Optimal Security Hardening Using Multi-objective Optimization on Attack Tree Models of Networks

Rinku Dewri, Nayot Poolsappasit, Indrajit Ray and Darrell Whitley
Department of Computer Science
Colorado State University
Fort Collins, CO 80523, USA
{rinku,nayot,indrajit,whitley}@cs.colostate.edu

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. Many of them also addressed the problem of minimizing the total cost of implementing these hardening measures, given costs for individual measures. 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. Their problem is how to select a subset of security hardening measures so as to be within the budget and yet minimize the residual damage to the system caused by not plugging all required security holes. In this work, we develop a systematic approach to solve this problem by formulating it as a multi-objective optimization problem on an attack tree model of the system and then use an evolutionary algorithm to solve it.

Categories and Subject Descriptors

C.2.3 [**Computer-Communication Network**]: Network Operations—*Network management*; C.2.0 [**Computer-Communication Network**]: General—*Security and protection*

General Terms

Security

Keywords

Security management, Attack trees, Multi-objective optimization

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.

With cost-effectiveness occurring as a major factor in deciding the extent to which an organization would secure its network, it is not sufficient to detect the presence or absence of a vulnerability and implement a security measure to rectify it. Further analysis is required to understand the contribution of the vulnerabilities towards any possible damage to the organization's assets. Often, vulnerabilities are not exploited in isolation, but rather used in groups to compromise a system. Similarly, security policies can have a coverage for multiple vulnerabilities. Thus, cost-effective security management requires researchers to evaluate the different scenarios that can lead to the damage of a secured asset, and then come up with an optimal set of security policies to defend such assets.

Researchers have proposed building security models for networked systems using paradigms like *attack graphs* [1, 11, 15, 18, 20] and *attack trees* [6, 13, 16, 17], 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 [14]. 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. In addition, the system administrator may also want to determine optimal robust solutions. These are sets of security hardening measures that have the property that even if some of the measures within a set fail, the system is still not compromised.

We believe that the problem should be addressed in a more systematic manner, utilizing the different tools of optimization at hand. A decision maker would possibly make a

better choice by successively exploring the different levels of optimization possible, rather than accepting a solution from an "off-the-shelf" optimizer. Towards this end, the current work makes four major contributions. First, we refine and formalize the notion of attack trees so as to encode the contribution of different security conditions leading to system compromise. Next, we develop a model to quantify the potential damage that can occur in a system from the attacks modeled by the system attack tree. We also quantify the security control cost incurred to implement a set of security hardening measures. Third, we model the system administrator's decision problem as three successively refined optimization problems on the attack tree model of the system. We progressively transform one problem into the next to cater to more cost-benefit information as may be required by the decision maker. Last but not the least, we discuss our thoughts and observations regarding the solutions, in particular the robust solutions identified by our optimization process, with a belief that such discussion will help the system administrator decide what methodology to adopt.

The rest of the paper is organized as follows. We discuss some of the previous works related to determining optimum security hardening measures in Section 2. Section 3 gives some background information on multi-objective optimization. In Section 4 we describe a simple network that we use to illustrate our problem formulation and solution. The attack tree model formalism and the cost model are presented in Sections 5 and 6 respectively. The three optimization problems and the evolutionary algorithm used to solve them are presented in Section 7 with results and discussion following in Section 8. Finally we conclude in Section 9.

2. RELATED WORK

Network vulnerability management has been previously addressed in a variety of ways. Noel et al. use exploit dependency graphs [14] 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. [11] on the other hand do not consider any cost for the hardening measures. Rather, their approach involve finding the minimal set of atomic attacks critical for reaching the goal and then finding the minimal set of security measures that cover the minimal 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. [10]. 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 then is to minimize the cost of implementing the security policies, as well as the weighted residual vulnerabilities. However, the authors finally scalarize the two objectives into a single objective using relative weights for the objectives.

3. BACKGROUND ON MULTI-OBJECTIVE OPTIMIZATION

In real world scenarios, often a problem is formulated to cater to several criteria or design objectives, and a decision choice to optimize these objectives is sought for. An optimum design problem must then be solved with multiple objectives and constraints taken into consideration. This type of decision making problems falls under the broad category of multi-criteria, multi-objective, or vector optimization problem.

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. In case of multi-objective optimization, there is no such concept of a single optimum solution. 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 consideration. Decision making under such situations thus require 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 to a multi-objective problem. Most multi-objective algorithms thus use the concept of dominance to compare feasible solutions.

Definition 1. Dominance and Pareto-optimal set
In a minimization problem with M objectives, a feasible solution vector $\vec{x}$ is said to dominate another feasible solution vector $\vec{y}$ if

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

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

The surface generated by the Pareto-optimal solutions in the objective space is called the Pareto-front or Pareto-surface.

For a security optimization problem like ours, concentrating on the minimization of hardening measure costs and the network damage, the dominance concept plays a crucial role in evaluating solutions. A solution which reduces one of the objectives would most likely increase the other. Dominance based comparison would identify solutions with such trade-off properties in the two objectives.

Evolutionary algorithms for multi-objective optimization (EMO) have been extensively studied and applied to a wide spectrum of real-world problems. 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. A number of algorithms have been proposed in this context [5, 7]. We employ the Non-dominated Sorting Genetic Algorithm-II (NSGA-II) [8] for the multi-objective optimization in this study. NSGA-II has gained wide popularity in the multi-objective optimization community because of its efficiency in terms of the convergence and diversity of solutions obtained.

4. A SIMPLE NETWORK MODEL

To illustrate our methodology, we consider the hypothetical network as shown in Fig. 1. 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.

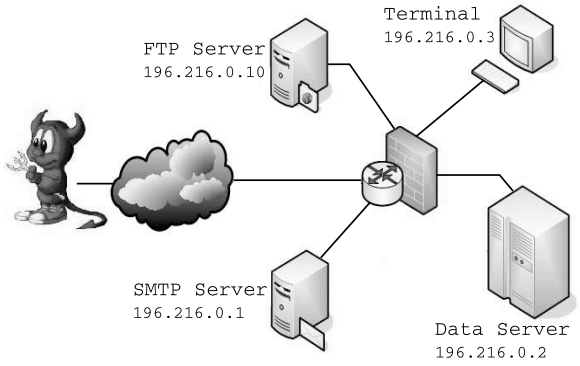

Figure 1: Example network model.

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

Table 1: Initial vulnerability per host in example network.

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. The consequence of the ftp/.rhost exploit is that it establishes a trust relation between the host and attacker machines, and introduces an

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

Table 2: Connectivity in example network.

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, as in our example network model, are represented using an *attack tree*, discussed in details in the next section.

5. ATTACK TREE MODEL

Given the complexity of today's network infrastructure, 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, 11, 14, 18] and attack tree [16, 17] 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 re-

quired 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. The representation also helps us calculate the cost factors we are interested in efficiently.

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, 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.

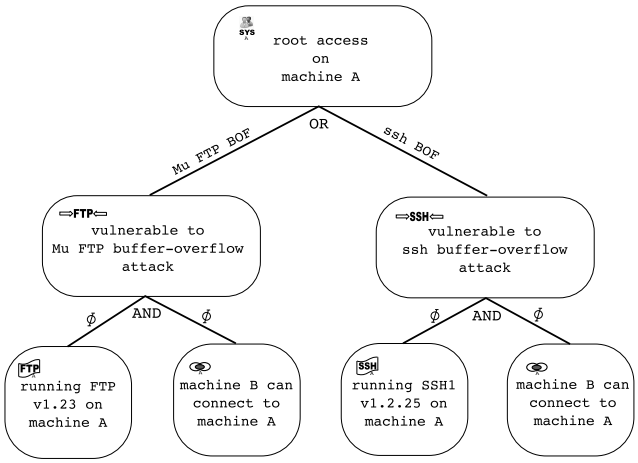

Figure 2: Example attack tree.

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 $d_j \in \{AND, OR\}$.

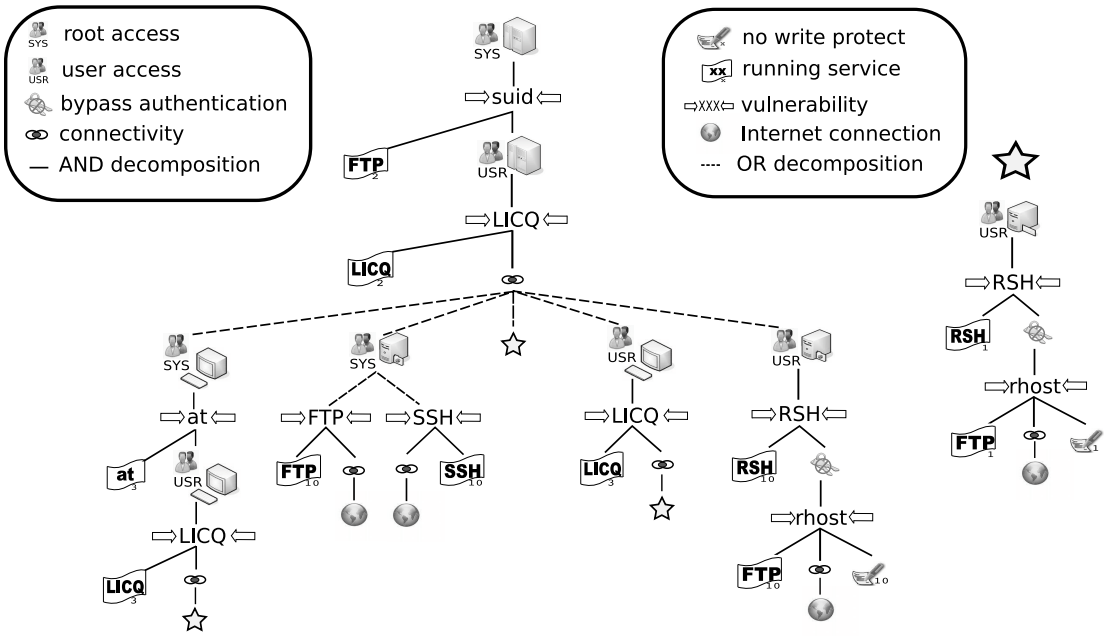

Figure 3: Attack tree of example network model.

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 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, reflect the edges in the tree. The existence of an edge between two nodes imply 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 $s_i \in N_{external}$. Fig. 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. We use an in-house tool to generate this attack tree.

6. COST MODEL

In order to defend against the attacks possible, a security manager (decision maker) can choose to implement a variety of safeguard technologies, each of which comes with different costs and coverage. For example, to defend against the ftp/.rhost exploit, one might choose to apply a security patch, disable the FTP service, or simply tighten the write protection on the .rhost directory. Each choice of action can have a different cost. Besides, some measures have multiple coverage, but with higher costs. A security manager has to make a decision and choose to implement a subset of these policies in order to maximize the resource utilization. However, given the number of permutations possible in choosing this subset (2^n for n policies), this decision is not a trivial task.

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) [19]. For simplicity, the security manager can choose to evaluate the risks by considering a relative magnitude of loss and hardening costs [2, 12, 19]. 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 [3, 4] 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.

First we define the notion of a security control in the context of the attack tree definition.

Definition 6. Security Control

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 m security controls, we use the boolean vector $\vec{T} = (T_1, T_2, \ldots, T_m)$ to indicate if a security control is chosen by a security manager. Note that the choice of this vector indirectly specifies which attributes in the attack tree would be *false* to begin with.

6.1 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.

Step1: Identify potential consequences of having a *true* value for the attribute, induced by some attack. 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}.

Step2: 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]

Step3: 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 values can be summed together under a single standard scale.

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

Step4: 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.

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

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

Definition 7. 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.

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

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.

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

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. The residual damage of the augmented tree is then defined as follows.

Definition 8. Residual Damage

Given an augmented-attack tree $(s_{root}, S, \tau, \varepsilon)|\langle I, V\rangle$ and a vector $\vec{T} = (T_i)$, $T_i \in \{0, 1\}; 1 \leq i \leq m$, the residual damage is defined as the value associated with s_{root}, i.e.,

$$RD(\vec{T}) = V_{root}$$

6.2 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 9. Total Security Control Cost

Given a set of m security controls, each having a cost $C_i; 1 \leq i \leq m$, and a vector $\vec{T} = (T_i)$, $T_i \in \{0, 1\}; 1 \leq i \leq m$, the total security control cost is defined as

$$SCC(\vec{T}) = \sum_{i=1}^{m} (T_i C_i)$$

7. PROBLEM FORMULATION

The two objectives we consider in this study are the total security control cost and the residual damage in the attack tree of our example network model. For the attack tree shown in Fig. 3, 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 also tried to 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.

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

Table 3: Security controls for example network model.

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

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

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* m *security controls, find a vector* $\vec{T^*} = (T_i^*)$, $T_i^* \in \{0, 1\}; 1 \leq i \leq m$, *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 $\vec{T} = (T_i)$ *be a boolean vector. A perturbed assignment of radius* r, $\vec{T_r}$, *is obtained by inverting the value of at most* r *elements of the vector* $\vec{T}$. *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* m *security controls, find a vector* $\vec{T^*} = (T_i^*)$, $T_i^* \in \{0, 1\}; 1 \leq i \leq m$, *which minimizes the total security control cost and the residual damage, satisfying the constraint*

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

where, 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.

We use a simple genetic algorithm (SGA) [9] to solve Problem 1. NSGA-II is used to solve Problem 2 and 3.

NSGA-II for security optimization

NSGA-II starts with a population P_0 of N randomly generated security control vectors $\vec{T}$. For each trial solution, the total security control cost is calculated using Def. 9. 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 DFS 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 Eq. 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 [9]. The residual damage and total security control cost corresponding to each solution in the child population are also computed.

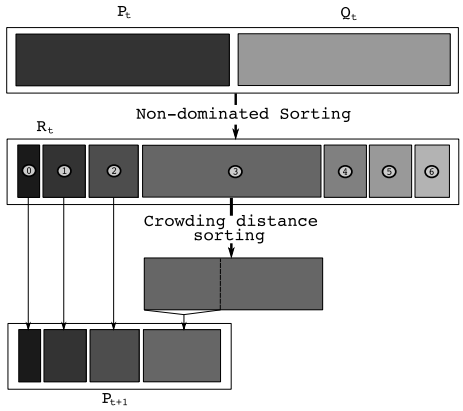

Figure 4: One generation of NSGA-II.

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 solution indicates that there are k other solutions of different ranks that dominate it. 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* [8], 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. Fig. 4 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 algorithms 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.

8. RESULTS AND DISCUSSION

We first present the sensitivity results of NSGA-II and SGA to their parameters. 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.

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. Fig. 5 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 as 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 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.

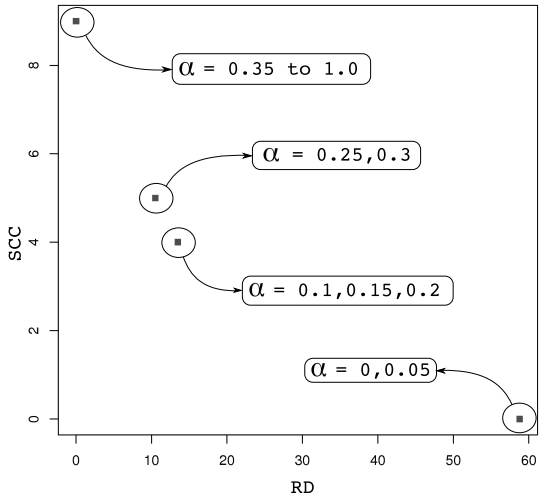

Figure 5: SGA solutions to Problem 1 with α varied from 0 to 1 in steps of 0.05.

The two solutions corresponding to $\alpha = 0.25$ and 0.1 in Fig. 5, including any other solutions 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. Fig. 6 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.

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. Fig. 6 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

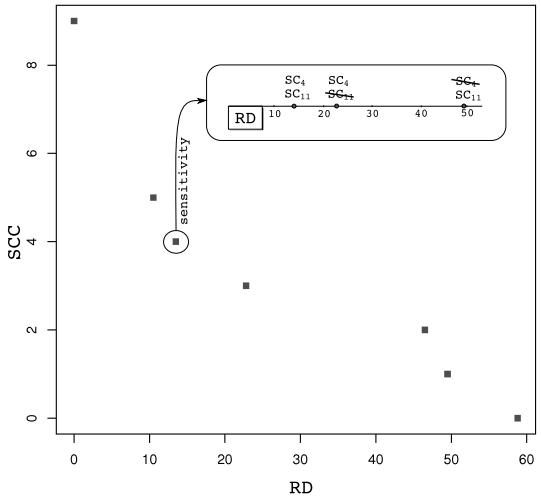

Figure 6: NSGA-II solutions to Problem 2 and sensitivity of a solution to optimum settings.

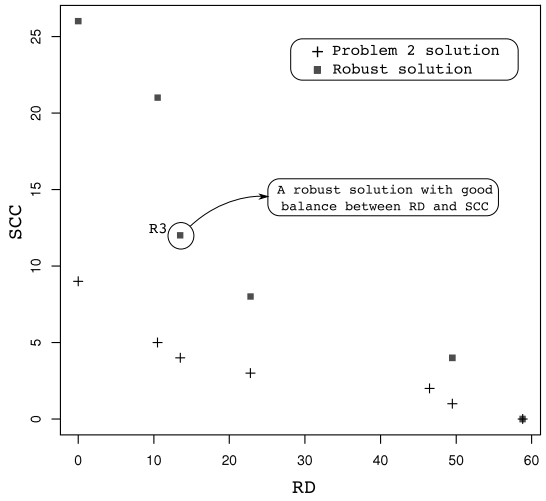

Figure 7: NSGA-II solutions to Problem 3 with $D = 0$ and $r = 1$. Problem 2 solutions are also shown for comparison.

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, we are mostly interested in obtaining solutions that are fully robust, meaning the residual damage should not increase, and hence set D to zero. Also, because of the sparse nature of the coverage matrix, we set the perturbation radius r to 1. Fig. 7 shows the solutions obtained for this problem.

	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

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

The solutions to Problem 3 reveals that none of the optimum solutions, except the trivial zero SCC solution, previously obtained 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.

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 Fig. 8. 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, is labeled with the maximum residual damage that can propagate to it 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.

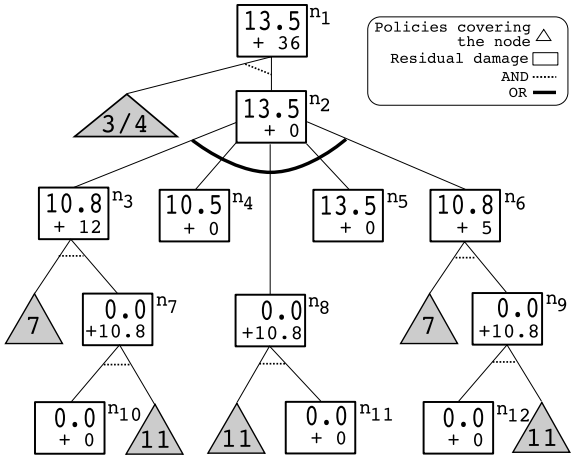

Figure 8: Compressed attack tree showing residual damage computation with R3 as security control set.

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

9. CONCLUSION AND FUTURE WORK

In this paper, we addressed the system administrator's dilemma, namely, how to select, when needed, 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.

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. In addition, we have assumed that the system administrator's decision is in no way influenced by an understanding of the cost to break the system. Furthermore, the possible decomposition of an attack tree to divide the problem into sub-problems is an interesting alternative to explore. Finally, there is a dynamic aspect to the system administrator's dilemma. During run time the system administrator may need to revise her decision based on emerging security conditions. In future we plan to refine our model to incorporate these scenarios.

10. ACKNOWLEDGMENTS

This work was partially supported by the U.S. Air Force Office of Scientific Research under contract 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.

11. REFERENCES

[1] Ammann, P., Wijesekera, D., and Kaushik, S. Scalable, Graph-Based Network Vulnerability Analysis. In *Proceedings of the Ninth Conference on Computer and Communications Security* (Washington, DC, USA, 2002), pp. 217–224.

[2] Berger, B. Data-centric Quantitative Computer Security Risk Assessment. *Information Security Reading Room, SANS* (2003).

[3] Butler, S. Security Attribute Evaluation Method: A Cost-benefit Approach. In *ICSE 2002: Proceedings of the 24rd International Conference on Software Engineering* (Orlando, FL, USA, 2002), pp. 232–240.

[4] Butler, S., and Fischbeck, P. Multi-attribute Risk Assessment. In *Proceedings of SREIS02 in conjunction of 10th IEEE International Requirements Engineering Conference* (Raleigh, NC, USA, 2002).

[5] Coello, C. A. C. An Updated Survey of GA-based Multiobjective Optimization Techniques. *ACM Computing Surveys 32*, 2 (2000), 109–143.

[6] Dawkins, J., Campbell, C., and Hale, J. Modeling Network Attacks: Extending the Attack Tree Paradigm. In *Proceedings of the Workshop on Statistical Machine Learning Techniques in Computer Intrusion Detection* (Baltimore, MD, USA, 2002), Johns Hopkins University.

[7] Deb, K. *Multi-objective Optimization Using Evolutionary Algorithms*. John Wiley & Sons Inc., 2001.

[8] Deb, K., Pratap, A., Agarwal, S., and Meyarivan, T. A Fast and Elitist Multiobjective Genetic Algorithm: NSGA–II. *IEEE Transactions on Evolutionary Computation 6*, 2 (2002), 182–197.

[9] Goldberg, D. E. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, 1989.

[10] Gupta, M., Rees, J., Chaturvedi, A., and Chi, J. Matching Information Security Vulnerabilities to Organizational Security Policies: A Genetic Algorithm Approach. *Decision Support Systems 41*, 3 (2006), 592–603.

[11] Jha, S., Sheyner, O., and Wing, J. M. Two Formal Analysis of Attack Graphs. In *Proceedings of the 15th IEEE Computer Security Foundations Workshop* (Cape Breton, Nova Scotia, Canada, 2002), pp. 49–63.

[12] Lee, W. Toward Cost-sensitive Modeling for Intrusion Detection and Response. *Journal of Computer Security 10*, 1 (2002), 5–22.

[13] Moore, A., Ellison, R., and Linger, R. Attack Modeling for Information Survivability. Technical Note CMU/SEI-2001-TN-001, Carnegie Melon University / Software Engineering Institute, March 2001.

[14] Noel, S., Jajodia, S., O'Berry, B., and Jacobs, M. Efficient Minimum-cost Network Hardening via Exploit Dependency Graphs. In *Proceedings of the 19th Annual Computer Security Applications Conference* (Las Vegas, NV, USA, 2003), pp. 86–95.

[15] Phillips, C., and Swiler, L. A Graph-Based System for Network-Vulnerability Analysis. In *Proceedings of the 1998 New Security Paradigms Workshop* (Chicago, IL, USA, 1998), pp. 71–79.

[16] Ray, I., and Poolsappasit, N. Using Attack Trees to Identify Malicious Attacks from Authorized Insiders. In *ESORICS 2005* (Milan, Italy, 2005), pp. 231–246.

[17] Schneier, B. Attack Trees. *Dr. Dobb's Journal* (1999).

[18] Sheyner, O., Haines, J., Jha, S., Lippmann, R., and Wing, J. M. Automated Generation and Analysis of Attack Graphs. In *SP 2002: Proceedings of the IEEE Symposium on Security and Privacy* (Oakland, CA, USA, 2002), pp. 273–284.

[19] Stoneburner, G., Goguen, A., and Feringa, A. Risk Management Guide for Information Technology Systems. *NIST Special Publication 800–30* (2002).

[20] Swiler, L., Phillips, C., Ellis, D., and Chakerian, S. Computer-Attack Graph Generation Tool. In *Proceedings of the DARPA Information Survivability Conference and Exposition II* (Anaheim, CA, USA, 2001), pp. 307–321.