about the attributes that make an attack possible. Further, a systematic analysis enabled us to approach the problem in a modular fashion, providing added information to a decision maker to form a concrete opinion about the quality of the different trade-off solutions possible.

We argue that the notion of optimal security hardening is often dictated by the constant interaction between the defender and the attacker. What is perceived as the optimal return on investment would cease to be so once the attacker's strategy to exploit the defensive configuration is understood. We highlight that the dynamic engagement between the attacker and the defender is a continuous process ending only when both enter a state of equilibrium. To this end, we formulate the requisite optimization problems and present the notion of equilibrium in terms of the formulated problems. As a viable methodology, we propose the use of competitive co-evolution to generate the aforementioned equilibrium strategies. The method involves an algorithm that intrinsically models the arms race undergoing between the attacker and the defender, with the ability to effectively find the equilibrium solutions.

Evolutionary algorithms often receive criticism for their time complexity, compared to other optimization methods. The multi-objective algorithm used in this study has a complexity of $\mathrm{O}(GN \log N)$, where G is the number of generations and N is the population size. However, the population based approach also makes it highly suitable for discovering multiple solution points on the Pareto-front. These algorithms are inherently parallel and can easily be adapted to utilize the processing power of most massively parallel systems [41]. The evolutionary algorithm is one viable methodology for the multi-objective optimization that we can think of at this moment. There is no doubt that more efficient methods are required. A similar argument applies to the competitive co-evolution algorithm as well. At this point, we are not aware of a more efficient method to explore the arms race. We strongly believe this would motivate some future studies in this area.

The cost model that we adopt in this paper is somewhat simplistic. We assume that, from a cost of implementation perspective, the security measures are independent of each other when in real life they may not be so. The problem can be made more interesting by designing payoff models that incorporate multiple attackers working in conjunction to achieve a particular objective (collaborative attacks). Incorporating network connectivity and the trust relations across organizations into the attack tree will generate far more complex attack scenarios. These forms of attacks are very likely in today's networked infrastructure and warrant a further study. Further work can be directed towards designing algorithms that can identify the existence of multiple equilibrium points simultaneously. We believe that Pareto analysis intended towards generation of such solutions is a promising avenue to explore. Formal analysis to determine if equilibrium solutions exists at all would be a major contribution as well. Furthermore, the possible decomposition of an attack tree to divide the problem into sub-problems is an interesting alternative to explore. Finally, exploring the optimality of security policies under the light of changing network characteristics and attacker capabilities remains one of the most challenging problems in this domain.

Acknowledgements This work was partially supported by the U.S. Air Force Office of Scientific Research under contracts FA9550-07-1-0042. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing official policies, either expressed or implied, of the U.S. Air Force or other federal government agencies.

References

1. P. Ammann, D. Wijesekera, and S. Kaushik. Scalable, Graph-Based Network Vulnerability Analysis. In *Proceedings of the 9th Conference on Computer and Communications Security*, pages 217–224, 2002.
2. S. Jha, O. Sheyner, and J. M. Wing. Two Formal Analysis of Attack Graphs. In *Proceedings of the 15th IEEE Computer Security Foundations Workshop*, pages 49–63, 2002.
3. C. Phillips and L. P. Swiler. A Graph-Based System for Network-Vulnerability Analysis. In *Proceedings of the 1998 New Security Paradigms Workshop*, pages 71–79, 1998.
4. O. Sheyner, J. Haines, S. Jha, R. Lippmann, and J. M. Wing. Automated Generation and Analysis of Attack Graphs. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 273–284, 2002.
5. L. Swiler, C. Phillips, D. Ellis, and S. Chakerian. Computer-Attack Graph Generation Tool. In *Proceedings of the DARPA Information Survivability Conference and Exposition II*, pages 307–321, 2001.
6. J. Dawkins, C. Campbell, and J. Hale. Modeling Network Attacks: Extending the Attack Tree Paradigm. In *Proceedings of the Workshop on Statistical Machine Learning Techniques in Computer Intrusion Detection*. Johns Hopkins University, 2002.
7. A. P. Moore, R. J. Ellison, and R. C. Linger. Attack Modeling for Information Survivability. Technical Note CMU/SEI-2001-TN-001, Carnegie Melon University / Software Engineering Institute, March 2001.
8. I. Ray and N. Poolsappasit. Using Attack Trees to Identify Malicious Attacks from Authorized Insiders. In *Proceedings of the 10th European Symposium On Research In Computer Security*, pages 231–246, 2005.
9. B. Schneier. Attack Trees. *Dr. Dobb's Journal*, 1999.
10. S. Noel, S. Jajodia, B. O'Berry, and M. Jacobs. Efficient Minimum-cost Network Hardening via Exploit Dependency Graphs. In *Proceedings of the 19th Annual Computer Security Applications Conference*, pages 86–95, 2003.
11. R. Dewri, N. Poolsappasit, I. Ray, and D. Whitley. Optimal Security Hardening Using Multi-objective Optimization on Attack Tree Models of Networks. In *Proceedings of the 14th Conference on Computer and Communications Security*, pages 204–213, 2007.
12. M. Gupta, J. Rees, A. Chaturvedi, and J. Chi. Matching Information Security Vulnerabilities to Organizational Security Policies: A Genetic Algorithm Approach. *Decision Support Systems*, 41(3):592–603, 2006.
13. S. Bistarelli, M. Dall'Aglio, and P. Perretti. Strategic Games on Defense Trees. In *Formal Aspects in Security and Trust*, pages 1–15. Springer, 2006.
14. P. F. Syverson. A Different Look at Secure Distributed Computation. In *Proceedings of the 10th Computer Security Foundations Workshop*, pages 109–115, 1997.
15. K. Lye and J. M. Wing. Game Strategies in Network Security. *International Journal of Information Security*, 4(1-2):71–86, 2005.
16. K. Sallhammar, S. J. Knapskog, and B. E. Helvik. Using Stochastic Game Theory to Compute the Expected Behavior of Attackers. In *Proceedings of the 2005 Symposium on Applications and the Internet Workshops*, pages 102–105, 2005.
17. K. Sallhammar, B. E. Helvik, and S. J. Knapskog. Towards a Stochastic Model for Integrated Security and Dependability Evaluation. In *Proceedings of the First International Conference on Availability, Reliability and Security*, pages 156–165, 2006.
18. P. Liu, W. Zang, and M. Yu. Incentive-Based Modeling and Inference of Attacker Intent, Objectives, and Strategies. *ACM Transactions on Information and System Security*, 8(1):78–118, 2005.
19. A. Buldas, P. Laud, J. Priisalu, M. Saarepera, and J. Willemson. Rational Choice of Security Measures Via Multi-parameter Attack Trees. *Critical Information Infrastructures Security*, 4347:235–248, 2006.
20. Z. Zhang, F. Nait-Abdesselam, and P. Ho. Boosting Markov Reward Models for Probabilistic Security Evaluation by Characterizing Behaviors of Attacker and Defender. In *Proceedings of the 3rd International Conference on Availability, Reliability and Security*, pages 352–359, 2008.
21. W. Jiang, H. Zhang, Z. Tian, and X. Song. A Game Theoretic Method for Decision and Analysis of the Optimal Active Defense Strategy. In *Proceedings of the 2007 International Conference on Computational Intelligence and Security*, pages 819–823, 2007.
22. C. A. Coello Coello. An Updated Survey of GA-based Multiobjective Optimization Techniques. *ACM Computing Surveys*, 32(2):109–143, 2000.
23. K. Deb. *Multi-objective Optimization Using Evolutionary Algorithms*. John Wiley & Sons Inc., 2001.
24. K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A Fast and Elitist Multiobjective Genetic Algorithm: NSGA–II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, 2002.
25. R. Axelrod. Evolution of Strategies in the Iterated Prisoner's Dilemma. In *Genetic Algorithms and Simulated Annealing*, pages 32–41. Morgan Kaufmann, 1987.
26. J. M. Smith. *Evolution and the Theory of Games*. Cambridge University Press, 1982.
27. C. D. Rosin and R. K. Blew. Methods for Competitive Co-Evolution: Finding Opponents Worth Beating. In *Proceedings of the 6th International Conference on Genetic Algorithms*, pages 373–381, 1995.
28. W. D. Hillis. Co-evolving Parasites Improve Simulated Evolution as an Optimization Procedure. In *Artificial Life II*. Addison-Wesley, 1991.
29. L. Bull. Coevolutionary Computation: An Introduction. www.cems.uwe.ac.uk/ lbull/intro.pdf, 1998.
30. R. Dawkins. *The Blind Watchmaker*. Norton & Company, Inc., 1986.
31. C. D. Rosin and R. K. Blew. New Methods for Competitive Coevolution. *Evolutionary Computation*, 5(1):1–29, 1997.
32. S. G. Ficici and J. B. Pollack. A Game-Theoretic Memory Mechanism for Coevolution. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 286–297, 2003.
33. K. O. Stanley and R. Miikkulainen. The Dominance Tournament Method of Monitoring Progress in Coevolution. In *Proceedings of the Genetic and Evolutionary Computation Conference Workshop Program*, pages 242–248, 2002.
34. G. Stoneburner, A. Goguen, and A. Feringa. Risk Management Guide for Information Technology Systems. *NIST Special Publication 800–30*, 2002.
35. B. Berger. Data-centric Quantitative Computer Security Risk Assessment. *Information Security Reading Room, SANS*, 2003.
36. W. Lee. Toward Cost-sensitive Modeling for Intrusion Detection and Response. *Journal of Computer Security*, 10(1):5–22, 2002.

37. S. A. Butler. Security Attribute Evaluation Method: A Cost-benefit Approach. In *Proceedings of the 24rd International Conference on Software Engineering*, pages 232–240, 2002.
38. S. A. Butler and P. Fischbeck. Multi-attribute Risk Assessment. In *Proceedings of SREIS02 in conjunction with the 10th IEEE International Requirements Engineering Conference*, 2002.
39. J. Nash. Non-Cooperative Games. *The Annals of Mathematics*, 54(2):286–295, 1950.
40. D. E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, 1989.
41. E. Alba and M. Tomassini. Parallelism and Evolutionary Algorithms. *IEEE Transactions on Evolutionary Computation*, 6(5):443–462, 2002.