RRE: A Game-Theoretic Intrusion Response and Recovery Engine

Saman A. Zonouz, Himanshu Khurana, William H. Sanders, and Timothy M. Yardley
University of Illinois at Urbana-Champaign
{saliari2, hkhurana, whs, yardley}@illinois.edu

Abstract

Preserving the availability and integrity of networked computing systems in the face of fast-spreading intrusions requires advances not only in detection algorithms, but also in automated response techniques. In this paper, we propose a new approach to automated response called the Response and Recovery Engine (RRE). Our engine employs a game-theoretic response strategy against adversaries modeled as opponents in a two-player Stackelberg stochastic game. RRE applies attack-response trees to analyze undesired security events and their countermeasures using Boolean logic to combine lower-level attack consequences. In addition, RRE accounts for uncertainties in intrusion detection alert notifications. RRE then chooses optimal response actions by solving a partially observable competitive Markov decision process that is automatically derived from attack-response trees. Experimental results show that RRE, using Snort's alerts, can protect large networks for which attack-response trees have more than 900 nodes.

1. Introduction

The severity and number of intrusions on computer networks are rapidly increasing. Generally, incident-handling [9] techniques are categorized into three broad classes. First, there are intrusion prevention methods that take actions to prevent occurrence of attacks, e.g., network flow encryption to prevent man-in-the-middle attacks. Second, there are intrusion detection systems (IDSes), such as Snort [22], which try to detect inappropriate, incorrect, or anomalous network activities, e.g., perceiving CrashIIS attacks by detecting malformed packet payloads. Finally, there are intrusion response techniques that take responsive actions based on received IDS alerts to stop attacks before they can cause significant damage and to ensure safety of the computing environment. So far, most research has focused on improving techniques for intrusion prevention and detection, while intrusion response usually remains a manual process performed by network administrators who are notified by IDS alerts and respond to the intrusions. This manual response process inevitably introduces some delay between notification and response, which could be easily exploited by the attacker to achieve his or her goal and significantly increase the damage [6]. Therefore, to minimize the severity of attack damage resulting from delayed response, an automated intrusion response is required that provides instantaneous response to intrusion.

During the last five years, three types of techniques aimed at enhancing automation in the intrusion response were proposed. The majority of those techniques are based on lookup tables filled with predefined mappings, e.g., (response actions, intrusion alerts) [24]. These methods allow response systems to deal with intrusions faster. However, they suffer from a lack of 1) flexibility, mainly because these systems completely ignore the intrusion cost factor; and 2) scalability, since it is infeasible to predict all the alert combinations from IDSes in a large-scale computer network. A second group of intrusion response systems (IRSes) employs a dynamic rule-based selection procedure [28] that selects response actions based on a certain attack metric, e.g., confidence or severity of attack. Finally, there has been increasing interest in developing cost-sensitive models of response selection [26]. The main objective in applying such a model is to compare intrusion damage and response cost to ensure system recovery with minimum cost without sacrificing the normal functionality of the system under attack.

In this paper, we present an automated cost-sensitive intrusion response system called the Response and Recovery Engine (RRE) that models the security battle between itself and the attacker as a multi-step, sequential, hierarchical, non-zero-sum, two-player stochastic game. In each step of the game, RRE leverages a new extended attack tree structure, called the *attack-response tree* (ART), and the received IDS alerts to evaluate various security properties of the system. ARTs provide a formal way to describe system security based on possible intrusion and response scenarios for the attacker and response engine, respectively. More importantly, ARTs enable RRE to consider inherent uncertainties in alerts received from IDSes (i.e., false positive and false negative rates) when estimating the system's security and deciding on response actions. Then, the RRE automatically converts the attack-response trees into partially observable competitive Markov decision processes that are solved to find the optimal response action against the attacker, in the sense that the maximum discounted accumulative damage that the attacker can cause later in the game is minimized. Using this game-theoretic approach, RRE adaptively adjusts its behavior according to the attacker's possible future reactions, thus preventing the attacker from causing significant damage to the system by taking an intelligently-chosen sequence of actions. To deal with security issues with different granularities, RRE's two-layer architecture consists of local engines, which reside in individual host comput-

ers, and the global engine, which resides in the response and recovery server and decides on global response actions once the system is not recoverable by the local engines. Furthermore, the hierarchical architecture improves scalability, ease of design, and performance of RRE, so that it can protect computing assets against attackers in large-scale computer networks.

The contributions of RRE are as follows. First, RRE accounts for planned adversarial behavior in which attacks occur in stages in which adversaries execute well-planned strategies and address defense measures taken by system administrators along the way. It does so by applying game theory and seeking responses that optimize on long-term gains. Second, RRE concurrently accounts for inherent uncertainties in IDS alert notifications with attack-response trees converted to partially observable Markov decision processes that compute optimal responses despite these uncertainties. This is important because IDSes today and in the near future will be unable to generate alerts that match perfectly to successful intrusions, and response techniques must therefore allow for this imperfection in order to be practical. RRE achieves the above two goals with a unified modeling approach in which game theory and Markov decision processes are combined. We demonstrate that RRE is computationally efficient for relatively large networks via prototyping and experimentation, and demonstrate that it is practical by studying commonly found critical infrastructure networks associated with the power grid. However, we believe that RRE has wide applicability to all kinds of networks.

2. Problem Formulation

We formulate the optimal response selection problem in IRSes as a decision-making function $f_{IRS} : (\mathcal{W}, \mathcal{O}, \mathcal{M}, \mathcal{G}) \rightarrow \mathcal{M}$ with the following definitions:

$\mathcal{W}$: a set of the computing assets $w \in \mathcal{W}$, e.g., a SQL server, that are to be protected by the response engine.

$\mathcal{O}$: a set of IDS alerts $o \in \mathcal{O}$ that specifically indicate an adversarial attempt to exploit the existing specific vulnerabilities of the assets, e.g., alerts from Snort [22] warning about a packet transferring the Slammer worm that exploits a buffer overflow vulnerability in a SQL server.

$\mathcal{M}$: a set of all possible response actions $m \in \mathcal{M}$, including No-OPeration (NOP), from which the intrusion response engine picks response actions. For example, an intrusion response system can respond to SQL's buffer overflow exploitation by closing its TCP connection.

$\mathcal{G}$: a set of ART graphs $g \in \mathcal{G}$ that systematically define how intrusive (responsive) scenarios about the attacker (response engine) affect system security (see § 4.1).

The rest of the paper is devoted to a solution to the response selection function, i.e., $f_{IRS} = RRE$; in other words, we will focus on how the RRE finds the optimal response action based on given input arguments.

3. RRE's High-level Architecture

Before giving theoretical design and implementation details, we provide a high-level architecture of RRE, as illustrated in Fig. 1. It has two types of decision-making engines

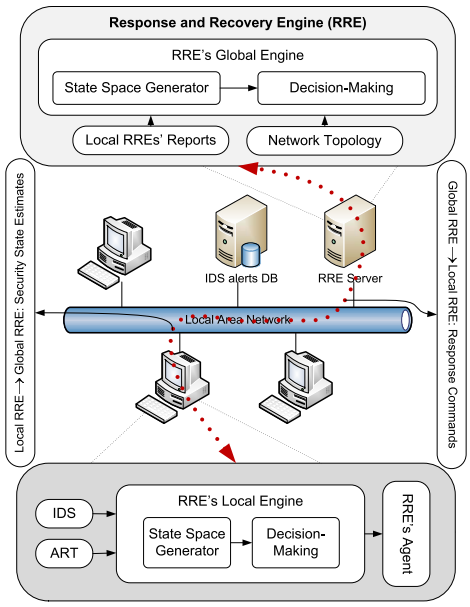


Figure 1. High-level architecture of the RRE

at two different layers, i.e., local and global. This hierarchical structure of RRE's architecture (discussed later) makes it capable of handling very frequent IDS alerts, and choosing optimal response actions. Moreover, the two-layer architecture improves its scalability for large-scale computer networks, in which RRE is supposed to protect a large number of host computers against malicious attackers. Finally, separation of high- and low-level security issues significantly simplifies the accurate design of response engines.

As the first layer, RRE's local engines are distributed in host computers. Their main inputs consist of IDS alerts and attack-response trees (described in § 4.1). All IDS alerts are sent to and stored in the alert database (Fig. 1) to which each local engine subscribes to get notified when any of the alerts related to its host computer is received. Using the mentioned local information, local engines compute local response actions and send them to RRE agents that are in charge of enforcing received commands and reporting back the accomplishment status, i.e., whether the command was successfully carried out. The internal architecture of engines includes two major components: the state space generator and the decision engine. Once inputs have been received, all possible cyber security states in which the host computer could be are generated. As discussed later in § 7, the state space might be intractably large; therefore, RRE partially generates the state space so that the decision-making unit can quickly decide on the optimal response action. The decision-making unit employs a game-theoretic algorithm that models attacker-RRE interaction as a two-player game in which each player tries to maximize his or her overall benefit. This implies that, once a system is under attack, immediate greedy response decisions are not necessarily the best choices, since they may not guarantee the minimum total accumulative cost involved in complete recovery from the attack.

Although individual local engines attempt to protect their corresponding host computers, global network-level

response actions require inputs from multiple host computers. Furthermore, individual local engines may become malicious themselves if they get compromised. To deal with these problems, RRE's global engine, as its second layer, gets high-level information from all host computers in the network, decides on optimal global response actions to take, and coordinates RRE agents to accomplish the actions by sending them relevant response commands. In addition to local security estimates from host computers, network topology is also fed into the global engine in the form of an ART graph, which shows 1) what combinations of compromised host computers will change the security status of the whole network, and 2) what global response actions are available to terminate attacks. The ARTs, in the global engine, depend on the network's topology; therefore, they should be specifically designed by experts for each network. In contrast, the ARTs for local assets, once designed, are simply re-used.

If the global engine is compromised, the ability to execute global-level response actions is affected; however, the local engines are not affected. Furthermore, global engines can be protected by employing adequate security measures, and if feasible, intrusion tolerant approaches.

4. Local Response and Recovery

Having given a high-level overview of how hierarchically structured components in RRE interact with each other, we now present the theoretical design of these components in detail. Starting with the lowest-level modules in RRE, we explain how local engines, residing in host computers, protect local computing assets using security-related information, i.e., IDS alerts, about them.

4.1. Attack-Response Tree

To protect a local computing asset, its corresponding local engine first tries to figure out what security properties of the asset have been attacked, given a received set of alerts. Attack trees [23] offer a convenient way to systematically categorize the different ways in which an asset can be attacked. Local engines make use of a new extended attack tree structure, called an attack-response tree (ART), that makes it possible 1) to incorporate possible countermeasure (response) actions against attacks, and 2) to consider intrusion detection uncertainties due to false positives and negatives in detecting successful intrusions, while estimating the current security state of the system. The attack-response trees are designed offline by experts on each computing asset, e.g., a SQL server, residing in a host computer. It is important to note that, unlike the attack tree that is designed according to all possible *attack scenarios*, the ART model is built based on the *attack consequences*, e.g., a SQL crash; thus the designer does not have to consider all possible attack scenarios that might cause those consequences.

The purpose of an attack-response tree $g_w \in \mathcal{G}$ for an asset $w \in \mathcal{W}$, e.g., a SQL server, is to define and analyze possible combinations of attack consequences that lead to violation of some security property of the asset. This security property, e.g., integrity, is assigned to the root node of the tree, which is also called the *top-event* node. In the current implementation of RRE's local engines, there

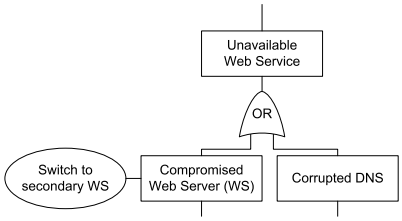


Figure 2. Node decomposition in ART

are at most three ARTs $\mathcal{G}_w = \{g_w^c, g_w^i, g_w^a\}$ for each asset w, which are typically concerned with confidentiality, integrity, and availability of assets; $\mathcal{G}_w \subset \mathcal{G}$ can be expanded to include other security properties. An attack-response tree's structure is expressed in the node hierarchy, allowing one to decompose an abstract attack goal (consequence) into a number of more concrete consequences called *sub-consequences*. A node decomposition scheme could be based on either 1) an *AND* gate, where all of the sub-consequences must happen for the abstract consequence to take place, or 2) an *OR* gate, where occurrence of any one of the sub-consequences will result in the abstract consequence. For a gate, the underlying sub-consequence(s) and the resulting abstract consequence are called *input(s)* and *output*, respectively. Being at the lowest level of abstraction in the attack-response tree structure, every leaf node consequence $l \in \mathcal{L}$ is mapped to (reported by) its related subset of IDS alerts $\mathcal{O}_l \subseteq \mathcal{O}$, each of which represents a specific vulnerability exploitation attempt by the attacker.

Some of the consequence nodes in an ART are tagged by response boxes that represent countermeasure (response) actions $m \in \mathcal{M}$ against the consequences to which they are connected (In § 4.3, we describe how these boxes are converted to response transitions, and used to decide upon optimal actions). Table 1 illustrates different classes of response actions that might be applicable to a consequence, depending on the type of consequence and the involved assets. Fig. 2 illustrates how a sample abstract consequence node (output), i.e., an unavailable web service, is decomposed into two sub-consequences (inputs) using an OR gate (i.e., the web service becomes unavailable if either the web server is compromised or the domain name server is corrupted). Furthermore, if a web service is unavailable due to the compromised web server, the response engine can switch to the secondary web server.

For every ART, a major goal is to probabilistically verify whether the security property regarding ART's root node has been violated, given the sequence of 1) the received alerts, and 2) the successfully taken response actions. Boolean values are assigned to all nodes in the attack-response tree. Each leaf node consequence $l \in \mathcal{L}$ is initially 0, and is set to 1 once any alert o_l from its corresponding alert set $\mathcal{O}_l \subseteq \mathcal{O}$ (defined earlier) is received from the IDS. These values for other consequence nodes, including the root node, are simply determined bottom-up according to leaf nodes' values in the subtree whose root is the consequence node under consideration. Response boxes are triggered once they are successfully taken by the response engine; as a result, all nodes in their subtree are reset to zero, and the corresponding received alerts are cleared. As a case

Table 1. Response and Recovery Action Classification [1]

Action class	Description	Examples
Rollback	bring the component back to a saved secure state	freeze/restore SW
Rollforward	find a new state, from which the component can operate	journal/restore/update process
Isolation	perform physical/logical exclusion of the faulty components	block attacker's IP
Reconfiguration	switch in spare component or reassign tasks to others	switch to 2nd Apache server
Reinitialization	check/record new configuration and update system tables	restart a TCP connection

in point, if the response box that is connected to ART's root node is triggered, all nodes in the ART are reset to zero.

Dealing with uncertainties. In reality, determination of Boolean values of the leaf node consequences in ART is more complicated, due to the uncertainty about whether 1) the received alerts actually represent some consequence occurrence, and 2) no consequence has happened if no alert has been received. Taking such uncertainties into account, RRE makes use of a *naive Bayes binary classifier*, similar to eBayes [27], that uses Bernouli variables, i.e., alerts, to determine the value of each leaf consequence node l, given the set of its related received alerts $\mathcal{O}_l^r \subseteq \mathcal{O}_l$:

$$\delta(l \mid \mathcal{O}_l^r) = \frac{P(l).\Pi_{o_l \in \mathcal{O}_l^r} P(o_l \mid l)}{P(l).\Pi_{o_l \in \mathcal{O}_l^r} P(o_l \mid l) + P(\bar{l}).\Pi_{o_l \in \mathcal{O}_l^r} P(o_l \mid \bar{l})} \quad (1)$$

where $P(l)$, the so-called class prior, is the probability of consequence l's occurrence, and $P(\bar{l})$ is simply its complement, i.e., $P(\bar{l}) = 1 - P(l)$. Furthermore, $P(o_l \mid l)$ denotes the probability that alert o_l was already received, given that consequence l has actually happened. These probability measures are calculated based on historical information about the system. One possible technique to obtain those measures is periodic *alert verification* [14], which is an automatic or manual, possibly time-consuming, process to periodically check whether the attack consequence l has occurred using the visible and checkable traces that a certain attack leaves at a host or on the network, e.g., a temporary file or an outgoing network connection. Consequently, $P(l)$ is calculated as a proportion of the past periodic checks that verified the occurrence of consequence l, and using Bayes' theorem:

$$P(o_l \mid l) = \frac{P(o_l, l)}{P(l)} \quad (2)$$

where $P(o_l, l)$ denotes the fraction of checks that verified that consequence l had occurred, and alert o_l had been received.

Given satisfaction probabilities of leaf nodes, the output probability of gate q with inputs $\mathcal{I}$ is simply calculated as follows:

$$\delta(q) = \begin{cases} \Pi_{i \in \mathcal{I}} \delta(i) & \text{if q is an AND gate,} \\ 1 - \Pi_{i \in \mathcal{I}} (1 - \delta(i)) & \text{otherwise} \end{cases} \quad (3)$$

where there is an implicit assumption that gate inputs are independent; otherwise, δ_q is computed using joint probability distributions of inputs. Starting from the root node and recursively using (3), it is simple to obtain δ_g, i.e., the probability that the security property of the root node in ART graph g has been compromised. This value, as a local security estimate, is reported by the local engine to the RRE server, where optimal *global* response actions are decided upon according to received local estimates (see § 5). Next, we will explain how ARTs and their nodes' satisfaction probabilities are used in a game-theoretic algorithm to decide on the optimal response action.

4.2. Stackelberg Game: RRE vs. Attacker

Reciprocal interaction between the adversary and response engine in a computer system is really a game in which each player tries to maximize his or her own benefit. The response selection process in RRE is modeled as a sequential Stackelberg stochastic game [19] in which RRE acts as the *leader* while the attacker is the *follower*; however, in our infinite-horizon game model, their roles may change without affecting the final solution to the problem.

Specifically, the game is a finite set of *security states* S that cover all possible security conditions that the system could be in. The system is in one of the security states s at each time instant. RRE, the leader, chooses and takes a response action $m_s \in \mathcal{M}$ admissible in s, which leads to a probabilistic security state transition to s'. The attacker, which is the follower, observes the action selected by the leader, and then chooses and takes an adversary action $o_{s'} \in \mathcal{O}$ admissible in s', resulting in a probabilistic state transition to s''. At each transition stage, players may receive some reward according to a reward function for each player. The reward function for an attacker is usually not known to RRE, because an attacker's reward depends on his final malicious goal, which is also not known; therefore, assuming that the attacker takes the worst possible adversary action, RRE chooses its response actions based on the security strategy, i.e., *maximin*, as discussed later. It is also important to note here that although S is a finite set, it is possible for the game to revert back to some previous state; therefore, the RRE-adversary game can theoretically continue forever. This stochastic game is essentially an antagonistic multicontroller Markov decision process, called a *competitive Markov decision process (CMDP)* [8].

A discrete competitive Markovian decision process Γ is defined as a tuple (S, A, r, P, γ) where S is the security state space, assumed to be an arbitrary non-empty set endowed with the discrete topology. A is a metric space standing for the action or control set, which itself is partitioned into response actions and adversary actions depending on the player. For every $s \in S$, $A(s) \subset A$ is the set of admissible actions at state s. The measurable function $r : K \to \Re$ is the reward where $K := \{(s, a, s') | a \in A(s); s, s' \in S\}$, and P is the transition law; that is, if the present state of the system is $s \in S$ and an action $a \in A(s)$ is taken, resulting in state transition to state s' with probability $P(s'|s, a)$, an immediate reward $r(s, a, s')$ is obtained by the player taking the action. γ is the discount factor, i.e., $0 < \gamma < 1$.

4.3. Automatic Conversion: ART-to-MDP

Using the ARTs, RRE's local engines automatically construct response decision process models, where security states are defined as a binary vector whose variables are actually the set of satisfied/unsatisfied (1/0) leaf consequence nodes in the ART under consideration. In other words, as a binary string, each security state vector represents the leaf node consequences that have already been set to 1 according to the received alerts from IDS systems. For instance, the security state space for an ART with n leaf nodes consists of 2^n n-bit state vectors. For ARTs with a large number of leaf nodes, this exponential growth of the security state space usually results in the state space explosion problem, which RRE deals with by making use of approximation techniques, as discussed in § 7.

Once local engines have generated the security state space, the next step in the decision process model generation is to construct state transitions for each state s, i.e., $A(s)$. As mentioned above, in a current security state s, there can be either of two types of actions, responsive $A_r(s)$ and adversarial $A_a(s)$, depending on the player making the decision. First, in state s, a response action $m \in \mathcal{M}$ yields a transition to state s', in which bits are all similar to those of s except for those bits that reflect leaf nodes of the ART subtree, whose root is m (explained in § 4.1), which are 0 in s'. As a clarifying example, assume that the system is in state $s \in S$, and RRE decides to enforce response action m_r, which is connected to the root node in the ART. The system's next state will be s', in which all bits are 0, i.e., the most secure state. Although it is tempting to always take m_r whenever any leaf nodes take 1, a cost-benefit evaluation (discussed later) usually results in the choice of another, cheaper response action, or in taking no action at all.

The second type of state-action-state transitions in CMDP is those due to adversarial actions. During automatic ART-to-CMDP conversion in RRE, each leaf consequence node l in the ART is mapped to an adversarial action that causes that l to be set to 1. In other words, suppose that the system is in a security state s of CMDPs. For every leaf node l whose bit in s is 0, a transition is built to state s', where all bits are the same as in s, except that the bit related to l is 1. It is observed that there is no adversary action transition from the s in which all bits are already set, i.e., the most insecure state.

The probabilities of state transition arcs $k \in K$ in CMDP are assigned based on previous actions' success rates computed using reports from local agents (see § 4.5); moreover, reward functions $r : K \rightarrow \Re$ must also be calculated. Indeed, $r(s, a, s')$ is payoff gained by the player, who is successfully taking action a in state s and causing a transition to s':

$$r(s, a, s') = (\delta_g(s) - \delta_g(s'))^{\tau_1} C(a)^{\tau_2}, \qquad (4)$$

where $0 \leq \tau_1 \leq 1$ and $\tau_2 \leq 0$ are two fixed parameters. $\delta_g(s)$ denotes the root node compromisation probability of the ART graph g whose leaf nodes' Boolean variables are set according to bits in s. This probability is simply computed using (1) and (3). Obviously, $\delta_g(s') \leq \delta_g(s)$, since a is a response action. Furthermore, $C(a)$ is the positive cost function for action a regardless of the source and destination states. This cost function should be defined specifically depending on the application for which it is used; for instance, one reasonable option would be the *mean-time-to-accomplish* measure. Consequently, (4) assigns higher rewards to less costly actions that contribute more to the overall security state.

Having automatically generated CMDP using the ART, RRE can now solve the decision process to find the optimal response action. As discussed earlier, due to a lack of knowledge about the attacker's cost function, given a system's current state, RRE uses the *maximin* approach to find the security strategy for the game. To do so, it must know the exact current state of the system. At every time instant, a reasonable choice (according to leaf nodes) is state s, where bits are 1 if their corresponding leaf nodes are set, and zero otherwise. Thus, in local engines, where leaf nodes of ARTs are mapped to subsets of IDS alerts, the system's current state s consists of 1s for bits that represent satisfied leaf nodes according to received alerts, and 0s for other bits in s. As mentioned earlier, the fact that leaf nodes have been set does not necessarily mean that they are truly positive, as in (1); hence, uncertainty in received information prevents RRE from precisely figuring out the current security state of the system. However, the probability of being in each state is calculated as follows:

$$b(s) = \prod_{l \in \mathcal{L}} (1_{[s_l=1]}.\delta(l) + 1_{[s_l=0]}.(1 - \delta(l))) \qquad (5)$$

where $\mathcal{L}$ is the set of leaf nodes in the ART; $\delta(l)$ is computed as in (1); s_l is the bit in state s that corresponds to leaf node l; and $1_{[\text{expr}]}$ is the indicator function, and is 1 if expr is true, and 0 otherwise. It is worth noting that (5) is based on the implicit assumption of independence among leaf nodes.

Therefore, to consider uncertainty, instead of determining the exact current state of the system, we obtain a probability distribution $b(.)$ on state space $s \in S$ (called the *belief state*) using (5). The axioms of probability require that $0 \leq b(s) \leq 1$ for all $s \in S$ and that $\sum_{s \in S} b(s) = 1$. Uncertainty in updating inputs, i.e., IDS alerts, converts our Markovian decision process into a higher-level model, called a *partially observable competitive Markov decision process (POCMDP)*, which is similar to the model described in [11] with the subtle difference that [11] studies simultaneous games, whereas the game here is sequential. Indeed, states $b \in \mathcal{B}$, in this higher-level model, are probability distributions over a set of states S in the underlying Markovian decision process model.

4.4. Optimal Response Strategy

As the last step in the decision-making process in local engines, RRE solves the POCMDP to find an optimal response action from its action space, and sends an action command to its agents that are in charge of enforcing received commands. Action optimization in RRE is accomplished by trying to maximize the accumulative long-run reward measure received while taking sequential response actions. To accumulate sequential achieved rewards, here, we use the *infinite-horizon discounted cost* technique [12], which gives more weight to nearer future rewards. In other words, in each step, the game value is computed by recursively adding up the immediate reward after both players

take their next actions and the discounted expected game value from then on.

To formalize the explanation just given, the solution of a POCMDP consists in computing an optimal policy, which is a function π^* that associates with any belief state $b \in \mathcal{B}$ an optimal action $\pi^*(b)$, which is an action that maximizes the expected accumulative reward on the remaining temporal horizon of the game. As discussed above, this accumulative reward is defined as the discounted sum of the local rewards r that are associated with the actual action transitions. The Markovian decision process theory assigns to every policy π a value function V_π, which associates every belief state $b \in \mathcal{B}$ with an expected global reward $V_\pi(b)$ obtained by applying π in b. For finite-horizon POMDPs, the optimal value function is piecewise-linear and convex [25], and it can be represented as a finite set of vectors. In the infinite-horizon formulation, a finite vector set can closely approximate the optimal value function V^*, whose shape remains convex. Bellman's optimality equations (6) characterize in a compact way the unique optimal value function V^*, from which an optimal policy π^*, which is discussed later, can be easily derived:

$$V^*(b) = \max_{a_r \in A_r(b)} \Psi(V^*, b, a_r) \tag{6}$$

where $A(b) = \cup_{s \in S: b(s) \neq 0} A(s)$. $A(.)$ is partitioned into $A_r(.)$ and $A_a(.)$ for response and adversary actions, respectively. Ψ is defined in (7), in which ρ is the POCMDP reward function. ρ is computed using reward function r in the inherent CMDP:

$$\rho(b, a, b') = \sum_{s, s' \in S} b(s) b'(s') r(s, a, s'). \tag{8}$$

Here, $b'_{b,a,o}$ is the updated next belief state if the current state is b, action a is taken, and observation o is received from sensors:

$$\begin{aligned} b'_{b,a,o}(s') &= P(s'|b, a, o) \\ &= \frac{P(o|s') \sum_{s \in S} P(s'|s, a) b(s)}{\sum_{s'' \in S} P(o|s'') \sum_{s \in S} P(s''|s, a) b(s)}, \end{aligned} \tag{9}$$

where, due to the independence assumption among the ART's leaf nodes, we have

$$P(o|s) = \prod_{l \in \mathcal{L}} (1_{[s_l = 1]}.P(o|l) + 1_{[s_l = 0]}.P(\bar{o}|l)), \tag{10}$$

where $P(\bar{o}|l) = 1 - P(o|l)$, and $P(o|l)$ is simply obtained using (2).

Once the partially observable decision process is formulated, the optimal response action is chosen based on the optimal value function. There are different techniques for getting the optimal value function. The decision-making unit in RRE uses a value iteration technique [3]:

$$V_t(b) = \max_{a_r \in A_r(b)} \Psi(V_{t-1}, b, a_r), \tag{11}$$

which applies dynamic programming updates to gradually improve on the value until it converges to the ϵ-optimal value function, i.e. $\mid V_t(b) - V_{t-1}(b) \mid < \epsilon$. Through improvement of the value, the policy is implicitly improved as well. Finally, optimal policy π^* maps the system's current belief state b to a response action:

$$\pi^*(b) = \arg \max_{a_r \in A_r(b)} \Psi(V^*, b, a_r), \tag{12}$$

which, in local engines, is sent to RRE agents that are in charge of carrying out the received response action commands. Agents then send status messages to the decision-making unit, indicating whether the received action command has been accomplished successfully. If it has, the decision-making unit updates the leaf nodes and variables in the corresponding ART.

So far, we have discussed how RRE's local engine estimates local security state and decides upon and takes local response actions following alerts received from the IDS. Next, we will address how RRE's server makes use of local information received from local engines to estimate the security status of the whole network, and then decide what global response actions to take. The information that is sent by local engines to RRE's server consists of root probabilities δ_g, as computed in (3), of local ARTs. In the current implementation of RRE, these include three root node probabilities of three ART trees reflecting confidentiality, integrity, and availability of local host systems.

4.5. Agents

In the abovementioned security battle between RRE and the adversary, agents play a key role in accomplishing each step of the game. They are in charge of taking response actions decided on by RRE engines. Actually, having received commands from engines, agents try to carry them out successfully and report the result, whether they were successful or not, back to the engine. If the agent's report indicates that some response action has been taken successfully, the engines update their ART trees' corresponding variables, which are leaf node values in the subtree for the successfully taken response action node. Consequently, as explained above, leaf node variables in ART trees are updated by two types of messages: IDS alerts and agents' reports. Otherwise, if the agent cannot respond successfully (e.g., within a specific amount of time), the second-best action is sent by the engine to carry out.

5. Global Response and Recovery

Although host-based intrusion response is taken into account by RRE's local engines using local ARTs and the IDS rule-set for computing assets, e.g., the SQL server, maintenance of global network-level security requires information about underlying network topology and profound understanding about what different combinations of secure assets are necessary to guarantee network security maintenance. In RRE, global network intrusion response is resolved in the global server, where, just as in local engines, ARTs are used for correlating received information, and then maximin theory is applied to choose the optimal global response action. Such a choice is not possible in local engines due to either their limited local information or their inability to manage cooperation among distributed RRE agents.

In contrast to ARTs in local engines for computing assets that demand one-time design effort for each asset (as in IDS rule-sets), global ARTs in RRE's server for network security should be designed specifically for each individual network in which RRE is deployed, since these higher-level

$$\Psi(V,b,a)=\sum_{o\in\mathcal{O}}P(o|b,a).\{\rho(b,a,b'_{b,a,o})+\sqrt{\gamma}.[\min_{a_a\in A_a(b'_{b,a,o})}\sum_{o'\in\mathcal{O}}P(o'|b'_{b,a,o},a_a).\big(\rho(b'_{b,a,o},a_a,b''_{b',a_a,o'})+\sqrt{\gamma}.V(b''_{b',a_a,o'})\big)]\} \quad (7)$$

ARTs depend on network topology, which implicitly affects a network's global security state. In our current implementation, there is only one global, i.e., network-level, ART in RRE that must be designed by an expert.

Generally, global ARTs in RRE's server have the same structure discussed in § 4.1, though some clarifying explanations are needed regarding their root and leaf nodes. As discussed in § 4.4, local engines send their local security estimates, i.e., root node probabilities δ_g of their ARTs, to RRE's server. Indeed, local security estimates contribute to leaf nodes in the global ART in RRE's server. Furthermore, the top-event node of the ART in the global engine is labeled *"network security violation,"* and is defined and formulated according to the underlying nodes. In other words, network security is defined by the global ART in RRE's server using its leaf nodes, which are themselves root nodes of local ARTs in RRE's local engines.

Global ART is employed for quantitative evaluation of a network's security state. The correlation and response selection calculations are the same as in local ARTs (§ 4), except that for global ART, the basic leaf node l probability measures are computed as $\delta(l)=\delta_{g(l)}$, where $g(l)$ denotes the local ART corresponding to leaf node l of the global ART in RRE's server.

6. Case Study: SCADA Networks

In this section, we describe the response selection process for a case study process control network for a power grid. We have chosen supervisory control and data acquisition (SCADA) networks as our case study for two reasons. First, since they control physical assets, they need timely response in the presence of attacks. Second, in contrast to general IT computer networks, they consist of computing assets with predefined specific responsibilities and communication patterns; this simplifies the design process for comprehensive ARTs and IDSes with thorough alert sets.

Fig. 3 shows a sample 3-bus power grid and its SCADA network, which is responsible for monitoring and controlling the underlying power system. There are a total of three generators, any one of which is able to provide the power required by customers, i.e., load. To monitor the power system, each bus is attached to a sensor, i.e., a phasor measurement unit (PMU). The sensor sends voltage phasors (i.e., magnitudes and phase angles) of the bus and current phasors of transmission lines connected to that particular bus to SCADA. Moreover, to control power generation, having received sensory data, SCADA computes optimal generation set points for individual generators. As shown in Fig. 3, SCADA consists of different components, among which there are constrained communications. First of all, given noisy sensory data, the state estimation server is responsible for estimating the state of the whole power system. A database stores these states and other information that might be used later by administrators or customers through the web server. The human machine interface (HMI) and security constrained optimal power flow (SCOPF) compute

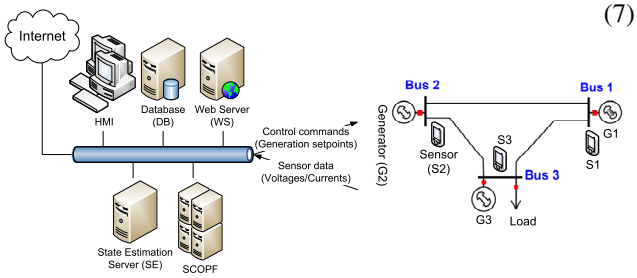


Figure 3. Case study: SCADA networks

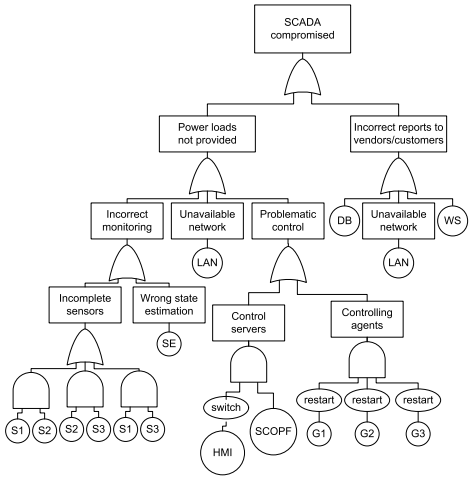


Figure 4. Attack-response tree

control commands using those estimated states. As demonstrated, a hot spare HMI is also active and connected as part of the network.

Fig. 4 illustrates a sample brief network-level ART for the process control network described above. The top event is chosen to be "SCADA is compromised," and its children denote deficiencies in providing loads and report generation, which are two main goals of the supervisory network. For simplicity, leaf nodes here denote compromise of individual host systems, and are updated by local engines. As a case in point, G1, if set to 1, indicates that the controller device for the generator on bus 3 is compromised, and the upper response node "restart" shows that a countermeasure for the compromised controller is to reinstall the control software and restart the device. Details such as action costs, rates, and probabilities are not shown.

The cyber-security state space definition for the above attack-response tree is shown in Fig. 5 as a binary vector, where each individual bit is set based on reports from local engines. For instance, the sample state vector in the figure indicates that HMI and G1 are compromised, according to reports from their corresponding local engines, while other hosts are in their normal operational mode.

Given the attack-response tree and reports from local engines, RRE starts online construction of its accompanying partially observable competitive Markov decision process. Starting from the current state, i.e., $s=(000100000010)$

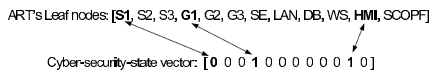


Figure 5. A sample cyber-security state

(Fig. 5), there are a total of 13 possible transitions, partitioned into two subsets: 1) response actions $A_r(s) = \{restart(G_1), switch(HMI), NOP\}$, and 2) adversarial actions $A_a(s)$ regarding leaf consequence nodes. Here, we assume that responsive actions $A_r(s)$ require some manual assistance by a SCADA operator; hence, they cannot be accomplished simultaneously due to limited human resources. The solution for this model by RRE gives *switch(HMI)* as the optimal response action, since if *restart*(G_1) or NOP was chosen, the attacker could cause a huge amount of damage to the system afterwards by compromising the SCOPF server (Fig. 4), leading to complete failure of the control subsystem that would consequently affect how power loads were provided and finally result in the top event "SCADA compromised." In other words, as explained earlier, the engine chooses the response action that minimizes the maximum damage that the attacker can cause later.

7. Computational Efficiency

Although the value iteration algorithm performs well in MDPs with several thousand states, RRE (like most state-based modeling techniques) faces the state space explosion problem when a large network that includes a large number of assets is to be protected using numerous alerts sent by distributed IDS systems. This exponential growth of the state space makes it infeasible to compute an optimal solution, i.e., response action, in large-scale applications. The problem becomes even worse when POCMDP is employed to find an optimal solution. Therefore, RRE uses two state-compaction techniques to deal with this problem.

First, the most likely state (MLS) approximation technique [5] is used to convert POMDP to MDP, which is more tractable for real-time response decision-making. To do so, we compute the most likely state using $s^* = \arg\max_s b(s)$, and define policy as $\pi(s) = \pi^{MDP}(s^*)$, which is computed using Bellman's optimality equations for the value function $V^*(s) = \max_{a_r \in A_r(s)} \Upsilon(V^*, s, a_r)$ and policy $\pi^{MDP}(s) = \arg\max_{a_r \in A_r(s)} \Upsilon(V^*, s, a_r)$, in which $\Upsilon(.)$ is as defined in (13).

The value iteration algorithm [3] is employed to compute the value function, i.e., $V(s) \leftarrow \max_{a \in A(s)} \Upsilon(V, s, a)$.

Using MLS in RRE is quite reasonable, since the probability of the most likely state is far greater than the probability of other states; however, the derived MDP is not yet small enough to deal with in real-time. Furthermore, due to its large state space, even off-line solution techniques are not usable, since most of them, e.g., value iteration, perform iterative updates over the entire state space. To focus computations on relevant states, an online *anytime algorithm*[1] called *envelope* is employed, making RRE capable of deciding real-time responses even in large-scale computer networks. In brief, the *envelope* algorithm performs a finite look-ahead search on a subset of states reachable from a given current state, i.e., s^* (mentioned above). This subset, called "envelope $\mathcal{E}_\pi$," initially contains only the current state and is progressively expanded. An approximate value function $\tilde{V}$ is used to evaluate the fringe states, i.e., the set of states that are not in the envelope but may be reached in one step from some state in the envelope:

$$\mathcal{F}_\pi = \{s \in S - \mathcal{E}_\pi | \exists s' \in \mathcal{E}_\pi, P(s', \pi(s'), s) > 0\}. \quad (14)$$

The *envelope* converges to the optimal policy [7], and its general scheme is as follows:

1) *Initialization:* Generate the initial envelope $\mathcal{E}_\pi = s^*$.

2) While ($\mathcal{E}_\pi \neq S$) and (not deadline) do

– *Fringe expansion:* Extend the envelope $\mathcal{E}_\pi$. Some $s \in \mathcal{F}_\pi$ is chosen, and its value is updated.

– *Ancestors update phase:* Generate an optimal policy π for the envelope.

3) Return π.

Using the *envelope*, RRE can solve huge MDPs very efficiently by producing partial policy, defined only on the envelope, without evaluating the entire state space.

8. Experimental Evaluation

In this section, we investigate how the proposed Response and Recovery Engine performs in reality. We have implemented RRE on top of Snort 2.7 [22], which is an open-source signature-based IDS. The experiments were run on a computer system with a 2.2 GHz AMD Athlon 64 Processor 3700+ with 1 MB of cache, 2 GB of memory, and the Ubuntu (Linux 2.6.24-21) operating system.

8.1. Scalability

To evaluate how RRE handles complex networks consisting of large numbers of host systems, we measured the time required by RRE to compute the optimal response action vs. various metrics. Fig. 6 shows the average time-to-response over ten runs vs. the attack-response tree order, i.e., the maximum number of children for each node. Given a fixed number, e.g., 500 in Fig. 6, of total nodes, the ART tree order determines the number of leaf nodes that contribute to the size of the state space in a Markovian decision model. For each tree order d, a balanced tree, in which each node has d children, is generated; gates are assigned to be AND or OR with equal probability, i.e., 0.5. In our experiments, the ϵ-optimality termination criterion in Bellman's equation and discounting factor are set to $\epsilon = 0.1$ and $\gamma = 0.99$, respectively. Then, a decision process is constructed and solved, and the total time spent is recorded (Fig. 6). As expected, the figure shows that increasing the ART order leads to rapid growth of the required time-to-response by the engine.

In another scalability evaluation experiment, we measured time-to-response vs. the number of nodes in balanced ART trees of order 2. Fig. 7 shows average results on ten runs for two schemes. First, given IDS alerts and the ART tree, the complete decision model consisting of all states in the state space was constructed. As shown in Fig. 7(a), the response engine can solve for optimal response actions for ART trees with up to 45 nodes within about 2 minutes. Second, an online finite-lookahead Markovian decision model with an expansion limit of 2 steps was generated and solved.

[1] An *anytime* algorithm can be interrupted at any point during execution to return an answer whose value, at least in certain classes of stochastic processes, improves in expectation as a function of the computation time.

$$\Upsilon(V,s,a) = \sum_{s' \in S} P(s'|s,a).\{r(s,a,s') + \sqrt{\gamma}.[\min_{a_a \in A_a(s')} \sum_{s'' \in S} P(s''|s',a_a).\big(r(s',a_a,s'') + \sqrt{\gamma}.V(s'')\big)]\} \quad (13)$$

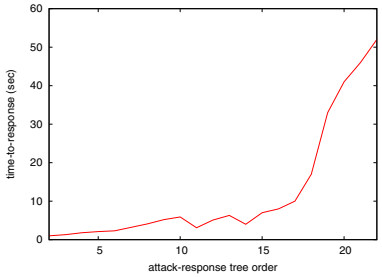


Figure 6. Tree order vs. time-to-response

As illustrated in Fig. 7(c), limited expansion improves a solution's convergence speed and increases the solvable ART size to trees with up to 900 nodes within 40 seconds. By solving ART trees with about 900 nodes in a minute, RRE can protect large-scale computer networks. Third, to further improve RRE's scalability, we evaluated how fast a decision process is solved with an upper expansion limit of 2. Fig. 7(b) compares total recovery cost between the abovementioned two schemes for all possible starting scenarios, i.e., states ($2^{|\mathcal{L}|} = 64$).

8.2. Comparison

This section evaluates how beneficial RRE is compared to static IRSes, particularly those that statically choose and take response actions from a lookup table that stores (IDS-alert, IRS-response) mappings. In our experiments, both IRS systems, i.e., RRE and static IRS, were given a sample ART tree with $\mid \mathcal{L} \mid = 6$ leaf nodes based on which they computed response actions. RRE's response action selection has already been explained in detail; the static IRS maps each alert, i.e., a leaf node in ART, to a response action that resets that particular leaf node with minimum cost. Given the current network state, we compared how much cost RRE and the static IRS spent toward the end of the game, i.e., the point at which all leaf nodes had been cleaned. ART parameters and graphs are omitted due to space constraints; however, final results are described here.

We modeled the attacker to be completely intelligent; in other words, in each step, he or she took the most harmful possible adversarial action. There were a total of $2^{|\mathcal{L}|} = 64$ starting scenarios (states) for two different game schemes. In the first scheme, the action ratio between IRS and the attacker was 1; in other words, for each action taken by the response system, the attacker was allowed to pick one adversarial action. As expected, for all initial scenarios, in picking the optimal action, RRE required a recovery cost less than or equal to what the static IRS did. In the second game scheme, we fortified the attacker's strength, and set the action ratio to 1/2, meaning that for each action by the IRS, the attacker was allowed to take 2 actions. In 5 scenarios (out of 64), RRE caused more recovery cost than its static competitor, the reason being that RRE chooses the optimal response action under the assumption that the action ratio is 1.

9. Related Work

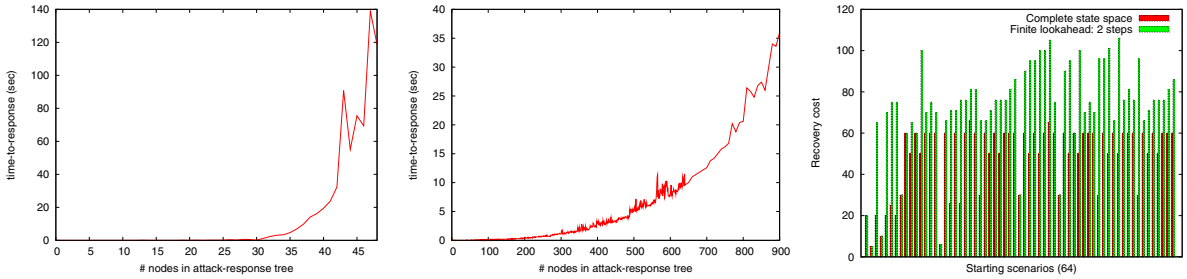
FLIPS [16], a host-based application-level firewall, uses the *selective transactional* technique to emulate selected application pieces prior to their real execution, and takes static response actions if any anomalous behavior is detected. Musman et al. [18] presented SoSMART which employs *case-based reasoning* to match the current system state to the situations previously identified as intrusive. A statically mapped response selection scheme makes FLIPS and SoSMART efficient and easy to implement, but diminishes their flexibility when deployed in a dynamically changing real-world environment.

EMERALD [20] is a dynamic cooperative response system in which the response components, in addition to analyzing IDS alerts, are able to communicate with their peers at other network layers. AAIRS [21] provides adaptation through a confidence metric associated with IDS alerts and through a success metric corresponding to response actions. Although EMERALD and AAIRS offer great infrastructure for automatic IRS, they do not attempt to balance intrusion damage and recovery cost.

αLADS [13] uses a POMDP to account for imperfect state information; however, αLADS is not applicable in general-purpose distributed systems due to its reliance on local responses and specific profile-based IDS. Balepin et al. [2] address an automated IRS that uses a resource type hierarchy tree and the *system map*. Both αLADS and the IRS in [2] can be exploited by the adversary, since neither of them takes into account the malicious attacker's potential next actions while choosing response actions.

Game theory in an IRS-related context has also been utilized in previous papers. Lye et al. [17] model the interactions between an attacker and the administrator as a two-player simultaneous game in which each player makes decisions without knowledge of the strategies being chosen by the other player; however, in reality, IDSes help administrators figure out what the attacker has done before they decide upon response actions, as in sequential games. AOAR [4], created by Bloem et al., is used to decide whether each attack should be forwarded to the administrator or taken care of by the automated response system. Use of a single-step game model makes the AOAR vulnerable to multi-step security attacks in which the attacker significantly damages the system with an intelligently chosen sequence of individually negligible adversarial actions.

ADEPTS [10] uses I-GRAPH, i.e., graphs of intrusion goals, to determine the spread of the intrusion and the appropriate response. A subtle, yet significant, difference between I-GRAPHS and the ART is that the former is designed according to intrusion scenarios, while the latter is based on consequences regardless of the attack scenarios that cause them. SARA [15] consists of several components that function as sensors, detectors, arbitrators (decision engines), and responders (response implementation); however, it does not study a particular response strategy.



(a) Complete state space generated (b) Finite lookahead: 2 steps (c) Recovery cost comparison

Figure 7. Scalability improvement and recovery cost overhead using finite lookahead solution

10. Conclusions

A game-theoretic intrusion response engine, called the *Response and Recovery Engine (RRE)*, was presented. We modeled the security maintenance of computer networks as a Stackelberg stochastic two-player game in which the attacker and response engine try to maximize their own benefits by taking optimal adversary and response actions, respectively. Using an extended attack tree structure called the *Attack-Response Tree (ART)*, RRE explicitly takes into account inaccuracies associated with IDS alerts in estimating the security state of the system. Moreover, RRE explores the intentional malicious attacker's next possible action space before deciding upon the optimal response action, so that it is guaranteed that the attacker cannot cause greater damage than what RRE predicts. Experiments show that RRE takes appropriate countermeasure actions against ongoing attacks, and brings an insecure network to its normal operational mode with the minimum possible cost.

Acknowledgments

This material is based upon work supported by the National Science Foundation under Grant No. CNS-0524695, as part of the NSF/DOE/DHS Trustworthy Cyber Infrastructure for Power Center (http://tcip.iti.illinois.edu).

References

[1] A. Avizienis, J. Laprie, B. Randell, and C. Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE Trans. on Dep. and Sec. Comp.*, 1:11–33, 2004.

[2] I. Balepin, S. Maltsev, J. Rowe, and K. Levitt. Using specification-based intrusion detection for automated response. *Proc. of the Int'l Symp. on Recent Advances in Intrusion Detection*, pages 136–54, 2003.

[3] R. Bellman. *Dynamic Programming*. Princeton University Press, 1957; republished 2003.

[4] M. Bloem, T. Alpcan, and T. Basar. Intrusion response as a resource allocation problem. *Proc. of Conf. on Decision and Control*, pages 6283–8, 2006.

[5] A. Cassandra. *Exact and Approximate Algorithms for Partially Observable Markov Decision Processes*. PhD thesis: Brown University, 1998.

[6] F. Cohen. Simulating cyber attacks, defenses, and consequences. *Journal of Comp. and Sec.*, 18:479–518, 1999.

[7] T. Dean, L. Kaelbling, J. Kirman, and A. Nicholson. Planning under time constraints in stochastic domains. *Artificial Intelligence*, 76:35–74, 1995.

[8] J. Filar and K. Vrieze. *Competitive Markov Decision Processes*. Springer-Verlag, 1997.

[9] B. Foo, M. Glause, G. Howard, Y. Wu, S. Bagchi, and E. Spafford. *Information assurance: Dependability and Security in Networked Systems*. Morgan Kaufmann, 2007.

[10] B. Foo, Y. Wu, Y. Mao, S. Bagchi, and E. Spafford. Adepts: adaptive intrusion response using attack graphs in an e-commerce environment. *Proc. of Dependable Systems and Networks*, pages 508–17, 2005.

[11] S. Hsu and A. Arapostathis. Competitive Markov decision processes with partial observation. *Proc. of IEEE lnt. Conf. on Systems, Man and Cybemetics*, 1:236–41, 2004.

[12] L. Kaelbling, M. Littman, and A. Cassandra. Partially observable Markov decision processes for artificial intelligence. *Proc. of the German Conference on Artificial Intelligence: Advances in Artificial Intelligence*, 981:1–17, 1995.

[13] O. P. Kreidl and T. M. Frazier. Feedback control applied to survivability: A host-based autonomic defense system. *IEEE Trans. on Reliability*, 53:148–66, 2004.

[14] C. Kruegel, W. Robertson, and G. Vigna. Using alert verification to identify successful intrusion attempts. *Info. Processing and Communication*, 27:220–8, 2004.

[15] S. Lewandowski, D. Hook, G. O'Leary, J. Haines, and M. Rossey. SARA: Survivable autonomic response architecture. *Proc. of the DARPA Info. Survivability Conf. and Exposition II*, 1:77–88, 2001.

[16] M. Locasto, K. Wang, A. Keromytis, and S. Stolfo. FLIPS: Hybrid adaptive intrusion prevention. *Proc. of the Symp. on Recent Advances in Intrusion Detection*, pages 82–101, 2005.

[17] K. Lye and J. Wing. Game strategies in network security. *Int'l Journal of Info. Security*, 4:71–86, 2005.

[18] S. Musman and P. Flesher. System or security managers adaptive response tool. *Proc. of the DARPA Info. Survivability Conf. and Exposition*, 2:56–68, 2000.

[19] G. Owen. *Game Theory*. Academic Press, 1995.

[20] P. Porras and P. Neumann. EMERALD: Event monitoring enabling responses to anomalous live disturbances. *Proc. of the Info. Systems Security Conf.*, pages 353–65, 1997.

[21] D. Ragsdale, C. Carver, J. Humphries, and U. Pooch. Adaptation techniques for intrusion detection and intrusion response system. *Proc. of the IEEE Int'l Conf. on Systems, Man, and Cybernetics*, pages 2344–9, 2000.

[22] R. Rehman. *Intrusion Detection Systems with Snort*. Prentice-Hall, 2003.

[23] B. Schneier. *Secrets & Lies: Digital Security in a Networked World*. John Wiley & Sons, 2000.

[24] A. Somayaji and S. Forrest. Automated response using system call delay. *Proc. of the USENIX Security Symp.*, pages 185–97, 2000.

[25] E. Sondik. *The Optimal Control of Partially Observable Markov Processes*. PhD Thesis: Standford University, 1971.

[26] N. Stakhanova, S. Basu, and J. Wong. A taxonomy of intrusion response systems. *Int'l Journal on Info. and Computer Security*, pages 169–84, 2007.

[27] A. Valdes and K. Skinner. Adaptive, model-based monitoring for cyber attack detection. *Proc. of the Recent Advances in Intrusion Detection*, pages 80–92, 2000.

[28] G. White, E. Fisch, and U. Pooch. Cooperating security managers: A peer-based intrusion detection system. *IEEE Network*, pages 20–3, 1996.